Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут» ім. Ігоря Сікорського

Розрахунково-графічна робота

з дисципліни «Бази Даних»

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

Виконав студент групи: КВ-32

Косарук Захар

Варіант: Медична система для збереження даних пацієнтів

Репозиторій на GitHub: https://github.com/zZaKko96/Medicine-Database.git

Telegram: https://t.me/zZaKko

Київ 2025

Рисунок 1. Діаграма сутність-зв’язок.

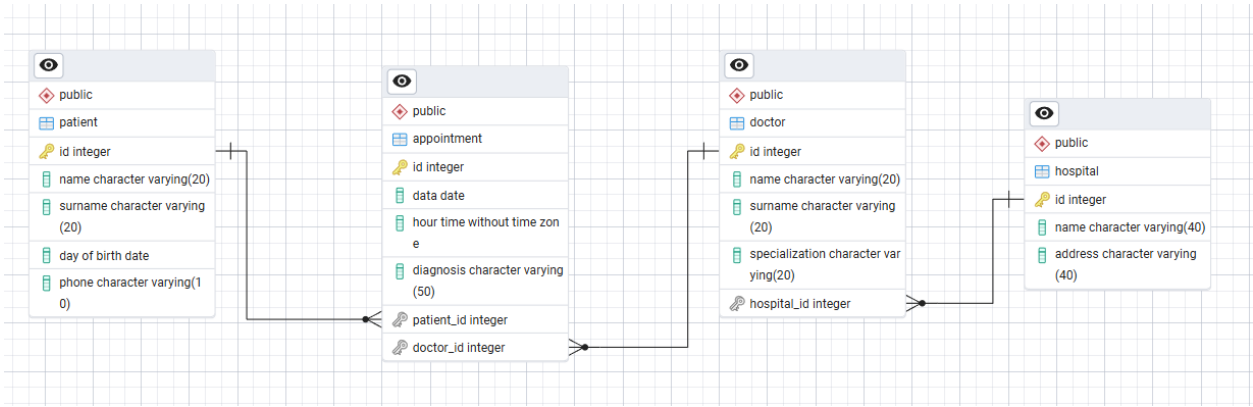


Рисунок 2. Структура бази даних.

Опис бази даних

Пацієнт – сутність для запису даних пацієнтів (ім’я, прізвище, дата народження, контактний телефон)

Лікар – сутність для запису даних лікарів (ім’я, прізвище, спеціалізація, лікарня)

Прийом – сутність для запису існуючих прийомів між пацієнтом і лікарем (дата, час, діагноз, id пацієнта (який записаний на прийом), id лікаря (який веде прийом))

Лікарня – сутність для запису даних про лікарні (назва, адреса)

Схема меню користувача

```
ГОЛОВНЕ МЕНЮ
 |
 +-- 1. Додати
 |  |
 |  +-- 1. Додати пацієнта
 |  +-- 2. Додати лікаря
 |  +-- ...
 |  \-- 0. Вихід
 |
 +-- 2. Видалити
 |  |
 |  +-- 1. Видалити пацієнта
 |  \-- 0. Вихід
 |
 +-- 3. Переглянути
 |  |
 |  +-- 1. Переглянути таблицю Пацієнт
 |  \-- 0. Вихід
 |
 +-- 4. Редагувати
 |  |
 |  +-- 1. Редагувати таблицю Пацієнт
 |  \-- 0. Вихід
 |
 +-- 5. Запустити генерацію
 +-- 6. Очистити базу даних
 +-- 7. Пошук 1
 +-- 8. Пошук 2
 +-- 9. Пошук 3
 \-- 0. Вихід
```

Головне меню

- **1. Додати**
    - Переводить користувача у підменю для створення нових записів у базі даних (пацієнтів, лікарів, лікарень, прийомів).
- **2. Видалити**
    - Переводить у підменю для видалення існуючих записів. Система включає перевірку обмежень цілісності (наприклад, неможливо видалити лікарню, якщо за нею закріплені лікарі).

- **3. Переглянути**
    - Переводить у підменю для перегляду вмісту таблиць бази даних.
- **4. Редагувати**
    - Переводить у підменю для оновлення даних в існуючих записах.
- **5. Запустити генерацію 'рандомних' даних**
    - Викликає GenerateDataAsync. Запитує у користувача кількість записів та виконує один складний SQL-запит для заповнення всіх таблиць реалістичними, випадковими даними, коректно обробляючи зовнішні ключі.
- **6. Очистити усю базу даних**
    - Викликає ClearDataAsync. Після підтвердження користувачем (ТАК), виконує TRUNCATE ... RESTART IDENTITY CASCADE для повного очищення всіх таблиць та скидання лічильників id до 1.
- **7. Пошук пацієнтів за прізвищем лікаря та датою**
    - Викликає SearchPatientsAsync. Запитує прізвище лікаря та діапазон дат. Виконує SQL-запит з JOIN, WHERE та GROUP BY, показуючи, скільки прийомів мав кожен пацієнт у цього лікаря. Виводить час виконання запиту в мс.
- **8. Вивести статистику по лікарях**
    - Викликає SearchDoctorStatisticsAsync. Запитує спеціалізацію та діапазон дат. Виконує SQL-запит з LEFT JOIN, WHERE, GROUP BY та COUNT, показуючи, скільки прийомів провів кожен лікар. Виводить час виконання запиту в мс.
- **9. Вивести статистику по лікарнях**
    - Викликає SearchHospitalStatisticsAsync. Запитує адресу та діагноз. Виконує SQL-запит з JOIN, WHERE, GROUP BY та COUNT(DISTINCT), показуючи, скільки унікальних пацієнтів з цим діагнозом прийняла кожна лікарня. Виводить час виконання запиту в мс.
- **0. Вихід**
    - **Опис:** Завершує головний цикл у Controller.cs та коректно закриває програму.

Підменю

Меню "Додати" (Пункт 1)

- **1. Додати пацієнта:** Створює новий запис у таблиці patient.
- **2. Додати лікаря:** Створює новий запис у doctor. Перевіряє існування hospital_id.
- **3. Додати прийом:** Створює новий запис у appointment. Перевіряє існування patient_id та doctor_id.
- **4. Додати лікарню:** Створює новий запис у таблиці hospital.
- **0. Вихід:** Повертає до головного меню.

Меню "Видалити" (Пункт 2)

- **1. Видалити пацієнта:** Видаляє пацієнта за id. Блокує видалення, якщо у пацієнта є прийоми.
- **2. Видалити лікаря:** Видаляє лікаря за id. Блокує видалення, якщо у лікаря є прийоми.
- **3. Видалити прийом:** Видаляє прийом за id.
- **4. Видалити лікарню:** Видаляє лікарню за id. Блокує видалення, якщо за лікарнею закріплені лікарі.
- **0. Вихід:** Повертає до головного меню.

Меню "Переглянути" (Пункт 3)

- **1. Переглянути таблицю Пацієнт:** Виконує SELECT * FROM patient та виводить список.
- **2. Переглянути таблицю Лікар:** Виконує SELECT * FROM doctor та виводить список.
- **3. Переглянути таблицю Прийом:** Виконує SELECT * FROM appointment та виводить список.
- **4. Переглянути таблицю Лікарня:** Виконує SELECT * FROM hospital та виводить список.
- **0. Вихід:** Повертає до головного меню.

Меню "Редагувати" (Пункт 4)

- **1. Редагувати таблицю Пацієнт:** Запитує id, завантажує поточні дані, дозволяє користувачу ввести нові значення та виконує UPDATE.
- **2. Редагувати таблицю Лікар:** Аналогічно до пацієнта, оновлює дані лікаря.

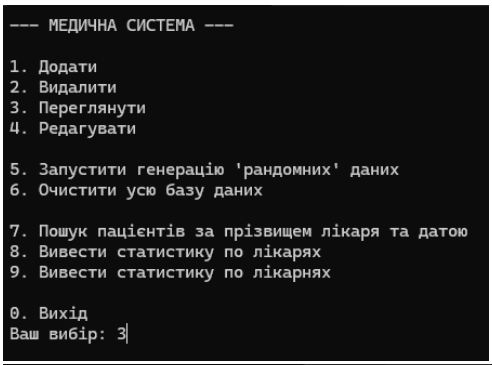
- **3. Редагувати таблицю Прийом:** Аналогічно, оновлює дані прийому.
- **4. Редагувати таблицю Лікарня:** Аналогічно, оновлює дані лікарні.
- **0. Вихід:** Повертає до головного меню.

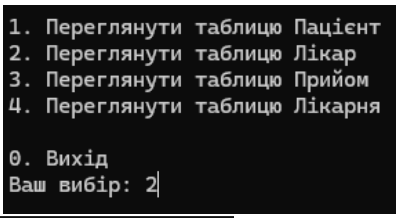
Мова програмування та використані бібліотеки

- Мова програмування: C#
- Платформа: .NET 8
- Середовище розробки: Visual Studio 2022
- Система управління базами даних (СУБД): PostgreSQL
- Бібліотека для взаємодії з БД: Npgsql

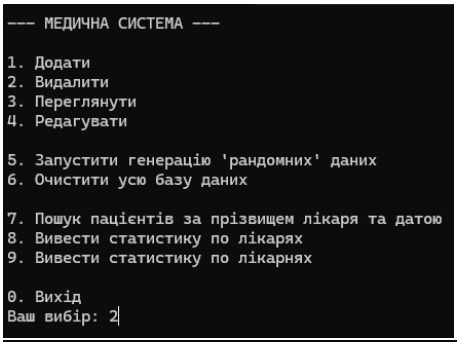
Деталізоване завдання №1

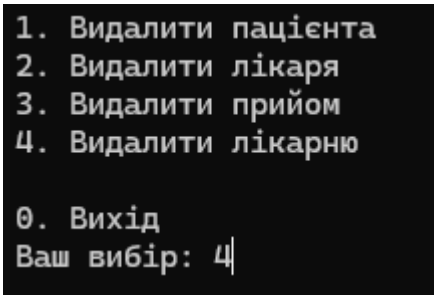
А. Невдале видалення:

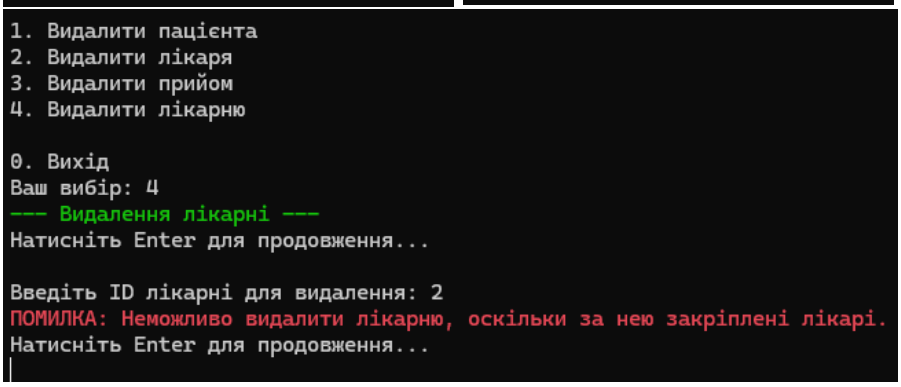




Вміст дочірньої таблиці doctor до спроби видалення.

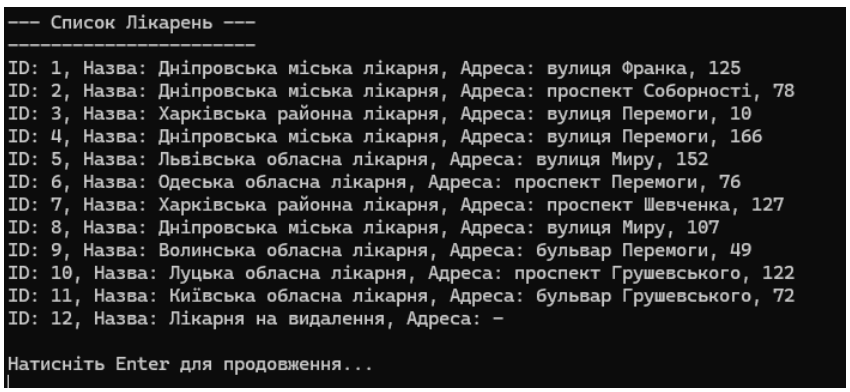






Результат перехоплення помилки при спробі видалення батьківського запису (Лікарня ID=2)

Б. Успішне видалення



Вміст таблиці hospital до успішного видалення

```
1. Видалити пацієнта
2. Видалити лікаря
3. Видалити прийом
4. Видалити лікарню

0. Вихід
Ваш вибір: 4
--- Видалення лікарні ---
Натисніть Enter для продовження...

Введіть ID лікарні для видалення: 12
Лікарню успішно видалено.
Натисніть Enter для продовження...
```

```
--- Список Лікарень ---
-------------------------
ID: 1, Назва: Дніпровська міська лікарня, Адреса: вулиця Франка, 125
ID: 2, Назва: Дніпровська міська лікарня, Адреса: проспект Соборності, 78
ID: 3, Назва: Харківська районна лікарня, Адреса: вулиця Перемоги, 10
ID: 4, Назва: Дніпровська міська лікарня, Адреса: вулиця Перемоги, 166
ID: 5, Назва: Львівська обласна лікарня, Адреса: вулиця Миру, 152
ID: 6, Назва: Одеська обласна лікарня, Адреса: проспект Перемоги, 76
ID: 7, Назва: Харківська районна лікарня, Адреса: проспект Шевченка, 127
ID: 8, Назва: Дніпровська міська лікарня, Адреса: вулиця Миру, 107
ID: 9, Назва: Волинська обласна лікарня, Адреса: бульвар Перемоги, 49
ID: 10, Назва: Луцька обласна лікарня, Адреса: проспект Грушевського, 122
ID: 11, Назва: Київська обласна лікарня, Адреса: бульвар Грушевського, 72

Натисніть Enter для продовження...
```

```
--- Список Лікарів ---
----------------------
ID: 1, Ім'я: Олег Петров, Спец: Педіатр, ID Лікарні: 2
ID: 2, Ім'я: Тарас Захаров, Спец: Кардіолог, ID Лікарні: 3
ID: 3, Ім'я: Володимир Попова, Спец: Хірург, ID Лікарні: 8
ID: 4, Ім'я: Володимир Петров, Спец: Хірург, ID Лікарні: 4
ID: 5, Ім'я: Володимир Гончарук, Спец: Терапевт, ID Лікарні: 3
ID: 6, Ім'я: Володимир Павлюк, Спец: Невролог, ID Лікарні: 10
ID: 7, Ім'я: Наталія Давидова, Спец: Офтальмолог, ID Лікарні: 3
ID: 8, Ім'я: Володимир Павлюк, Спец: Невролог, ID Лікарні: 8
ID: 9, Ім'я: Олег Лисенко, Спец: Офтальмолог, ID Лікарні: 10
ID: 10, Ім'я: Максим Петров, Спец: Хірург, ID Лікарні: 2
ID: 11, Ім'я: Олег Ковальчук, Спец: Педіатр, ID Лікарні: 4
ID: 12, Ім'я: Ірина Попова, Спец: Невролог, ID Лікарні: 1
ID: 13, Ім'я: Ігор Мартинюк, Спец: Педіатр, ID Лікарні: 3
ID: 14, Ім'я: Назар Мартинюк, Спец: Невролог, ID Лікарні: 8
ID: 15, Ім'я: Ольга Попова, Спец: Педіатр, ID Лікарні: 8
ID: 16, Ім'я: Наталія Павлюк, Спец: Терапевт, ID Лікарні: 8
ID: 17, Ім'я: Максим Гончарук, Спец: Хірург, ID Лікарні: 5
ID: 18, Ім'я: Ольга Лисенко, Спец: Кардіолог, ID Лікарні: 9
ID: 19, Ім'я: Тарас Іваненко, Спец: Хірург, ID Лікарні: 5
ID: 20, Ім'я: Світлана Захаров, Спец: Офтальмолог, ID Лікарні: 4
ID: 21, Ім'я: Тарас Павлюк, Спец: Проктолог, ID Лікарні: 1

Натисніть Enter для продовження...
```

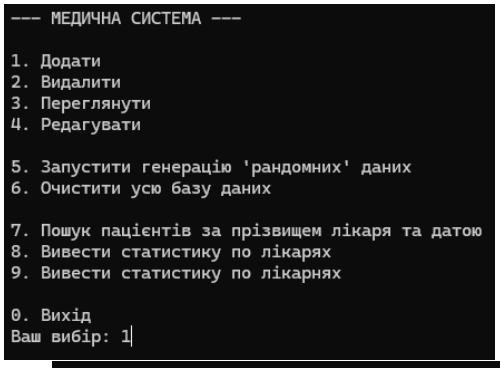
Вміст дочірньої таблиці doctor після успішного видалення hospital (ID=12). Таблиця не змінилася, оскільки до видаленої лікарні не були прив'язані лікарі

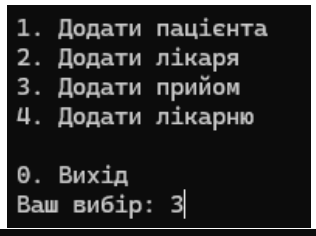
```
public async Task<string> DeleteHospitalAsync(int hospitalId)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "DELETE FROM hospital WHERE id = @id";
        await using var cmd = new NpgsqlCommand(sql, conn);
        cmd.Parameters.AddWithValue("id", hospitalId);
        await cmd.ExecuteNonQueryAsync();
        return "Лікарню успішно видалено.";
    }
    catch (PostgresException ex) when (ex.SqlState == "23503")
    {
        return "ПОМИЛКА: Неможливо видалити лікарню, оскільки за нею закріплені
лікарі.";
    }
    catch (Exception ex)
```

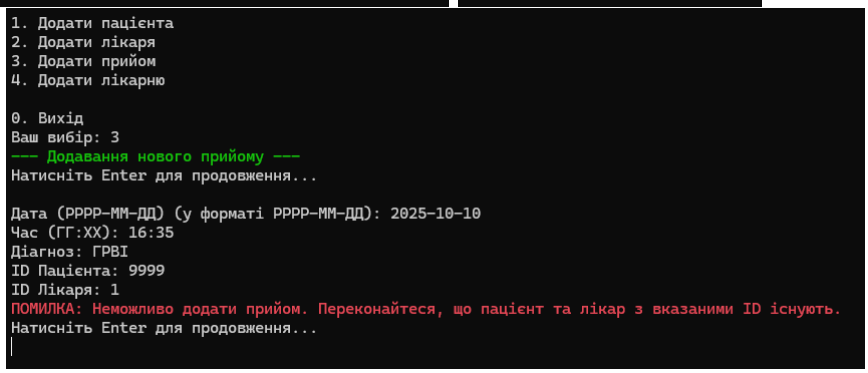
```
    {
        return $"Загальна помилка: {ex.Message}";
    }
}
```

Причина помилки полягає у спрацьовуванні обмеження зовнішнього ключа (foreign key constraint) на рівні СУБД PostgreSQL. У таблиці doctor стовпець hospital_id є зовнішнім ключем, який посилається на id в таблиці hospital. База даних не дозволяє видалити "батьківський" запис (hospital.id = 2), доки існують "дочірні" записи (doctor), які на нього посилаються.

Програмно ця помилка була коректно оброблена. Замість аварійного завершення, програма перехопила виняток Npgsql.PostgresException, ідентифікувала його за кодом ex.SqlState == "23503" (foreign_key_violation) та вивела користувачу відповідне повідомлення про неможливість видалення.







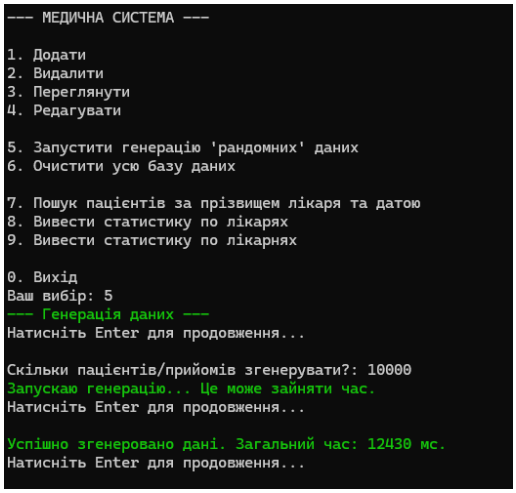
Результат перехоплення помилки при спробі вставки запису в дочірню таблицю (appointment) з неіснуючим зовнішнім ключем (patient_id=9999).

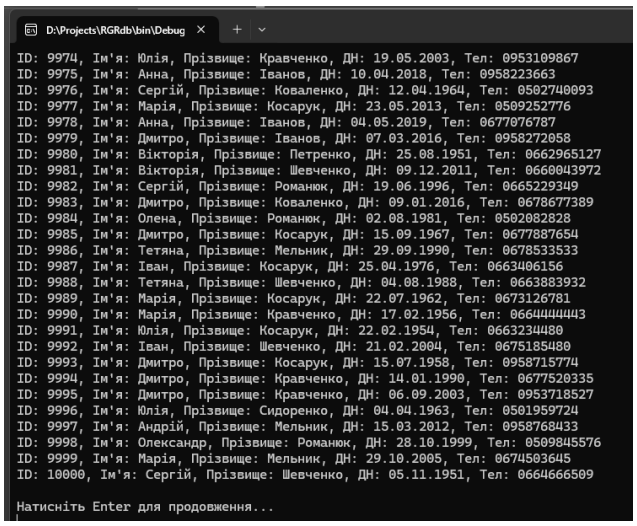
```
public async Task<string> AddAppointmentAsync(Appointment app)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "INSERT INTO appointment (data, hour, diagnosis, patient_id,
doctor_id) " +
                  "VALUES (@data, @hour, @diag, @p_id, @d_id)";
        await using var cmd = new NpgsqlCommand(sql, conn);

        cmd.Parameters.AddWithValue("data", app.Data);
        cmd.Parameters.AddWithValue("hour", app.Hour);
        cmd.Parameters.AddWithValue("diag", app.Diagnosis);
        cmd.Parameters.AddWithValue("p_id", app.PatientId);
        cmd.Parameters.AddWithValue("d_id", app.DoctorId);

        await cmd.ExecuteNonQueryAsync();
```

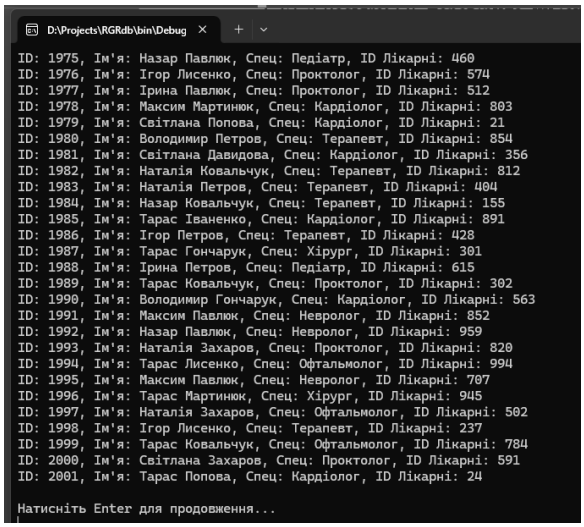
```
            return "Прийом успішно додано.";
        }
        catch (PostgresException ex) when (ex.SqlState == "23503")
        {
            return "ПОМИЛКА: Неможливо додати прийом. " +
                   "Переконайтеся, що пацієнт та лікар з вказаними ID існують.";
        }
    }
```

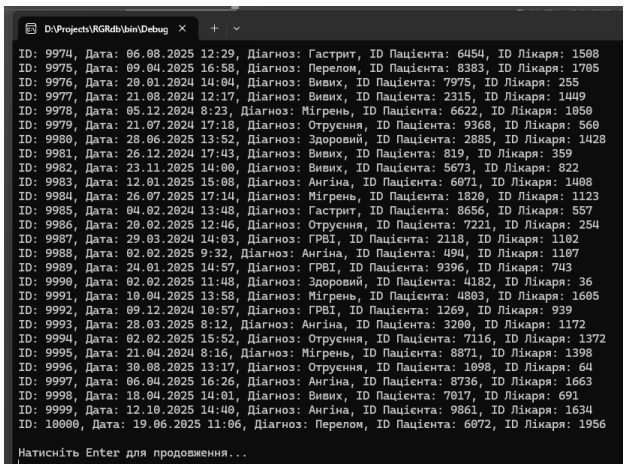
Причина помилки полягає у спрацьовуванні обмеження зовнішнього ключа (foreign key constraint). Користувач намагався додати запис у таблицю appointment зі значенням patient_id=9999.
СУБД PostgreSQL перевірила, чи існує запис з id=99999 у батьківській таблиці patient. Оскільки такого запису не існує, база даних згенерувала помилку порушення цілісності (SQLSTATE 23503), заборонивши операцію INSERT.
Як видно з лістингу, ця помилка була коректно перехоплена в коді C# за допомогою блоку try-catch, і програма вивела користувачу зрозуміле повідомлення замість аварійного завершення.

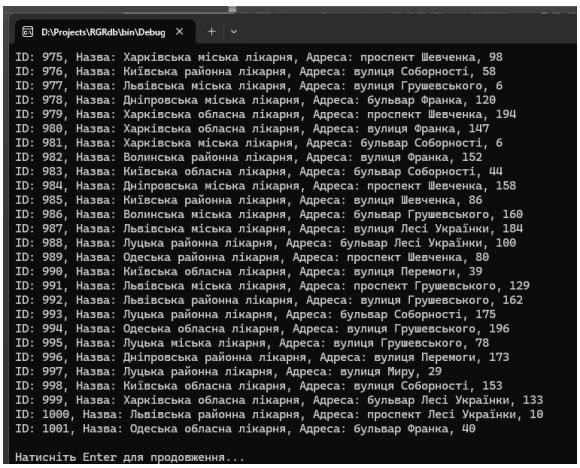
Деталізоване завдання №2











10000 записів пацієнтів, 10000 записів прийомів, 2000 записів лікарів і 1000 записів лікарень згенерувались разом за 12430мс.

```
-- Вхідний параметр (count) = 10000
-- 1. Генеруємо 10000 пацієнтів і повертаємо їх ID
WITH generated_patients AS (
    INSERT INTO patient (name, surname, ""day of birth"", phone)
    SELECT
        (ARRAY['Іван', 'Петро', 'Олександр', 'Сергій', 'Андрій', 'Дмитро', 'Марія',
'Анна', 'Олена', 'Тетяна', 'Вікторія', 'Юлія'])[floor(random() * 12 + 1)::int],
        (ARRAY['Мельник', 'Шевченко', 'Коваленко', 'Бондаренко', 'Ткаченко',
'Кравченко', 'Олійник', 'Петренко', 'Іванов', 'Сидоренко', 'Романюк',
'Косарук'])[floor(random() * 12 + 1)::int],
        DATE '1950-01-01' + (RANDOM() * (365 * 70))::integer,
        (ARRAY['050', '095', '066', '067'])[floor(random() * 4 + 1)::int] ||
LPAD((RANDOM() * 9999999)::int::text, 7, '0')
    FROM generate_series(1, 10000) AS s(id)
    RETURNING id
),

-- 2. Генеруємо лікарні (10000 / 10 + 1 = 1001) і повертаємо їх ID
generated_hospitals AS (
    INSERT INTO hospital (name, address)
    SELECT
        (ARRAY['Київська', 'Львівська', 'Луцька', 'Харківська', 'Одеська',
'Дніпровська', 'Волинська'])[floor(random() * 7 + 1)::int]
        || ' ' ||
        (ARRAY['районна', 'обласна', 'міська'])[floor(random() * 3 + 1)::int]
        || ' лікарня',
        (ARRAY['вулиця', 'проспект', 'бульвар'])[floor(random() * 3 + 1)::int]
        || ' ' ||
        (ARRAY['Шевченка', 'Лесі Українки', 'Франка', 'Грушевського', 'Перемоги',
'Миру', 'Соборності'])[floor(random() * 7 + 1)::int]
        || ', ' ||
        (floor(random() * 200 + 1)::int)::text
    FROM generate_series(1, 10000/ 10 + 1) AS s(id)
    RETURNING id
),

-- 3. Генеруємо лікарів (10000 / 5 + 1 = 2001) і повертаємо їх ID
generated_doctors AS (
    INSERT INTO doctor (name, surname, specialization, hospital_id)
    SELECT
        (ARRAY['Олег', 'Ігор', 'Володимир', 'Максим', 'Назар', 'Тарас', 'Ольга',
'Ірина', 'Наталія', 'Світлана'])[floor(random() * 10 + 1)::int],
        (ARRAY['Петров', 'Іваненко', 'Захаров', 'Павлюк', 'Лисенко', 'Гончарук',
'Попова', 'Давидова', 'Ковальчук', 'Мартинюк'])[floor(random() * 10 + 1)::int],
```

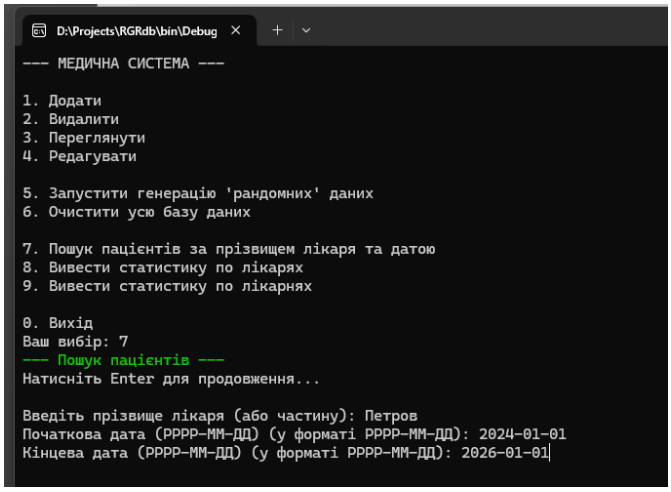
```
        (ARRAY['Терапевт', 'Хірург', 'Проктолог', 'Офтальмолог', 'Кардіолог',
'Невролог', 'Педіатр'])[floor(random() * 7 + 1)::int],
        h_rand.id
    FROM generate_series(1, 10000 / 5 + 1) AS s(id)

    CROSS JOIN LATERAL (
        SELECT id FROM generated_hospitals
        WHERE s.id > 0
        ORDER BY random() LIMIT 1
    ) AS h_rand
    RETURNING id
)
```

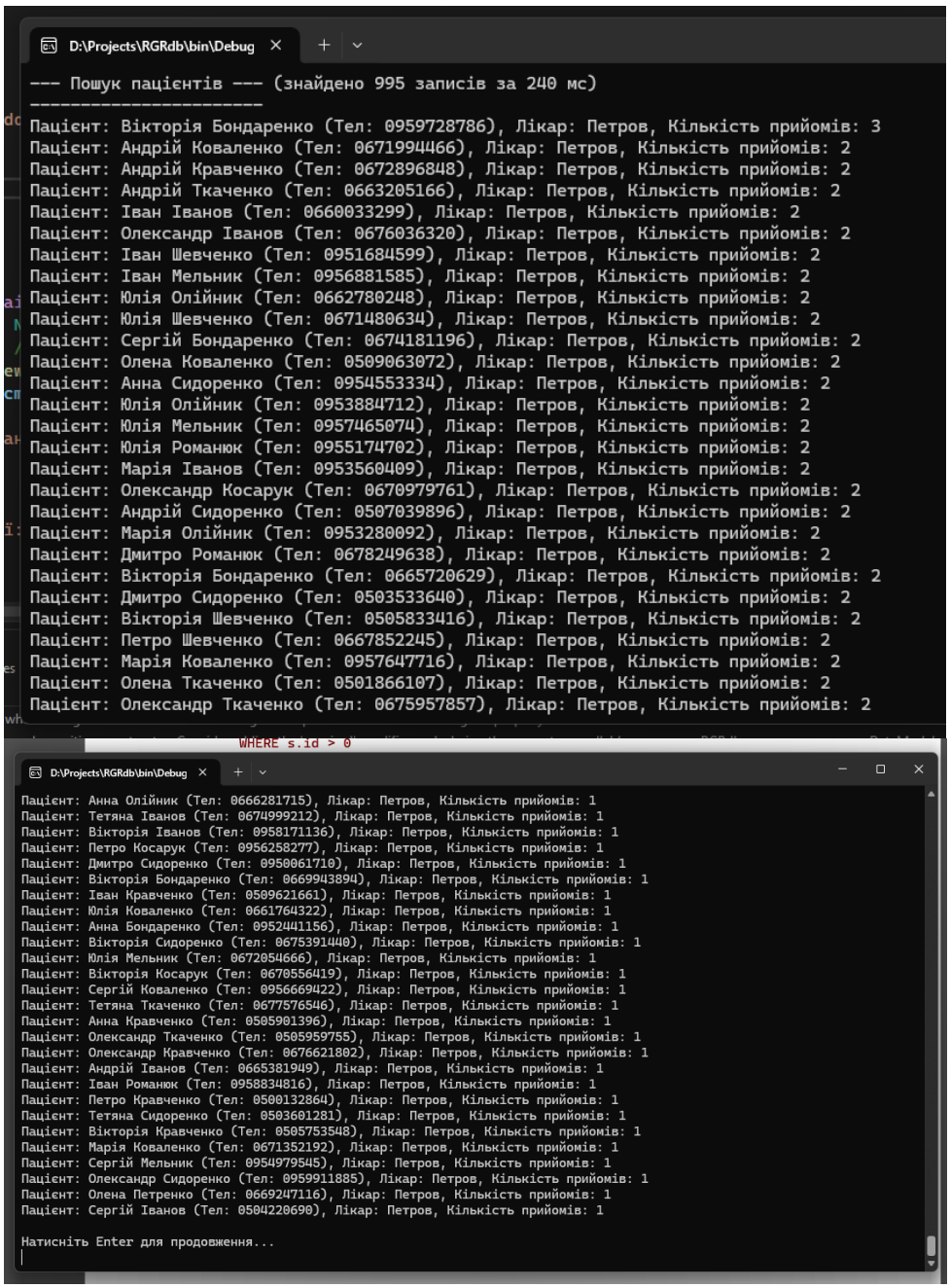
-- 4. Генеруємо 10000 прийомів, використовуючи ID, отримані на попередніх кроках

```
INSERT INTO appointment (data, hour, diagnosis, patient_id, doctor_id)
SELECT
    DATE '2024-01-01' + (RANDOM() * 700)::integer,
    MAKE_TIME(floor(random() * 10 + 8)::int, floor(random() * 60)::int,
floor(random() * 60)::double precision),
    (ARRAY['Здоровий', 'ГРВІ', 'Перелом', 'Вивих', 'Мігрень', 'Ангіна', 'Гастрит',
'Отруєння'])[floor(random() * 8 + 1)::int],
    p_rand.id,
    d_rand.id
FROM generate_series(1, 10000) AS s(id)

CROSS JOIN LATERAL (
    SELECT id FROM generated_patients
    WHERE s.id > 0
    ORDER BY random() LIMIT 1
) AS p_rand

CROSS JOIN LATERAL (
    SELECT id FROM generated_doctors
    WHERE s.id > 0
    ORDER BY random() LIMIT 1
) AS d_rand;
```

Деталізоване завдання №3

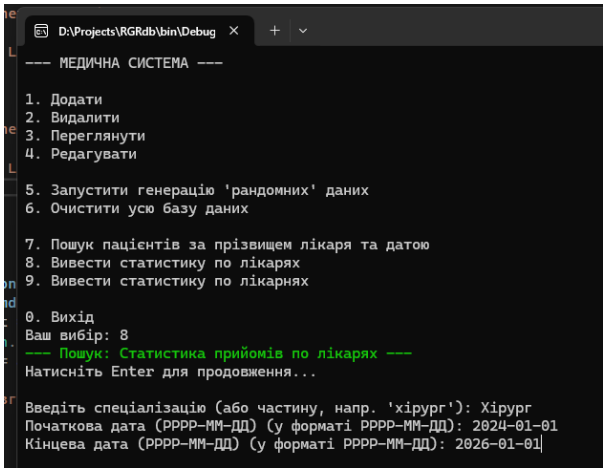
Пошуковий запит №1

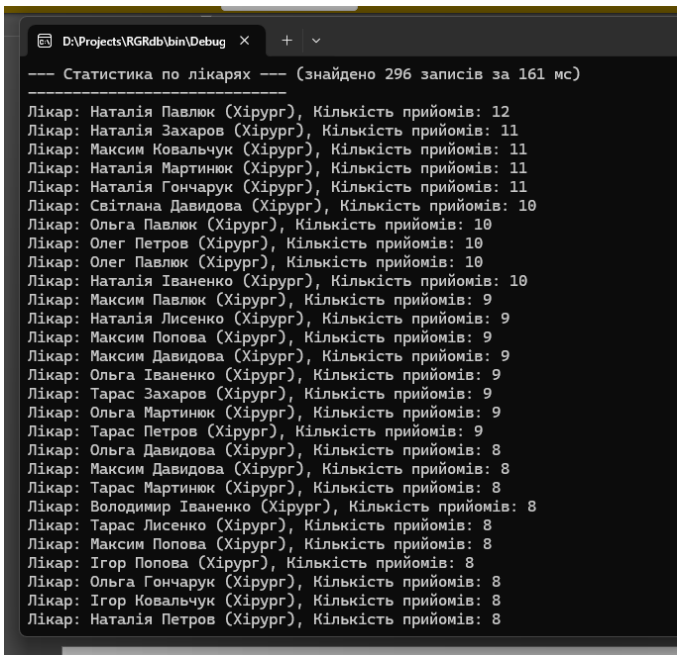




Результат виконання Пошуку №1 (статистика пацієнтів у лікаря)

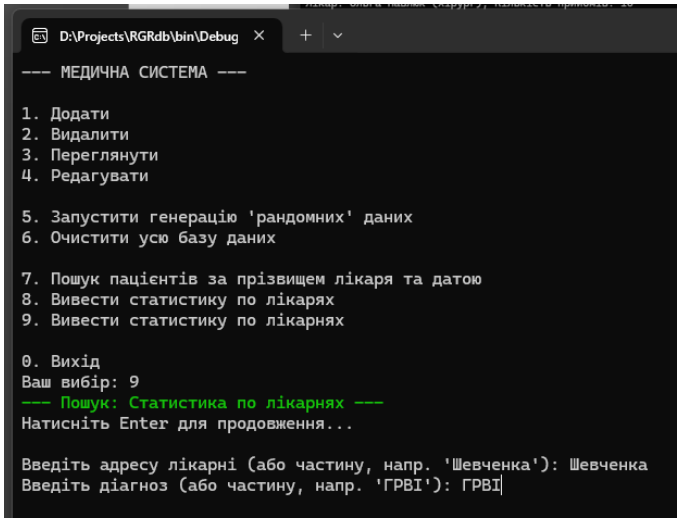
Пошуковий запит №2

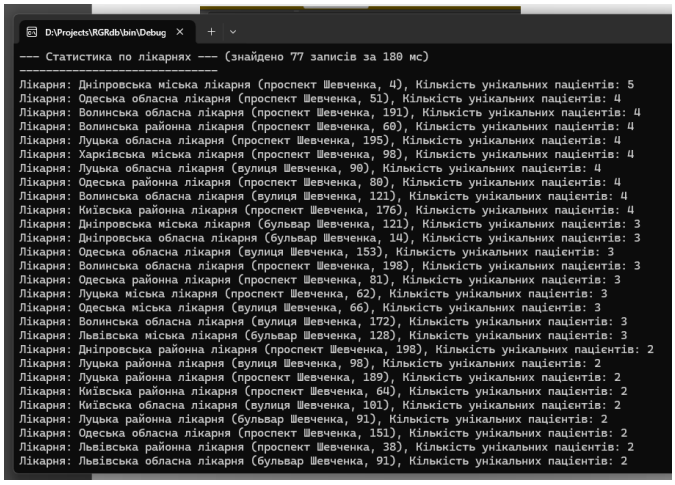




Результат виконання Пошуку №2 (статистика по лікарях)

Пошуковий запит №3





Результат виконання Пошуку №3 (статистика по лікарнях)

Запит підраховує, скільки прийомів мав кожен пацієнт у лікаря з певним прізвищем у заданому діапазоні дат:

```
SELECT
    p.name, p.surname, p.phone,
    d.surname AS doctor_surname,
    COUNT(a.id) AS appointment_count
FROM patient p
JOIN appointment a ON p.id = a.patient_id
JOIN doctor d ON a.doctor_id = d.id
WHERE
    d.surname ILIKE '%Петров%' AND
    a.data BETWEEN '2024-01-01' AND '2026-01-01'
GROUP BY
    p.id, p.name, p.surname, p.phone, d.surname
ORDER BY
    appointment_count DESC;
```

Запит підраховує, скільки прийомів провів кожен лікар певної спеціалізації за вказаний період:

```
SELECT
    d.name,
    d.surname,
    d.specialization,
    COUNT(a.id) AS appointment_count
FROM doctor d
LEFT JOIN appointment a ON d.id = a.doctor_id
WHERE
    d.specialization ILIKE '%Хірург%' AND
    a.data BETWEEN '2024-01-01' AND '2026-01-01'
GROUP BY
    d.id, d.name, d.surname, d.specialization
ORDER BY
    appointment_count DESC;
```

Запит підраховує, скільки унікальних пацієнтів із певним діагнозом прийняла кожна лікарня, що знаходиться за певною адресою:

```
SELECT
    h.name,
    h.address,
    COUNT(DISTINCT a.patient_id) AS patient_count
FROM hospital h
JOIN doctor d ON h.id = d.hospital_id
JOIN appointment a ON d.id = a.doctor_id
WHERE
    h.address ILIKE '%Шевченка%' AND
    a.diagnosis ILIKE '%ГРВІ%'
GROUP BY
    h.id, h.name, h.address
ORDER BY
    patient_count DESC;
```

Деталізоване завдання №4

Модуль **Model** (реалізований у класі DatabaseModel) є ключовим компонентом архітектурного шаблону MVC. Його головна відповідальність — це повна **інкапсуляція всієї логіки взаємодії з базою даних PostgreSQL**.

Це єдиний модуль у програмі, який "знає" про існування СУБД, мови SQL та бібліотеки Npgsql. Він повністю приховує від Controller та View деталі реалізації запитів.

Керування підключенням

- **GetConnectionAsync**: (Приватний метод) Відкриває та повертає нове асинхронне з'єднання з базою даних PostgreSQL.

```
25 references
private async Task<NpgsqlConnection> GetConnectionAsync()
{
    var conn = new NpgsqlConnection(_connectionString);
    await conn.OpenAsync();
    return conn;
}
```

Create (Створення)

- **AddPatientAsync**: Додає нового пацієнта в таблицю patient.

```
public async Task AddPatientAsync(Patient patient)
{
    await using var conn = await GetConnectionAsync();
    var sql = "INSERT INTO patient (name, surname, \"day of birth\", phone) " +
      "VALUES (@name, @surname, @dob, @phone)";
    await using var cmd = new NpgsqlCommand(sql, conn);

    cmd.Parameters.AddWithValue("name", patient.Name);
    cmd.Parameters.AddWithValue("surname", patient.Surname);
    cmd.Parameters.AddWithValue("dob", patient.DayOfBirth);
    cmd.Parameters.AddWithValue("phone", patient.Phone);

    await cmd.ExecuteNonQueryAsync();
}
```

- **AddDoctorAsync**: Додає нового лікаря в таблицю doctor, перехоплюючи помилку, якщо вказаний hospital_id не існує.

```
public async Task<string> AddDoctorAsync(Doctor doctor)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "INSERT INTO doctor (name, surname, specialization, hospital_id) " +
                  "VALUES (@name, @surname, @spec, @h_id)";
        await using var cmd = new NpgsqlCommand(sql, conn);

        cmd.Parameters.AddWithValue("name", doctor.Name);
        cmd.Parameters.AddWithValue("surname", doctor.Surname);
        cmd.Parameters.AddWithValue("spec", doctor.Specialization);
        cmd.Parameters.AddWithValue("h_id", doctor.HospitalId);

        await cmd.ExecuteNonQueryAsync();
        return "Лікаря успішно додано.";
    }
    catch (PostgresException ex) when (ex.SqlState == "23503")
    {
        return "ПОМИЛКА: Неможливо додати лікаря. " +
               "Переконайтеся, що лікарня з ID = " + doctor.HospitalId + " існує.";
    }
    catch (Exception ex)
    {
        return $"Загальна помилка: {ex.Message}";
    }
}
```

- **AddAppointmentAsync**: Додає новий прийом у таблицю appointment, перехоплюючи помилку, якщо patient_id або doctor_id не існують.

```
public async Task<string> AddAppointmentAsync(Appointment app)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "INSERT INTO appointment (data, hour, diagnosis, patient_id, doctor_id) " +
                  "VALUES (@data, @hour, @diag, @p_id, @d_id)";
        await using var cmd = new NpgsqlCommand(sql, conn);

        cmd.Parameters.AddWithValue("data", app.Data);
        cmd.Parameters.AddWithValue("hour", app.Hour);
        cmd.Parameters.AddWithValue("diag", app.Diagnosis);
        cmd.Parameters.AddWithValue("p_id", app.PatientId);
        cmd.Parameters.AddWithValue("d_id", app.DoctorId);

        await cmd.ExecuteNonQueryAsync();
        return "Прийом успішно додано.";
    }
    catch (PostgresException ex) when (ex.SqlState == "23503")
    {
        return "ПОМИЛКА: Неможливо додати прийом. " +
               "Переконайтеся, що пацієнт та лікар з вказаними ID існують.";
    }
}
```

- **AddHospitalAsync**: Додає нову лікарню в таблицю hospital.

```
public async Task<string> AddHospitalAsync(Hospital hospital)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "INSERT INTO hospital (name, address) VALUES (@name, @address)";
        await using var cmd = new NpgsqlCommand(sql, conn);

        cmd.Parameters.AddWithValue("name", hospital.Name);
        cmd.Parameters.AddWithValue("address", hospital.Address);

        await cmd.ExecuteNonQueryAsync();
        return "Лікарню успішно додано.";
    }
    catch (Exception ex)
    {
        return $"ПОМИЛКА: {ex.Message}";
    }
}
```

Read (Читання)

- **GetAllPatientsAsync**: Виконує SELECT *, отримує та повертає повний список усіх пацієнтів (List<Patient>).

```
public async Task<List<Patient>> GetAllPatientsAsync()
{
    var patients = new List<Patient>();
    await using var conn = await GetConnectionAsync();
    var sql = "SELECT id, name, surname, \"day of birth\", phone FROM patient ORDER BY id";
    await using var cmd = new NpgsqlCommand(sql, conn);

    await using var reader = await cmd.ExecuteReaderAsync();
    while (await reader.ReadAsync())
    {
        patients.Add(new Patient
        {
            Id = reader.GetInt32(reader.GetOrdinal("id")),
            Name = reader.IsDBNull(reader.GetOrdinal("name")) ? "[Немає даних]" : reader.GetString(reader.GetOrdinal("name")),
            Surname = reader.IsDBNull(reader.GetOrdinal("surname")) ? "[Немає даних]" : reader.GetString(reader.GetOrdinal("surname")),
            DayOfBirth = reader.IsDBNull(reader.GetOrdinal("day of birth")) ? DateOnly.MinValue : DateOnly.FromDateTime(reader.GetDateTime(reader.GetOrdinal("day of birth"))),
            Phone = reader.IsDBNull(reader.GetOrdinal("phone")) ? "[Немає даних]" : reader.GetString(reader.GetOrdinal("phone"))
        });
    }
    return patients;
}
```

- **GetAllDoctorsAsync**: Повертає повний список усіх лікарів (List<Doctor>).

```
public async Task<List<Doctor>> GetAllDoctorsAsync()
{
    var doctors = new List<Doctor>();
    await using var conn = await GetConnectionAsync();
    var sql = "SELECT id, name, surname, specialization, hospital_id FROM doctor ORDER BY id";
    await using var cmd = new NpgsqlCommand(sql, conn);

    await using var reader = await cmd.ExecuteReaderAsync();
    while (await reader.ReadAsync())
    {
        doctors.Add(new Doctor
        {
            Id = reader.GetInt32(reader.GetOrdinal("id")),
            Name = reader.IsDBNull(reader.GetOrdinal("name")) ? "[Немає даних]" : reader.GetString(reader.GetOrdinal("name")),
            Surname = reader.IsDBNull(reader.GetOrdinal("surname")) ? "[Немає даних]" : reader.GetString(reader.GetOrdinal("surname")),
            Specialization = reader.IsDBNull(reader.GetOrdinal("specialization")) ? "[Немає даних]" : reader.GetString(reader.GetOrdinal("specialization")),
            HospitalId = reader.GetInt32(reader.GetOrdinal("hospital_id"))
        });
    }
    return doctors;
}
```

- **GetAllHospitalsAsync**: Повертає повний список усіх лікарень (List<Hospital>).

```
public async Task<List<Hospital>> GetAllHospitalsAsync()
{
    var hospitals = new List<Hospital>();
    await using var conn = await GetConnectionAsync();
    var sql = "SELECT id, name, address FROM hospital ORDER BY id";
    await using var cmd = new NpgsqlCommand(sql, conn);

    await using var reader = await cmd.ExecuteReaderAsync();
    while (await reader.ReadAsync())
    {
        hospitals.Add(new Hospital
        {
            Id = reader.GetInt32(reader.GetOrdinal("id")),
            Name = reader.IsDBNull(reader.GetOrdinal("name")) ? "[Немає даних]" : reader.GetString(reader.GetOrdinal("name")),
            Address = reader.IsDBNull(reader.GetOrdinal("address")) ? "[Немає даних]" : reader.GetString(reader.GetOrdinal("address"))
        });
    }
    return hospitals;
}
```

- **GetAllAppointmentsAsync**: Повертає повний список усіх прийомів (List<Appointment>).

```
public async Task<List<Appointment>> GetAllAppointmentsAsync()
{
    var appointments = new List<Appointment>();
    await using var conn = await GetConnectionAsync();
    var sql = "SELECT id, data, hour, diagnosis, patient_id, doctor_id FROM appointment ORDER BY id";
    await using var cmd = new NpgsqlCommand(sql, conn);

    await using var reader = await cmd.ExecuteReaderAsync();
    while (await reader.ReadAsync())
    {
        appointments.Add(new Appointment
        {
            Id = reader.GetInt32(reader.GetOrdinal("id")),
            Data = reader.IsDBNull(reader.GetOrdinal("data")) ? DateOnly.MinValue : DateOnly.FromDateTime(reader.GetDateTime(reader.GetOrdinal("data"))),
            Hour = reader.IsDBNull(reader.GetOrdinal("hour")) ? TimeOnly.MinValue : TimeOnly.FromTimeSpan(reader.GetTimeSpan(reader.GetOrdinal("hour"))),
            Diagnosis = reader.IsDBNull(reader.GetOrdinal("diagnosis")) ? "[Немає даних]" : reader.GetString(reader.GetOrdinal("diagnosis")),
            PatientId = reader.GetInt32(reader.GetOrdinal("patient_id")),
            DoctorId = reader.GetInt32(reader.GetOrdinal("doctor_id"))
        });
    }
    return appointments;
}
```

- **GetPatientByIdAsync**: Знаходить та повертає один об'єкт Patient за його id.

```
public async Task<Patient> GetPatientByIdAsync(int id)
{
    await using var conn = await GetConnectionAsync();
    var sql = "SELECT id, name, surname, \"day of birth\", phone FROM patient WHERE id = @id";
    await using var cmd = new NpgsqlCommand(sql, conn);
    cmd.Parameters.AddWithValue("id", id);

    await using var reader = await cmd.ExecuteReaderAsync();
    if (await reader.ReadAsync())
    {
        return new Patient
        {
            Id = reader.GetInt32(reader.GetOrdinal("id")),
            Name = reader.IsDBNull(reader.GetOrdinal("name")) ? "" : reader.GetString(reader.GetOrdinal("name")),
            Surname = reader.IsDBNull(reader.GetOrdinal("surname")) ? "" : reader.GetString(reader.GetOrdinal("surname")),
            DayOfBirth = reader.IsDBNull(reader.GetOrdinal("day of birth")) ? DateOnly.MinValue : DateOnly.FromDateTime(reader.GetDateTime(reader.GetOrdinal("day of birth"))),
            Phone = reader.IsDBNull(reader.GetOrdinal("phone")) ? "" : reader.GetString(reader.GetOrdinal("phone"))
        };
    }
    return null;
}
```

- **GetDoctorByIdAsync**: Знаходить та повертає один об'єкт Doctor за його id.

```
public async Task<Doctor> GetDoctorByIdAsync(int id)
{
    await using var conn = await GetConnectionAsync();
    var sql = "SELECT id, name, surname, specialization, hospital_id FROM doctor WHERE id = @id";
    await using var cmd = new NpgsqlCommand(sql, conn);
    cmd.Parameters.AddWithValue("id", id);

    await using var reader = await cmd.ExecuteReaderAsync();
    if (await reader.ReadAsync())
    {
        return new Doctor
        {
            Id = reader.GetInt32(reader.GetOrdinal("id")),
            Name = reader.IsDBNull(reader.GetOrdinal("name")) ? "" : reader.GetString(reader.GetOrdinal("name")),
            Surname = reader.IsDBNull(reader.GetOrdinal("surname")) ? "" : reader.GetString(reader.GetOrdinal("surname")),
            Specialization = reader.IsDBNull(reader.GetOrdinal("specialization")) ? "" : reader.GetString(reader.GetOrdinal("specialization")),
            HospitalId = reader.GetInt32(reader.GetOrdinal("hospital_id"))
        };
    }
    return null;
}
```

- **GetHospitalByIdAsync**: Знаходить та повертає один об'єкт Hospital за його id.

```
public async Task<Hospital> GetHospitalByIdAsync(int id)
{
    await using var conn = await GetConnectionAsync();
    var sql = "SELECT id, name, address FROM hospital WHERE id = @id";
    await using var cmd = new NpgsqlCommand(sql, conn);
    cmd.Parameters.AddWithValue("id", id);

    await using var reader = await cmd.ExecuteReaderAsync();
    if (await reader.ReadAsync())
    {
        return new Hospital
        {
            Id = reader.GetInt32(reader.GetOrdinal("id")),
            Name = reader.IsDBNull(reader.GetOrdinal("name")) ? "" : reader.GetString(reader.GetOrdinal("name")),
            Address = reader.IsDBNull(reader.GetOrdinal("address")) ? "" : reader.GetString(reader.GetOrdinal("address"))
        };
    }
    return null;
}
```

- **GetAppointmentByIdAsync**: Знаходить та повертає один об'єкт Appointment за його id.

```
public async Task<Appointment> GetAppointmentByIdAsync(int id)
{
    await using var conn = await GetConnectionAsync();
    var sql = "SELECT id, data, hour, diagnosis, patient_id, doctor_id FROM appointment WHERE id = @id";
    await using var cmd = new NpgsqlCommand(sql, conn);
    cmd.Parameters.AddWithValue("id", id);

    await using var reader = await cmd.ExecuteReaderAsync();
    if (await reader.ReadAsync())
    {
        return new Appointment
        {
            Id = reader.GetInt32(reader.GetOrdinal("id")),
            Data = reader.IsDBNull(reader.GetOrdinal("data")) ? DateOnly.MinValue : DateOnly.FromDateTime(reader.GetDateTime(reader.GetOrdinal("data"))),
            Hour = reader.IsDBNull(reader.GetOrdinal("hour")) ? TimeOnly.MinValue : TimeOnly.FromTimeSpan(reader.GetTimeSpan(reader.GetOrdinal("hour"))),
            Diagnosis = reader.IsDBNull(reader.GetOrdinal("diagnosis")) ? "" : reader.GetString(reader.GetOrdinal("diagnosis")),
            PatientId = reader.GetInt32(reader.GetOrdinal("patient_id")),
            DoctorId = reader.GetInt32(reader.GetOrdinal("doctor_id"))
        };
    }
    return null;
}
```

Update (Оновлення)

- **UpdatePatientAsync**: Оновлює дані існуючого пацієнта в базі даних за його id.

```
public async Task<string> UpdatePatientAsync(Patient patient)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "UPDATE patient SET name = @name, surname = @surname, \"day of birth\" = @dob, phone = @phone " +
                  "WHERE id = @id";
        await using var cmd = new NpgsqlCommand(sql, conn);

        cmd.Parameters.AddWithValue("name", patient.Name);
        cmd.Parameters.AddWithValue("surname", patient.Surname);
        cmd.Parameters.AddWithValue("dob", patient.DayOfBirth);
        cmd.Parameters.AddWithValue("phone", patient.Phone);
        cmd.Parameters.AddWithValue("id", patient.Id);

        int rowsAffected = await cmd.ExecuteNonQueryAsync();
        return rowsAffected > 0 ? "Дані пацієнта оновлено." : "ПОМИЛКА: Пацієнта не знайдено.";
    }
    catch (Exception ex)
    {
        return $"ПОМИЛКА: {ex.Message}";
    }
}
```

- **UpdateDoctorAsync**: Оновлює дані існуючого лікаря, перехоплюючи помилку, якщо новий hospital_id є недійсним.

```
public async Task<string> UpdateDoctorAsync(Doctor doctor)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "UPDATE doctor SET name = @name, surname = @surname, specialization = @spec, hospital_id = @h_id " +
                  "WHERE id = @id";
        await using var cmd = new NpgsqlCommand(sql, conn);

        cmd.Parameters.AddWithValue("name", doctor.Name);
        cmd.Parameters.AddWithValue("surname", doctor.Surname);
        cmd.Parameters.AddWithValue("spec", doctor.Specialization);
        cmd.Parameters.AddWithValue("h_id", doctor.HospitalId);
        cmd.Parameters.AddWithValue("id", doctor.Id);

        int rowsAffected = await cmd.ExecuteNonQueryAsync();
        return rowsAffected > 0 ? "Дані лікаря оновлено." : "ПОМИЛКА: Лікаря не знайдено.";
    }
    catch (PostgresException ex) when (ex.SqlState == "23503")
    {
        return "ПОМИЛКА: Неможливо оновити. Лікарня з ID = " + doctor.HospitalId + " не існує.";
    }
    catch (Exception ex)
    {
        return $"ПОМИЛКА: {ex.Message}";
    }
}
```

- **UpdateHospitalAsync**: Оновлює назву та адресу існуючої лікарні.

```
public async Task<string> UpdateHospitalAsync(Hospital hospital)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "UPDATE hospital SET name = @name, address = @address WHERE id = @id";
        await using var cmd = new NpgsqlCommand(sql, conn);

        cmd.Parameters.AddWithValue("name", hospital.Name);
        cmd.Parameters.AddWithValue("address", hospital.Address);
        cmd.Parameters.AddWithValue("id", hospital.Id);

        int rowsAffected = await cmd.ExecuteNonQueryAsync();
        return rowsAffected > 0 ? "Дані лікарні оновлено." : "ПОМИЛКА: Лікарню не знайдено.";
    }
    catch (Exception ex)
    {
        return $"ПОМИЛКА: {ex.Message}";
    }
}
```

- **UpdateAppointmentAsync**: Оновлює всі дані існуючого прийому, перехоплюючи помилки недійсних patient_id або doctor_id.

```
public async Task<string> UpdateAppointmentAsync(Appointment app)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "UPDATE appointment SET data = @data, hour = @hour, diagnosis = @diag, " +
                  "patient_id = @p_id, doctor_id = @d_id WHERE id = @id";
        await using var cmd = new NpgsqlCommand(sql, conn);

        cmd.Parameters.AddWithValue("data", app.Data);
        cmd.Parameters.AddWithValue("hour", app.Hour);
        cmd.Parameters.AddWithValue("diag", app.Diagnosis);
        cmd.Parameters.AddWithValue("p_id", app.PatientId);
        cmd.Parameters.AddWithValue("d_id", app.DoctorId);
        cmd.Parameters.AddWithValue("id", app.Id);

        int rowsAffected = await cmd.ExecuteNonQueryAsync();
        return rowsAffected > 0 ? "Дані прийому оновлено." : "ПОМИЛКА: Прийом не знайдено.";
    }
    catch (PostgresException ex) when (ex.SqlState == "23503")
    {
        return "ПОМИЛКА: Неможливо оновити. Переконайтеся, що пацієнт та лікар існують.";
    }
    catch (Exception ex)
    {
        return $"ПОМИЛКА: {ex.Message}";
    }
}
```

Delete (Видалення)

- **DeletePatientAsync**: Видаляє пацієнта за id. Перехоплює помилку (23503), якщо у пацієнта є пов'язані прийоми.

```
public async Task<string> DeletePatientAsync(int patientId)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "DELETE FROM patient WHERE id = @id";
        await using var cmd = new NpgsqlCommand(sql, conn);
        cmd.Parameters.AddWithValue("id", patientId);

        int rowsAffected = await cmd.ExecuteNonQueryAsync();

        if (rowsAffected > 0)
        {
            return "Пацієнта успішно видалено.";
        }
        else
        {
            return "ПОМИЛКА: Пацієнта з ID = " + patientId + " не знайдено.";
        }
    }
    catch (PostgresException ex) when (ex.SqlState == "23503")
    {
        return "ПОМИЛКА: Неможливо видалити пацієнта, оскільки за ним закріплені 'прийоми'.";
    }
    catch (Exception ex)
    {
        return $"Загальна помилка: {ex.Message}";
    }
}
```

- **DeleteDoctorAsync**: Видаляє лікаря за id. Перехоплює помилку, якщо у лікаря є пов'язані прийоми.

```
public async Task<string> DeleteDoctorAsync(int doctorId)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "DELETE FROM doctor WHERE id = @id";
        await using var cmd = new NpgsqlCommand(sql, conn);
        cmd.Parameters.AddWithValue("id", doctorId);

        int rowsAffected = await cmd.ExecuteNonQueryAsync();

        if (rowsAffected > 0)
        {
            return "Лікаря успішно видалено.";
        }
        else
        {
            return "ПОМИЛКА: Лікаря з ID = " + doctorId + " не знайдено.";
        }
    }
    catch (PostgresException ex) when (ex.SqlState == "23503")
    {
        return "ПОМИЛКА: Неможливо видалити лікаря, оскільки за ним закріплені 'прийоми'.";
    }
    catch (Exception ex)
    {
        return $"Загальна помилка: {ex.Message}";
    }
}
```

- **DeleteAppointmentAsync**: Видаляє прийом за id.

```
public async Task<string> DeleteAppointmentAsync(int appointmentId)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "DELETE FROM appointment WHERE id = @id";
        await using var cmd = new NpgsqlCommand(sql, conn);
        cmd.Parameters.AddWithValue("id", appointmentId);

        int rowsAffected = await cmd.ExecuteNonQueryAsync();

        if (rowsAffected > 0)
        {
            return "Прийом успішно видалено.";
        }
        else
        {
            return "ПОМИЛКА: Прийом з ID = " + appointmentId + " не знайдено.";
        }
    }
    catch (Exception ex)
    {
        return $"Загальна помилка: {ex.Message}";
    }
}
```

- **DeleteHospitalAsync**: Видаляє лікарню за id. Перехоплює помилку, якщо за лікарнею закріплені лікарі.

```
public async Task<string> DeleteHospitalAsync(int hospitalId)
{
    try
    {
        await using var conn = await GetConnectionAsync();
        var sql = "DELETE FROM hospital WHERE id = @id";
        await using var cmd = new NpgsqlCommand(sql, conn);
        cmd.Parameters.AddWithValue("id", hospitalId);
        await cmd.ExecuteNonQueryAsync();
        return "Лікарню успішно видалено.";
    }
    catch (PostgresException ex) when (ex.SqlState == "23503")
    {
        return "ПОМИЛКА: Неможливо видалити лікарню, оскільки за нею закріплені лікарі.";
    }
    catch (Exception ex)
    {
        return $"Загальна помилка: {ex.Message}";
    }
}
```

Адміністрування та Пошук

- **GenerateRandomDataAsync**: Виконує єдиний, складний SQL-запит (з WITH та CROSS JOIN LATERAL) для заповнення всіх таблиць вказаною кількістю (count) реалістичних, випадково згенерованих даних.

```
public async Task<string> GenerateRandomDataAsync(int count)
{
    var sql = $@"
    WITH generated_patients AS (
        INSERT INTO patient (name, surname, ""day of birth"", phone)
        SELECT
            (ARRAY['Іван', 'Петро', 'Олександр', 'Сергій', 'Андрій', 'Дмитро', 'Марія', 'Анна', 'Олена', 'Тетяна', 'Вікторія', 'Юлія'])[floor(random() * 12 + 1)::int],
            (ARRAY['Мельник', 'Шевченко', 'Коваленко', 'Бондаренко', 'Ткаченко', 'Кравченко', 'Олійник', 'Петренко', 'Іванов', 'Сидоренко', 'Романюк', 'Косарук'])[floor(random() * 12 + 1)::int],
            DATE '1950-01-01' + (RANDOM() * (365 * 70))::integer,
            (ARRAY['050', '095', '066', '067'])[floor(random() * 4 + 1)::int] || LPAD((RANDOM() * 9999999)::int::text, 7, '0')
        FROM generate_series(1, {count}) AS s(id)
        RETURNING id
    ),
    generated_hospitals AS (
        INSERT INTO hospital (name, address)
        SELECT
            (ARRAY['Київська', 'Львівська', 'Луцька', 'Харківська', 'Одеська', 'Дніпровська', 'Волинська'])[floor(random() * 7 + 1)::int]
            || ' ' ||
            (ARRAY['районна', 'обласна', 'міська'])[floor(random() * 3 + 1)::int]
            || ' лікарня',
            (ARRAY['вулиця', 'проспект', 'бульвар'])[floor(random() * 3 + 1)::int]
            || ' ' ||
            (ARRAY['Шевченка', 'Лесі Українки', 'Франка', 'Грушевського', 'Перемоги', 'Миру', 'Соборності'])[floor(random() * 7 + 1)::int]
            || ', ' ||
            (floor(random() * 200 + 1)::int)::text
        FROM generate_series(1, {count / 10 + 1}) AS s(id)
        RETURNING id
    ),
    generated_doctors AS (
        INSERT INTO doctor (name, surname, specialization, hospital_id)
        SELECT
            (ARRAY['Олег', 'Ігор', 'Володимир', 'Максим', 'Назар', 'Тарас', 'Ольга', 'Ірина', 'Наталія', 'Світлана'])[floor(random() * 10 + 1)::int],
            (ARRAY['Петров', 'Іваненко', 'Захаров', 'Павлюк', 'Лисенко', 'Гончарук', 'Попова', 'Давидова', 'Ковальчук', 'Мартинюк'])[floor(random() * 10 + 1)::int],
            (ARRAY['Терапевт', 'Хірург', 'Проктолог', 'Офтальмолог', 'Кардіолог', 'Невролог', 'Педіатр'])[floor(random() * 7 + 1)::int],
            h_rand.id
        FROM generate_series(1, {count / 5 + 1}) AS s(id)

        CROSS JOIN LATERAL (
            SELECT id FROM generated_hospitals
            WHERE s.id > 0
            ORDER BY random() LIMIT 1
        ) AS h_rand
```

```
        RETURNING id
    )
    INSERT INTO appointment (data, hour, diagnosis, patient_id, doctor_id)
    SELECT
        DATE '2024-01-01' + (RANDOM() * 700)::integer,
        MAKE_TIME(floor(random() * 10 + 8)::int, floor(random() * 60)::int, floor(random() * 60)::double precision),
        (ARRAY['Здоровий', 'ГРВІ', 'Перелом', 'Вивих', 'Мігрень', 'Ангіна', 'Гастрит', 'Отруєння'])[floor(random() * 8 + 1)::int],
        p_rand.id,
        d_rand.id
    FROM generate_series(1, {count}) AS s(id)

    CROSS JOIN LATERAL (
        SELECT id FROM generated_patients
        WHERE s.id > 0
        ORDER BY random() LIMIT 1
    ) AS p_rand

    CROSS JOIN LATERAL (
        SELECT id FROM generated_doctors
        WHERE s.id > 0
        ORDER BY random() LIMIT 1
    ) AS d_rand;
";

    try
    {
        await using var conn = await GetConnectionAsync();
        await using var cmd = new NpgsqlCommand(sql, conn);
        cmd.CommandTimeout = 300; // 5
        var sw = Stopwatch.StartNew();
        int rowsAffected = await cmd.ExecuteNonQueryAsync();
        sw.Stop();
        return $"Успішно згенеровано дані. Загальний час: {sw.ElapsedMilliseconds} мс.";
    }
    catch (Exception ex)
    {
        return $"Помилка генерації: {ex.Message}";
    }
}
```

- **ClearAllDataAsync**: Повністю очищує всі таблиці (TRUNCATE ... CASCADE) та скидає лічильники id до 1.

```
public async Task<string> ClearAllDataAsync()
{
    try
    {
        await using var conn = await GetConnectionAsync();

        var sql = @"
        TRUNCATE hospital RESTART IDENTITY CASCADE;
        TRUNCATE patient RESTART IDENTITY CASCADE;
    ";

        await using var cmd = new NpgsqlCommand(sql, conn);
        await cmd.ExecuteNonQueryAsync();

        return "Успіх! Усі дані з таблиць видалено, лічильники ID скинуто до 1.";
    }
    catch (Exception ex)
    {
        return $"ПОМИЛКА: Не вдалося очистити базу даних. {ex.Message}";
    }
}
```

- **SearchPatientsByDoctorAndDateAsync**: Виконує аналітичний запит (Пошук №1), який рахує (GROUP BY) кількість візитів кожного пацієнта до лікаря (за прізвищем) у межах діапазону дат. Повертає результат та час виконання.

```
public async Task<(List<string> results, long timeMs)> SearchPatientsByDoctorAndDateAsync(
    string doctorSurnamePattern, DateOnly startDate, DateOnly endDate)
{
    var results = new List<string>();
    var sw = Stopwatch.StartNew();

    var sql = $@"
    SELECT
        p.name, p.surname, p.phone,
        d.surname AS doctor_surname,
        COUNT(a.id) AS appointment_count
    FROM patient p
    JOIN appointment a ON p.id = a.patient_id
    JOIN doctor d ON a.doctor_id = d.id
    WHERE
        d.surname ILIKE @surnamePattern AND
        a.data BETWEEN @startDate AND @endDate
    GROUP BY
        p.id, p.name, p.surname, p.phone, d.surname
    ORDER BY
        appointment_count DESC;
";

    await using var conn = await GetConnectionAsync();
    await using var cmd = new NpgsqlCommand(sql, conn);

    cmd.Parameters.AddWithValue("surnamePattern", $"%{doctorSurnamePattern}%");
    cmd.Parameters.AddWithValue("startDate", startDate);
    cmd.Parameters.AddWithValue("endDate", endDate);

    await using var reader = await cmd.ExecuteReaderAsync();
    while (await reader.ReadAsync())
    {
        results.Add(
            $"Пацієнт: {reader["name"]} {reader["surname"]} (Тел: {reader["phone"]}), " +
            $"Лікар: {reader["doctor_surname"]}, " +
            $"Кількість прийомів: {reader["appointment_count"]}"
        );
    }

    sw.Stop();
    return (results, sw.ElapsedMilliseconds);
}
```

- **SearchDoctorStatisticsAsync**: Виконує аналітичний запит (Пошук №2), який рахує (GROUP BY) кількість прийомів для лікарів певної спеціалізації у межах діапазону дат. Повертає результат та час виконання.

```
public async Task<(List<string> results, long timeMs)> SearchDoctorStatisticsAsync(
    string specializationPattern, DateOnly startDate, DateOnly endDate)
{
    var results = new List<string>();
    var sw = Stopwatch.StartNew();

    var sql = $@"
    SELECT
        d.name,
        d.surname,
        d.specialization,
        COUNT(a.id) AS appointment_count
    FROM doctor d
    LEFT JOIN appointment a ON d.id = a.doctor_id
    WHERE
        d.specialization ILIKE @specPattern AND
        a.data BETWEEN @startDate AND @endDate
    GROUP BY
        d.id, d.name, d.surname, d.specialization
    ORDER BY
        appointment_count DESC;
";

    await using var conn = await GetConnectionAsync();
    await using var cmd = new NpgsqlCommand(sql, conn);

    cmd.Parameters.AddWithValue("specPattern", $"%{specializationPattern}%");
    cmd.Parameters.AddWithValue("startDate", startDate);
    cmd.Parameters.AddWithValue("endDate", endDate);

    await using var reader = await cmd.ExecuteReaderAsync();
    while (await reader.ReadAsync())
    {
        results.Add(
            $"Лікар: {reader["name"]} {reader["surname"]} ({reader["specialization"]}), " +
            $"Кількість прийомів: {reader["appointment_count"]}"
        );
    }

    sw.Stop();
    return (results, sw.ElapsedMilliseconds);
}
```

- **SearchHospitalStatisticsAsync**: Виконує аналітичний запит (Пошук №3), який рахує (COUNT(DISTINCT)) кількість унікальних пацієнтів з певним діагнозом для лікарень за певною адресою. Повертає результат та час виконання.

```
public async Task<(List<string> results, long timeMs)> SearchHospitalStatisticsAsync(
    string addressPattern, string diagnosisPattern)
{
    var results = new List<string>();
    var sw = Stopwatch.StartNew();

    var sql = $@"
    SELECT
        h.name,
        h.address,
        COUNT(DISTINCT a.patient_id) AS patient_count
    FROM hospital h
    JOIN doctor d ON h.id = d.hospital_id
    JOIN appointment a ON d.id = a.doctor_id
    WHERE
        h.address ILIKE @addressPattern AND
        a.diagnosis ILIKE @diagnosisPattern
    GROUP BY
        h.id, h.name, h.address
    ORDER BY
        patient_count DESC;
";

    await using var conn = await GetConnectionAsync();
    await using var cmd = new NpgsqlCommand(sql, conn);

    cmd.Parameters.AddWithValue("addressPattern", $"%{addressPattern}%");
    cmd.Parameters.AddWithValue("diagnosisPattern", $"%{diagnosisPattern}%");

    await using var reader = await cmd.ExecuteReaderAsync();
    while (await reader.ReadAsync())
    {
        results.Add(
            $"Лікарня: {reader["name"]} ({reader["address"]}), " +
            $"Кількість унікальних пацієнтів: {reader["patient_count"]}"
        );
    }

    sw.Stop();
    return (results, sw.ElapsedMilliseconds);
}
```

Висновок

Під час виконання цієї розрахунково-графічної роботи було успішно розроблено та реалізовано консольний додаток на мові C# для взаємодії з базою даних «Медична система», створеною в СУБД PostgreSQL.

Проєкт було побудовано з чітким дотриманням архітектурного шаблону **MVC (Model-View-Controller)**, де кожен компонент було винесено в окремий файл: Model інкапсулював всю логіку роботи з базою, View відповідав за консольний інтерфейс, а Controller керував логікою програми. Взаємодія з базою даних відбувалася виключно засобами **"чистого" SQL** з використанням бібліотеки Npgsql, без залучення ORM, як того вимагало завдання.

У ході роботи було реалізовано повний набір **CRUD-операцій** (Create, Read, Update, Delete) для всіх сутностей бази даних (patient, doctor, hospital, appointment). Особливу увагу було приділено коректній обробці **обмежень цілісності даних**:

1. Реалізовано перехоплення (try...catch) специфічних винятків PostgresException (код 23503) при спробі **видалення** "батьківських" записів (наприклад, hospital), що мають залежні "дочірні" записи (doctor).
2. Аналогічно оброблено помилки при вставці "дочірніх" записів (appointment) з неіснуючими зовнішніми ключами (patient_id або doctor_id).

Це дозволило уникнути аварійного завершення програми та виводити користувачу зрозумілі повідомлення про помилки.

Також було реалізовано функціонал **пакетної генерації** великої кількості (10 000+) реалістичних, рандомізованих даних. Ця операція виконується одним складним SQL-запитом, який використовує CTE (WITH), CROSS JOIN LATERAL та RETURNING id для коректної обробки зовнішніх ключів безпосередньо на стороні сервера.

На згенерованих даних було протестовано **три складні пошукові запити**, які, згідно з завданням, використовують JOIN кількох таблиць, фільтрацію WHERE за параметрами користувача та агрегацію GROUP BY (COUNT, COUNT(DISTINCT)). Програма вимірює та виводить час виконання кожного з цих запитів у мілісекундах.

У результаті було здобуто практичні навички з проєктування архітектури додатків (MVC), роботи з .NET-драйверами баз даних (Npgsql), написання складних SQL-запитів, реалізації надійного механізму обробки помилок

цілісності даних та тестування продуктивності запитів на великих обсягах даних.