

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського» ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ Кафедра системного програмування та спеціалізованих комп'ютерних систем

Лабораторна робота №2

з дисципліни «Бази даних і засоби управління»

Виконав: студент III курсу ФПМ групи КВ-13 Ольховський Максим Олександрович

Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання роботи полягає у наступному:

- 1. Перетворити модуль "Модель" з шаблону MVC РГР у вигляд об'єктно-реляційної проекції (ORM).
- 2. Створити та проаналізувати різні типи індексів у PostgreSQL.
- 3. Розробити тригер бази даних PostgreSQL.
- 4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

Концептуальна модель предметної галузі "Система обліку наявності медичних препаратів у аптеках"

У даній моделі предметної галузі "Система обліку наявності медичних препаратів у аптеках" (Малюнок 1) я виділив наступні сутності та зв'язки між ними:

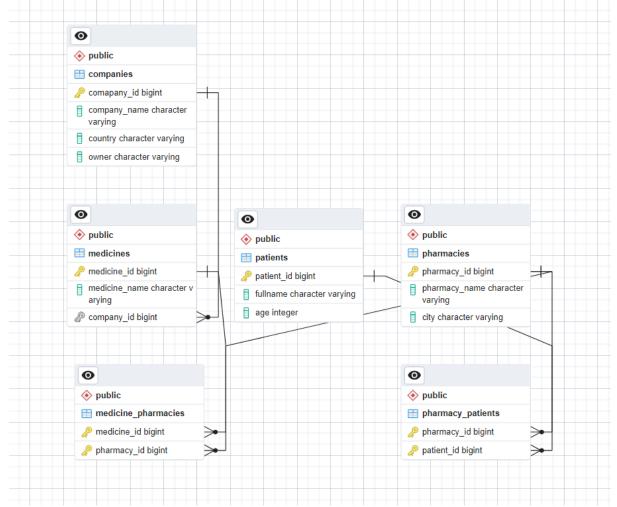
- 1. Сутність "Компанії" з атрибутами: ID, Назва, Країна, Власник;
- 2. Сутність "Ліки" з атрибутами: ІD, Назва, Виробник;
- 3. Сутність "Аптеки" з атрибутами: ID, Місто, Назва;
- 4. Сутність "Пацієнти" з атрибутами: ID, ПІБ, Вік.

Одна компанія може створити декілька варіантів ліків, тому між сутностями "Компанії" та "Ліки" зв'язок R(1:N).

Один вид ліків може направитися в багато аптек і багато видів ліків можуть направитися в одну аптеку, тому зв'язок між сутностями "Ліки" та "Аптеки" R(N:M).

Один пацієнт може відвідати багато аптек і одну аптеку можуть відвідати багато пацієнтів, тому зв'язок між сутностями "Аптеки" та "Пацієнти" R(N:M).

Діаграма сутність-зв'язок та структура бази даних

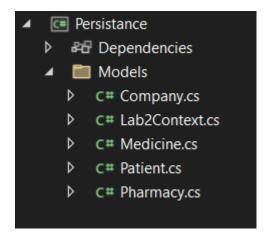


Перетворити модуль "Модель" з шаблону МVС РГР у вигляд об'єктно-реляційної проекції (ORM)

Для перетворення бази даних було використано EntityFrameworkCore. Щоб пов'язати класи сутності з базою даних було вибрано підхід Database First, дже ми вже маємо спроектувану базу даних. Щоб створити класи сутностей бази даних використовують команду scaffold. Приклад команди:

dotnet ef dbcontext scaffold "User ID=postgres;Password=pass12345;Host=localhost;Port=5433;Database=Lab2;" Npgsql.EntityFrameworkCore.PostgreSQL -o Models

Дана команда автоматично генерує класи всіх сутностей і також клас контексту (базовий клас, який працює з базою даних), який відповідає нашій базі даних. Після скафолдингу ми отримали такі файли:



Код контексту, який описує всі зв'язки і деталі кожного поля:

```
using Microsoft.EntityFrameworkCore;
namespace Persistance. Models;
public partial class Lab2Context : DbContext
  public Lab2Context()
  public Lab2Context(DbContextOptions<Lab2Context> options)
    : base(options)
  public virtual DbSet<Company> Companies { get; set; }
  public virtual DbSet<Medicine> Medicines { get; set; }
  public virtual DbSet<Patient> Patients { get; set; }
  public virtual DbSet<Pharmacy> Pharmacies { get; set; }
  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
#warning To protect potentially sensitive information in your connection string, you should move it out of
source code. You can avoid scaffolding the connection string by using the Name= syntax to read it from
configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection
strings, see https://go.microsoft.com/fwlink/?LinkId=723263.
    => optionsBuilder.UseNpgsql("User
ID=postgres;Password=pass12345;Host=localhost;Port=5433;Database=Lab2;");
  protected override void OnModelCreating(ModelBuilder modelBuilder)
    modelBuilder.Entity<Company>(entity =>
       entity.HasKey(e => e.CompanyId).HasName("companies_pkey");
       entity.ToTable("companies");
       entity.Property(e => e.CompanyId).HasColumnName("company_id");
       entity.Property(e => e.CompanyName)
         .HasColumnType("character varying")
```

```
.HasColumnName("company_name");
  entity.Property(e => e.Country)
    .HasColumnType("character varying")
    .HasColumnName("country");
  entity.Property(e => e.Owner)
    .HasColumnType("character varying")
    .HasColumnName("owner");
});
modelBuilder.Entity<Medicine>(entity =>
  entity.HasKey(e => e.MedicineId).HasName("medicines pkey");
  entity.ToTable("medicines");
  entity.Property(e => e.MedicineId).HasColumnName("medicine id");
  entity.Property(e => e.CompanyId).HasColumnName("company_id");
  entity.Property(e => e.MedicineName)
    .HasColumnType("character varying")
    .HasColumnName("medicine_name");
  entity.HasOne(d => d.Company).WithMany(p => p.Medicines)
    .HasForeignKey(d => d.CompanyId)
    .OnDelete(DeleteBehavior.ClientSetNull)
    .HasConstraintName("medicines_company_id_fkey");
  entity.HasMany(d => d.Pharmacies).WithMany(p => p.Medicines)
    .UsingEntity<Dictionary<string, object>>(
       "MedicinePharmacy",
      r => r.HasOne<Pharmacy>().WithMany()
         .HasForeignKey("PharmacyId")
         .OnDelete(DeleteBehavior.ClientSetNull)
         .HasConstraintName("medicine pharmacies pharmacy id fkey"),
      1 => 1.HasOne<Medicine>().WithMany()
         .HasForeignKey("MedicineId")
         .OnDelete(DeleteBehavior.ClientSetNull)
         .HasConstraintName("medicine_pharmacies_medicine_id_fkey"),
         j.HasKey("MedicineId", "PharmacyId").HasName("medicine_pharmacies_pkey");
         j.ToTable("medicine_pharmacies");
         j.IndexerProperty<long>("MedicineId").HasColumnName("medicine_id");
         j.IndexerProperty<long>("PharmacyId").HasColumnName("pharmacy_id");
       });
});
modelBuilder.Entity<Patient>(entity =>
  entity.HasKey(e => e.PatientId).HasName("patients_pkey");
  entity.ToTable("patients");
  entity.Property(e => e.PatientId).HasColumnName("patient id");
  entity.Property(e => e.Age).HasColumnName("age");
  entity.Property(e => e.Fullname)
    .HasColumnType("character varying")
    .HasColumnName("fullname");
});
modelBuilder.Entity<Pharmacy>(entity =>
```

```
entity.HasKey(e => e.PharmacyId).HasName("pharmacies_pkey");
    entity.ToTable("pharmacies");
    entity.Property(e => e.PharmacyId).HasColumnName("pharmacy_id");
    entity.Property(e => e.City)
      .HasColumnType("character varying")
       .HasColumnName("city");
    entity.Property(e => e.PharmacyName)
       .HasColumnType("character varying")
       .HasColumnName("pharmacy name");
    entity.HasMany(d \Rightarrow d.Patients).WithMany(p \Rightarrow p.Pharmacies)
       .UsingEntity<Dictionary<string, object>>(
         "PharmacyPatient",
         r => r.HasOne<Patient>().WithMany()
           .HasForeignKey("PatientId")
           .OnDelete(DeleteBehavior.ClientSetNull)
           .HasConstraintName("pharmacy_patients_patient_id_fkey"),
         1 => 1.HasOne<Pharmacy>().WithMany()
           .HasForeignKey("PharmacyId")
           .OnDelete(DeleteBehavior.ClientSetNull)
           .HasConstraintName("pharmacy_patients_pharmacy_id_fkey"),
        j =>
           j.HasKey("PharmacyId", "PatientId").HasName("pharmacy_patients_pkey");
           j.ToTable("pharmacy_patients");
           j.IndexerProperty<long>("PharmacyId").HasColumnName("pharmacy_id");
           j.IndexerProperty<long>("PatientId").HasColumnName("patient_id");
         });
  });
  OnModelCreatingPartial(modelBuilder);
partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
```

Більш детально розглянемо сутність, яка реалізує 1-M та M-N зв'язки, а саме Medicine.

```
namespace Persistance.Models;

public partial class Medicine {
   public long MedicineId { get; set; }
   public string MedicineName { get; set; } = null!;

   public long CompanyId { get; set; }

   public virtual Company Company { get; set; } = null!;

   public virtual ICollection<Pharmacy> Pharmacies { get; set; } = new List<Pharmacy>();
}
```

Як можемо побачити з класу, ми маємо доступ до сутності Company напряму через сутність класу. Також через колекцію Pharmacies ми маємо доступ до всіх сутностей Pharmacy через таблицю medicine pharmacies.

Створити та проаналізувати різні типи індексів у PostgreSQL

13 BTree, BRIN	before insert, delete
----------------	-----------------------

Індекс BTree

Індекс ВТree ϵ найбільш поширеним типом індексу для ефективного зберігання та отримання даних у PostgreSQL. Це типовий тип індексу. Кожного разу, коли ми використовуємо команду CREATE INDEX без вказівки типу індексу, PostgreSQL створить індекс ВТree для таблиці або стовпця.

Індекс ВТгее організований у вигляді структури подібної до дерева. Індекс починається з кореневого вузла, з вказівниками на дочірні вузли. Кожен вузол в дереві, як правило, містить кілька пар ключ-значення, де ключі використовуються для індексації, а значення вказують на відповідні дані в таблиці.

Індекс BRIN

BRIN, або Block Range Index, - це тип індексу в PostgreSQL, призначений для забезпечення ефективного індексування великих таблиць з відсортованими даними. BRIN-індекс містить мінімум і максимум у групі сторінок бази даних.

Індекс BRIN - це найпростіший спосіб оптимізації швидкості. Він особливо корисний для даних, які мають послідовні або відсортовані характеристики, наприклад, дані часових рядів або дані з природним впорядкуванням.

Ось огляд індексу BRIN:

- 1. Індекси BRIN розбивають таблицю на логічні блоки і зберігають підсумкову інформацію про кожен блок.
- 2. Кожен блок містить діапазон значень, а індекс зберігає мінімальні та максимальні значення в межах кожного блоку.
- 3. Замість того, щоб зберігати окремі індексні записи для кожного рядка, індекси BRIN зберігають підсумки на рівні блоків, що робить їх меншими за розміром порівняно з іншими типами індексів.
- 4. Індекси BRIN добре працюють, коли дані відсортовані або коли послідовне сканування ϵ більш ефективним, ніж індексне.

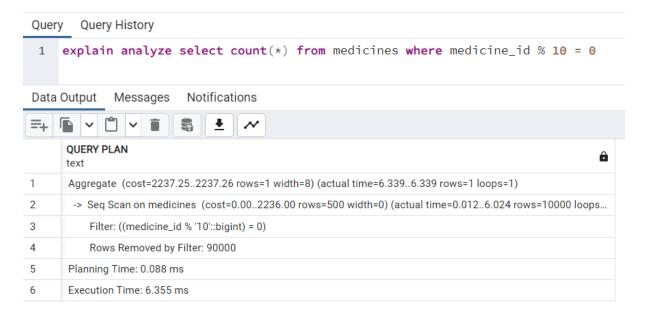
Тестування

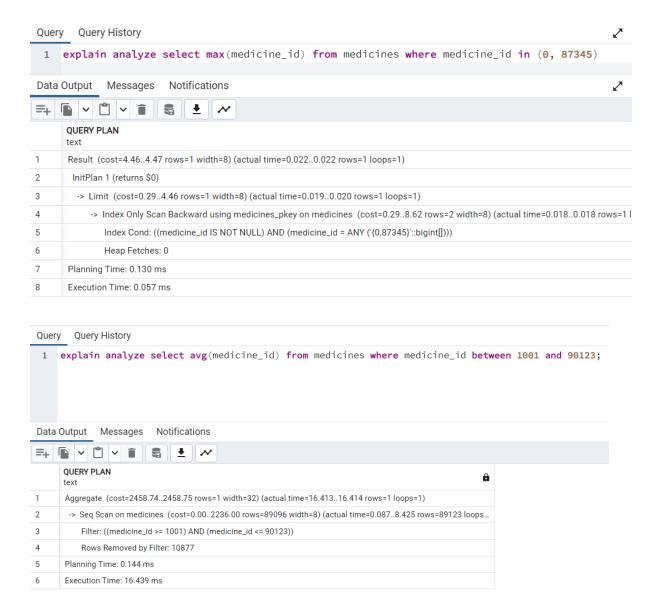
```
Enter count to generate 100000

Generated 100000 records in table medicines

Press to continue...
```

Виконаємо такі команди для тестування індексів і відразу подивимось на їх швидкодію:



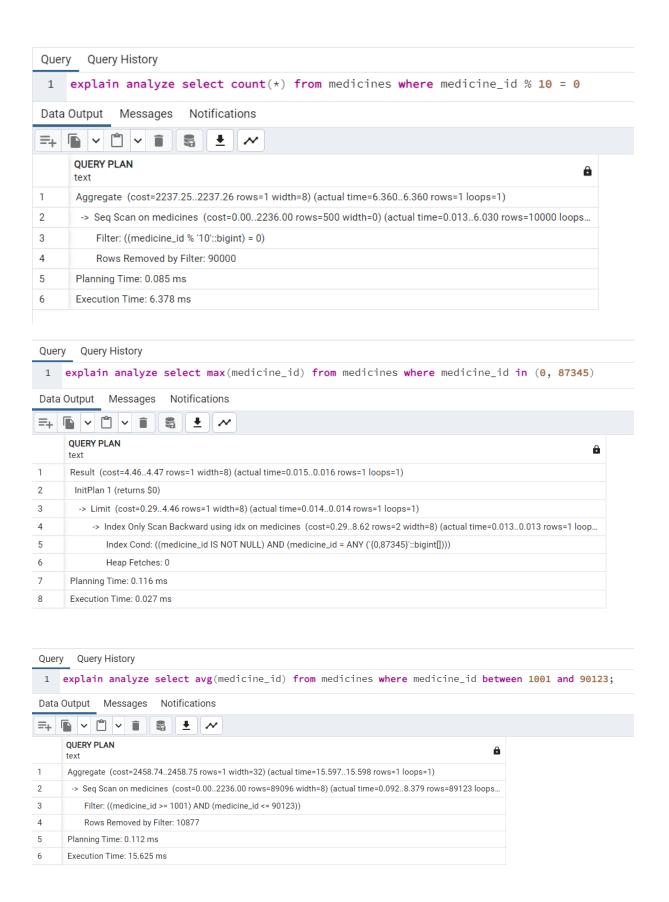


B-Tree

Створимо індекс:



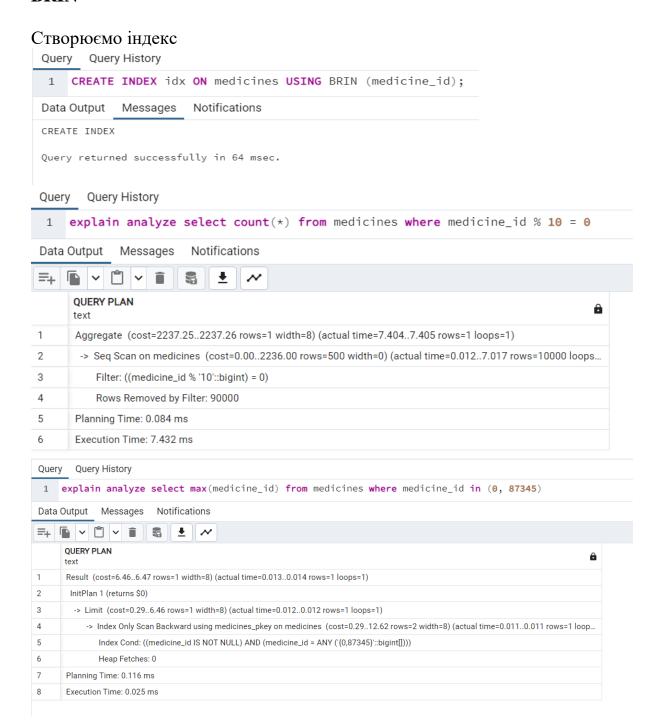
Проведемо тестування кожного запиту ще раз:



Спочатку розглянемо другий випадок. Як можемо побачити приріст швидкодії запиту ϵ і доволі значний і відсотках, але не значний у реальному часі. Це і поясню ϵ нам, чому обидва інші запити не мають

значного приросту або взагалі не мають приросту по швидкодії, адже індекс свою роботу і швидкодії порівняння виконав, але основну частину часу займає агрегатна функція.

BRIN





Схожа ситуація як і з індесом В-Тree. Обидва індекси прискорюють операції порівняння, тому доводи схожі як і результати.

Мій телеграм: https://t.me/Maks_01111

Мій GitHub: https://github.com/fantomas00/Laba2 DB