Національний технічний університет України

“Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет прикладної математики

Кафедра системного програмування і 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»

Група: КВ-31

Виконала: Галактіонова А.

Telegram: @nimfajcjdj

Оцінка:

Київ – 2025

Звіт

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).

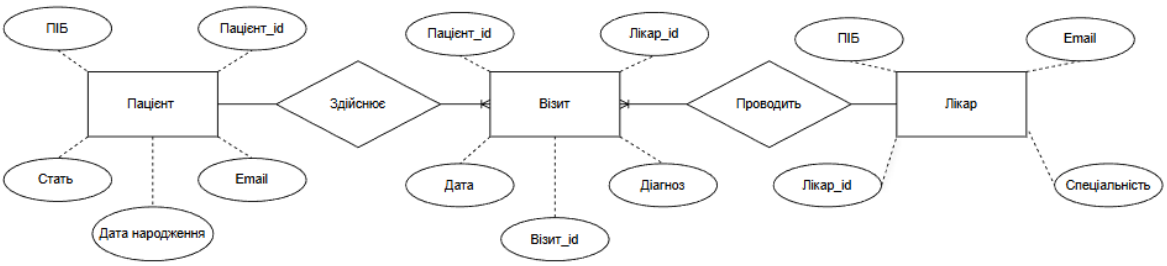
«Платформа для зберігання історій хвороб пацієнтів»

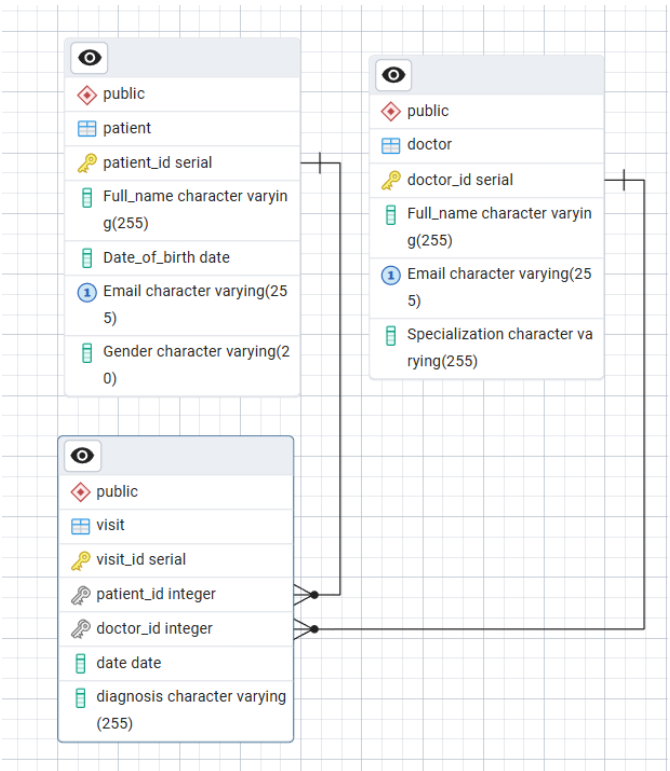
Перелік сутностей і опис їх призначення:

Patient (Пацієнт), сутність призначена для збереження даних про пацієнтів клініки – ПІБ, дата народження, стать та контактна інформація (email).
Doctor (Лікар), сутність призначена для збереження даних про лікарів, які працюють у клініці – ПІБ, спеціалізація та контактна інформація (email).
Visit (Візит), сутність призначена для збереження даних про факти візитів пацієнтів до лікарів, включаючи дату візиту та встановлений діагноз.

Опис зв'язків:
Один пацієнт може мати довільну кількість візитів до різних лікарів. Один лікар, у свою чергу, приймає багато пацієнтів та проводить багато візитів. Кожен візит однозначно пов'язує одного конкретного пацієнта з одним конкретним лікарем у певний день та фіксує діагноз.

Платформа для зберігання історій хвороб пацієнтів.
-пацієнт (ПІБ, пошта, дата народження, стать)
-лікар (ПІБ, пошта, кваліфікація)
-візиті (дата візиту, назва хвороби)





Меню користуача

У додатку реалізовано консольний інтерфейс, який надає доступ до функцій управління базою даних через головне меню. Користувач обирає номер опції для виконання відповідної дії.

Структура та опис пунктів меню:

Список лікарів
- Призначення: Перегляд даних (Read).
- Опис: Виконує запит до таблиці doctor та виводить у консоль перелік усіх лікарів із зазначенням їхнього ID, ПІБ та спеціалізації. Дозволяє перевірити наявність даних у базі.

Додати пацієнта
- Призначення: Вставка даних (Create).
- Опис: Запускає процедуру введення даних нового пацієнта (ПІБ, дата народження, пошта, стать). Після введення виконується INSERT запит до таблиці patient.

Видалити лікаря (ID)
- Призначення: Видалення даних (Delete) з контролем цілісності.
- Опис: Запитує ID лікаря для видалення. Якщо у лікаря є пов'язані записи в таблиці visit, програма перехоплює помилку бази даних (Foreign Key Violation) та повідомляє користувача про неможливість операції. Якщо записів немає — лікар видаляється.

Генерація даних
- Призначення: Автоматизація наповнення БД.
- Опис: Запитує кількість записів, які необхідно створити. Запускає SQL-скрипт, що використовує масиви даних та функцію random() для генерації реалістичних тестових даних у таблицях patient, doctor та visit.

Пошук візитів
- Призначення: Пошук та фільтрація.
- Опис: Дозволяє знайти візити за ключовими словами в діагнозі (частковий збіг) та в межах певного діапазону дат. Результат виводиться у вигляді таблиці з іменами пацієнта та лікаря.

Вихід
- Опис: Коректно завершує роботу програми та закриває з'єднання з базою даних.

Назва мови програмування та бібліотек

Для реалізації проекту було використано:

- **Мова програмування:** Python 3.10+
- **СУБД:** PostgreSQL 17
- **Використані бібліотеки:**
    - **`psycopg2-binary`**: Основний драйвер для підключення Python до PostgreSQL. Забезпечує виконання SQL-запитів та управління транзакціями.
    - **`time`**: Стандартна бібліотека для заміру часу виконання операцій (використовується для оцінки швидкодії генерації даних та пошуку).
- **Середовище розробки:** Visual Studio Code.

Пункт №1. Контроль цілісності даних (Foreign Keys)

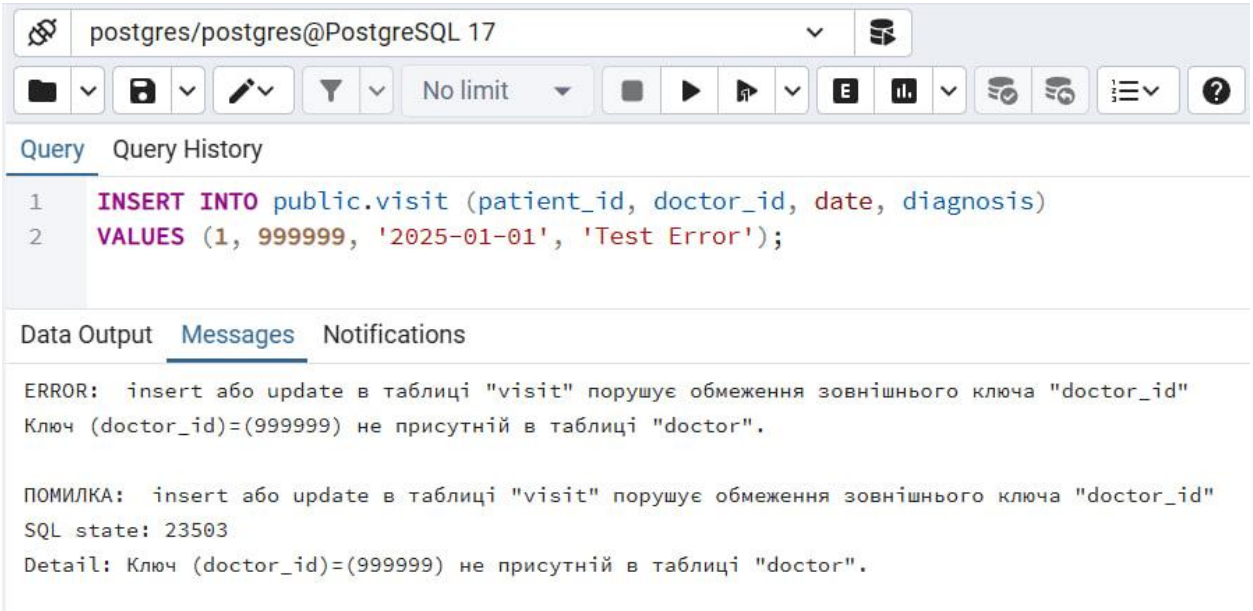
Вимога: Забезпечити контроль видалення та вставки даних у зв'язаних таблицях (відношення 1:N).

1.1. Операція вилучення запису батьківської таблиці При спробі видалити лікаря, який має заплановані візити (записи в дочірній таблиці visit), СУБД PostgreSQL блокує операцію для збереження посилальної цілісності.

```
--- MENU ---
1. Список лікарів
2. Додати пацієнта
3. Видалити лікаря (ID)
4. Генерація даних
5. Пошук візитів
0. Вихід
 > 3
Введіть ID лікаря: 34
Error: Cannot delete doctor because they have patients/visits.
```

Пояснення причини помилки: Таблиця visit містить зовнішній ключ doctor_id, який посилається на таблицю doctor. Видалення запису з doctor призвело б до того, що записи у visit посилалися б на неіснуючого лікаря, що неприпустимо в реляційній моделі даних.

1.2. Операція вставки запису в дочірню таблицю При спробі додати візит, вказавши ID лікаря або пацієнта, якого не існує в батьківських таблицях (doctor або patient), СУБД повертає помилку порушення обмеження зовнішнього ключа.



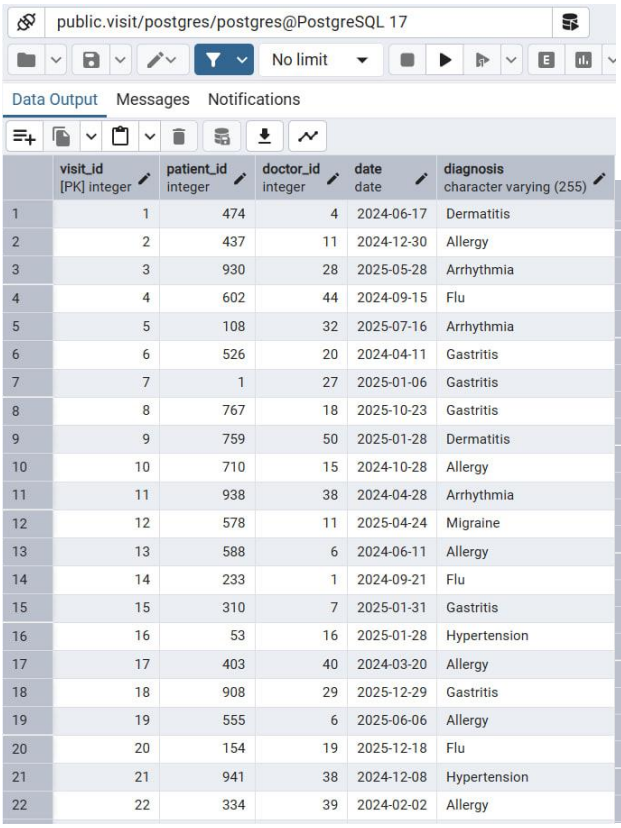
Пункт №2. Генерація даних

Вимога: Автоматична генерація великої кількості даних за допомогою SQL.

Для наповнення бази даних тестовими даними (1000 записів) розроблено SQL-запит, що використовує:

- generate_series для створення циклу вставки.

- random() та SQL-масиви для генерації реалістичних імен та діагнозів.
- Вибірку існуючих ID з батьківських таблиць для забезпечення цілісності.



public.visit/postgres/postgres@PostgreSQL 17

No limit

Data Output | Messages | Notifications

	visit_id [PK] integer	patient_id integer	doctor_id integer	date date	diagnosis character varying (255)
1	1	474	4	2024-06-17	Dermatitis
2	2	437	11	2024-12-30	Allergy
3	3	930	28	2025-05-28	Arrhythmia
4	4	602	44	2024-09-15	Flu
5	5	108	32	2025-07-16	Arrhythmia
6	6	526	20	2024-04-11	Gastritis
7	7	1	27	2025-01-06	Gastritis
8	8	767	18	2025-10-23	Gastritis
9	9	759	50	2025-01-28	Dermatitis
10	10	710	15	2024-10-28	Allergy
11	11	938	38	2024-04-28	Arrhythmia
12	12	578	11	2025-04-24	Migraine
13	13	588	6	2024-06-11	Allergy
14	14	233	1	2024-09-21	Flu
15	15	310	7	2025-01-31	Gastritis
16	16	53	16	2025-01-28	Hypertension
17	17	403	40	2024-03-20	Allergy
18	18	908	29	2025-12-29	Gastritis
19	19	555	6	2025-06-06	Allergy
20	20	154	19	2025-12-18	Flu
21	21	941	38	2024-12-08	Hypertension
22	22	334	39	2024-02-02	Allergy

	visit_id [PK] integer	patient_id integer	doctor_id integer	date date	diagnosis character varying (255)
132	132	244	6	2025-09-02	Dermatitis
133	133	88	8	2024-09-22	Migraine
134	134	96	38	2025-08-17	Gastritis
135	135	893	28	2024-06-01	Flu
136	136	176	35	2024-05-22	Flu
137	137	285	24	2025-04-08	Gastritis
138	138	945	21	2025-11-17	Dermatitis
139	139	299	25	2025-01-07	Allergy
140	140	729	8	2024-01-19	Arrhythmia
141	141	782	11	2025-08-04	Bronchitis
142	142	556	22	2025-05-21	Gastritis
143	143	4	49	2024-02-27	Gastritis
144	144	610	27	2025-01-01	Allergy
145	145	215	16	2025-02-23	Arrhythmia
146	146	825	29	2025-12-16	Allergy
147	147	25	10	2024-06-08	Dermatitis
148	148	959	18	2025-08-28	Dermatitis
149	149	88	41	2025-06-18	Migraine
150	150	361	7	2025-04-24	Bronchitis
151	151	435	22	2024-10-11	Arrhythmia
152	152	117	10	2024-06-05	Hypertension
153	153	894	12	2024-12-24	Gastritis

Копія SQL-запиту генерації (з методу generate_data):

```
# visits
sql_visit = f"""
WITH p_ids AS (SELECT array_agg(patient_id) as ids FROM public.patient),
     d_ids AS (SELECT array_agg(doctor_id) as ids FROM public.doctor)
INSERT INTO public.visit (patient_id, doctor_id, date, diagnosis)
SELECT
    p.ids[floor(random() * array_length(p.ids, 1) + 1)],
    d.ids[floor(random() * array_length(d.ids, 1) + 1)],
    timestamp '2024-01-01' + random() * (timestamp '2025-12-31' - timestamp '2024-01-01'),
    (ARRAY['Flu', 'Migraine', 'Gastritis', 'Bronchitis', 'Hypertension', 'Allergy', 'Dermatitis', 'Arrhythmia'])[floor(random()*8+1)]
FROM generate_series(1, {count}), p_ids p, d_ids d;
"""
```

Пункт №3. Пошук даних

Вимога: Пошук за декількома атрибутами з фільтрацією та виведенням часу виконання.

Реалізовано функцію пошуку візитів, яка фільтрує записи за:

1. Частиною назви діагнозу (оператор ILIKE для ігнорування регістру).
2. Діапазоном дат (оператор BETWEEN). Запит виконує об'єднання (JOIN) трьох таблиць для виведення імен замість ідентифікаторів.

```
--- Параметри пошуку ---
Діагноз (можна частину): flu
Дата з (2020-01-01): 2020-01-01
Дата по (2030-12-31): 2030-01-01

Результат: 100 записів (Час: 23.80 мс)
----------------------------------------
Дата: 2024-06-13 | Yulia Tkachenko -> Dr. Zhivago
Діагноз: Flu
----------------------------------------
Дата: 2024-08-02 | Maria Petrenko -> Dr. Watson
Діагноз: Flu
----------------------------------------
Дата: 2025-04-29 | Oleg Boyko -> Dr. House
Діагноз: Flu
----------------------------------------
Дата: 2025-02-24 | Ivan Petrenko -> Dr. Zhivago
Діагноз: Flu
----------------------------------------
Дата: 2025-11-18 | Ivan Petrenko -> Dr. Dolittle
Діагноз: Flu
----------------------------------------
Дата: 2025-02-24 | Ivan Tkachenko -> Dr. Who
Діагноз: Flu
----------------------------------------
Дата: 2024-05-18 | Petro Bondar -> Dr. Who
Діагноз: Flu
----------------------------------------
```

Копія SQL-запиту пошуку (з методу `search_visits`):

```python
def search_visits(self, diag, d_from, d_to):
    cursor = self.connection.cursor()
    t_start = time.time()
    try:
        sql = """
            SELECT v.visit_id, v.date, v.diagnosis, p."Full_name", d."Full_name"
            FROM public.visit v
            JOIN public.patient p ON v.patient_id = p.patient_id
            JOIN public.doctor d ON v.doctor_id = d.doctor_id
            WHERE v.diagnosis ILIKE %s
            AND v.date BETWEEN %s AND %s
            LIMIT 100;
        """
```

Пункт №4. Програмний код модуля "Model"

Вимога: Ілюстрація коду згідно з шаблоном MVC та опис функцій.

```python
import psycopg2
from psycopg2 import Error
import time
from config import DB_CONFIG

class Model:
    def __init__(self):
        self.connection = None
        try:
            self.connection = psycopg2.connect(**DB_CONFIG)
            self.connection.autocommit = True
```

```
            print("Connected to DB")
        except (Exception, Error) as error:
            print(f"Error connecting to PostgreSQL: {error}")

    def __del__(self):
        if self.connection:
            self.connection.close()
            print("Connection closed")

    def generate_data(self, count):
        start_time = time.time()
        cursor = self.connection.cursor()
        try:
            print(f"Generating {count} rows...")

            # patients
            sql_pat = f"""
            INSERT INTO public.patient ("Full_name", "Date_of_birth", "Email",
"Gender")
            SELECT
                (ARRAY['Ivan', 'Petro', 'Oleg', 'Andriy', 'Mykola', 'Maria',
'Olga', 'Anna', 'Yulia', 'Natali'])[floor(random()*10+1)]
                || ' ' ||
                (ARRAY['Ivanov', 'Petrenko', 'Kovalenko', 'Bondar', 'Tkachenko',
'Shevchenko', 'Boyko', 'Melnyk'])[floor(random()*8+1)],
                timestamp '1950-01-01' + random() * (timestamp '2010-01-01' -
timestamp '1950-01-01'),
                'user_' || md5(random()::text)::varchar(5) || '_' ||
generate_series || '@gmail.com',
                CASE WHEN random() > 0.5 THEN 'female' ELSE 'male' END
            FROM generate_series(1, {count});
            """
            cursor.execute(sql_pat)

            # doctors
            doc_count = max(5, count // 20)
            sql_doc = f"""
            INSERT INTO public.doctor ("Full_name", "Email", "Specialization")
            SELECT
                'Dr. ' || (ARRAY['House', 'Watson', 'Strange', 'Dolittle',
'Zhivago', 'Who'])[floor(random()*6+1)],
                'doctor_' || generate_series || '_' || trunc(random()*1000)::text
|| '@clinic.com',
                (ARRAY['Cardiologist', 'Dermatologist', 'Neurologist',
'Therapist', 'Surgeon'])[floor(random() * 5 + 1)]
            FROM generate_series(1, {doc_count});
            """
            cursor.execute(sql_doc)

            # visits
            sql_visit = f"""
```

```
            WITH p_ids AS (SELECT array_agg(patient_id) as ids FROM
public.patient),
                 d_ids AS (SELECT array_agg(doctor_id) as ids FROM public.doctor)
            INSERT INTO public.visit (patient_id, doctor_id, date, diagnosis)
            SELECT
                p.ids[floor(random() * array_length(p.ids, 1) + 1)],
                d.ids[floor(random() * array_length(d.ids, 1) + 1)],
                timestamp '2024-01-01' + random() * (timestamp '2025-12-31' -
timestamp '2024-01-01'),
                (ARRAY['Flu', 'Migraine', 'Gastritis', 'Bronchitis',
'Hypertension', 'Allergy', 'Dermatitis', 'Arrhythmia'])[floor(random()*8+1)]
            FROM generate_series(1, {count}), p_ids p, d_ids d;
            """
            cursor.execute(sql_visit)

            print(f"Done! Time: {time.time() - start_time:.2f} sec.")

        except (Exception, Error) as error:
            print(f"Gen error: {error}")
        finally:
            cursor.close()

    def search_visits(self, diag, d_from, d_to):
        cursor = self.connection.cursor()
        t_start = time.time()
        try:
            sql = """
                SELECT v.visit_id, v.date, v.diagnosis, p."Full_name",
d."Full_name"
                FROM public.visit v
                JOIN public.patient p ON v.patient_id = p.patient_id
                JOIN public.doctor d ON v.doctor_id = d.doctor_id
                WHERE v.diagnosis ILIKE %s
                AND v.date BETWEEN %s AND %s
                LIMIT 100;
            """
            cursor.execute(sql, (f'%{diag}%', d_from, d_to))
            res = cursor.fetchall()
            return res, (time.time() - t_start) * 1000
        except (Exception, Error) as error:
            print(f"Search error: {error}")
            return [], 0
        finally:
            cursor.close()

    def get_all_doctors(self):
        cursor = self.connection.cursor()
        try:
            cursor.execute('SELECT doctor_id, "Full_name", "Specialization" FROM
public.doctor ORDER BY doctor_id DESC LIMIT 20')
            return cursor.fetchall()
```

```
        finally:
            cursor.close()

    def add_patient(self, name, dob, email, gender):
        cursor = self.connection.cursor()
        try:
            sql = 'INSERT INTO public.patient ("Full_name", "Date_of_birth",
"Email", "Gender") VALUES (%s, %s, %s, %s)'
            cursor.execute(sql, (name, dob, email, gender))
            print("Patient added.")
        except Error as e:
            print(f"DB Error: {e}")
        finally:
            cursor.close()

    def delete_doctor(self, doc_id):
        cursor = self.connection.cursor()
        try:
            cursor.execute('DELETE FROM public.doctor WHERE doctor_id = %s',
(doc_id,))
            if cursor.rowcount > 0:
                print(f"Doctor {doc_id} deleted.")
            else:
                print("Not found.")
        except psycopg2.errors.ForeignKeyViolation:
            print("Error: Cannot delete doctor because they have
patients/visits.")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
```

Короткий опис функцій класу Model:

1. **__init__(self)**: Ініціалізує з'єднання з базою даних PostgreSQL, використовуючи параметри з файлу конфігурації. Встановлює режим autocommit.
2. **generate_data(self, count)**: Виконує пакетну вставку даних у таблиці patient, doctor та visit. Використовує оптимізовані SQL-запити для генерації великої кількості записів (test case: 1000+ рядків).
3. **search_visits(self, diag, d_from, d_to)**: Здійснює пошук візитів за введеними критеріями (діагноз, дати). Повертає список знайдених записів та час виконання запиту в мілісекундах.
4. **add_patient(self, name, dob, email, gender)**: Виконує запит INSERT для додавання нового пацієнта в базу даних з переданими параметрами.
5. **delete_doctor(self, doc_id)**: Виконує запит DELETE для видалення лікаря за ID. Обробляє виняток ForeignKeyViolation, якщо видалення неможливе через наявність залежних даних.
6. **get_all_doctors(self)**: Допоміжна функція для отримання списку лікарів (Read operation), що використовується для виведення інформації у View.

Висновок :

У ході виконання розрахунково-графічної роботи було розроблено консольний додаток мовою Python для взаємодії з системою управління базами даних PostgreSQL. Програмний код організовано згідно з архітектурним патерном MVC, що дозволило чітко відокремити логіку роботи з даними від інтерфейсу користувача та покращити структуру проекту.

Особливу увагу в роботі було приділено оптимізації взаємодії з даними. Замість повільного додавання записів через інтерпретатор Python, реалізовано механізм автоматичної генерації великого обсягу тестових даних безпосередньо через SQL-запити з використанням функцій generate_series та масивів. Це забезпечило миттєве наповнення бази даних тисячами реалістичних записів. Також було розроблено функціонал для складного пошуку інформації, який включає об'єднання кількох таблиць та фільтрацію за різними критеріями.

Крім того, реалізовано надійний контроль цілісності даних на рівні додатку. Програма коректно обробляє виняткові ситуації, пов'язані з обмеженнями зовнішніх ключів, та запобігає випадковому видаленню зв'язаних даних, що забезпечує стабільну роботу системи. Результати тестування підтвердили повну працездатність усіх модулів та їх відповідність поставленому завданню.