**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 ІII курсу

ФПМ групи КВ-13

Чоловенко Дмитро Володимирович

Telegram: @D\_4ubaka

GitHub: https://github.com/D-Cholik/DB\_Lab2.git

Перевірив:

**Київ – 2023**

**Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL**

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

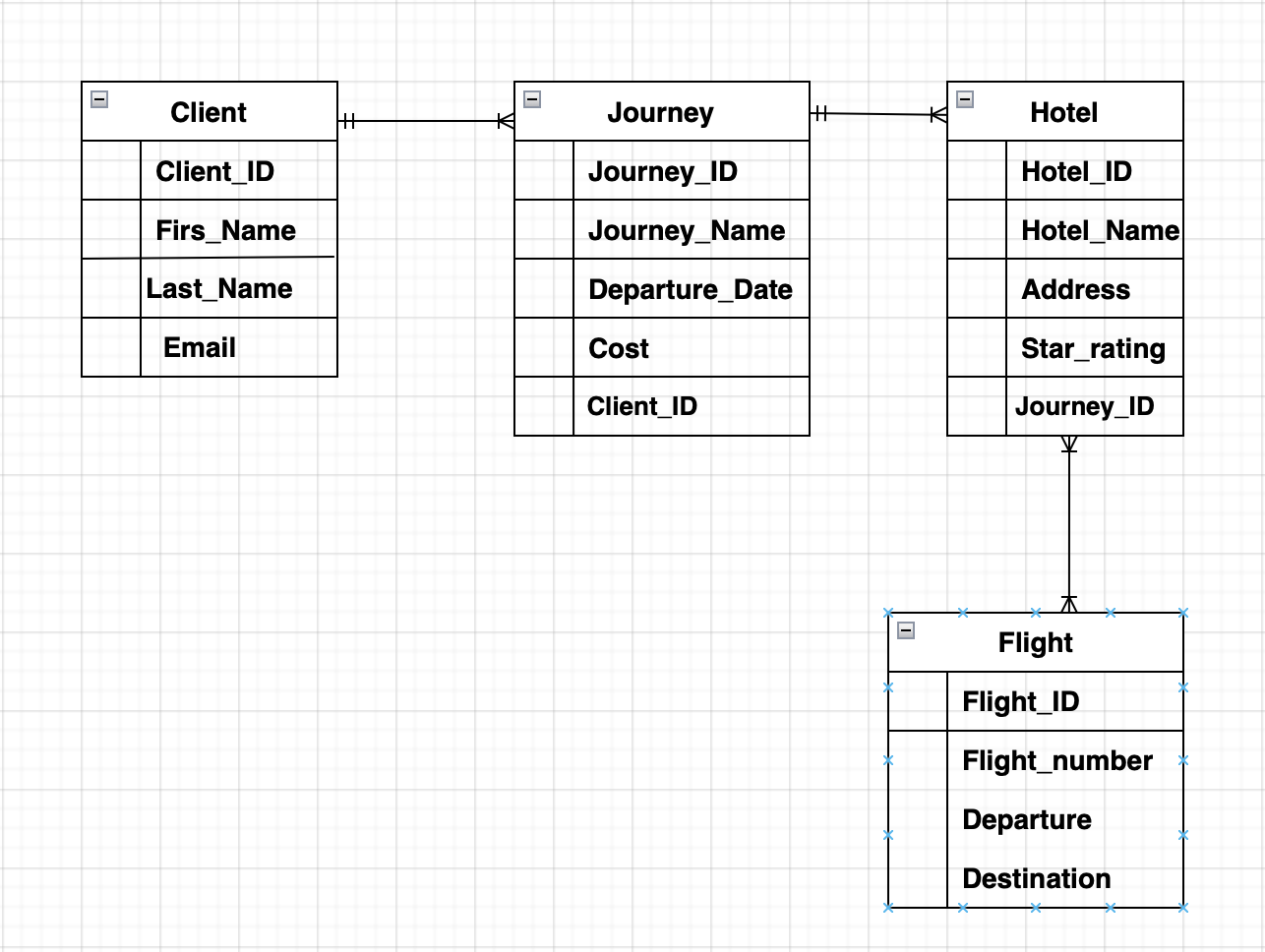
*Завдання* роботи полягає у наступному:

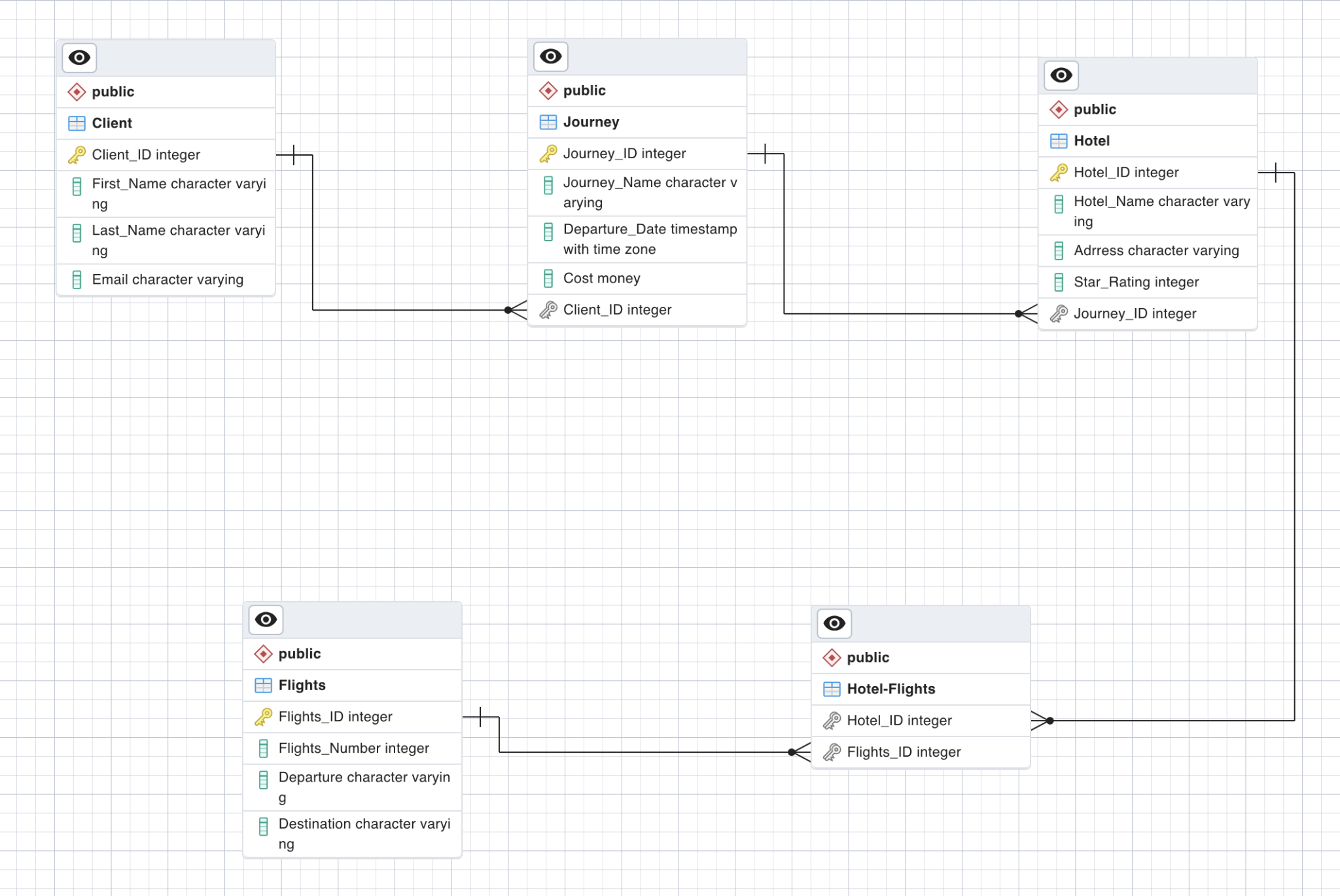
1. Перетворити модуль “Модель” з шаблону MVC РГР у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

**Короткий опис бази даних**

Між Готелями і Рейсами існує зв'язок "багато до багатьох" (M:N). Один готель може відповідати багатьом рейсам, і навпаки, один рейс може мати відношення до багатьох готелів. Для вираження цього зв'язку в базі даних потрібно створити додаткову таблицю яка буде містити зв'язки між готелями і рейсами за допомогою зовнішніх ключів, що посилаються на ID готелю і ID рейсу відповідно.

**Діаграма сутність-зв’язок та структура бази даних**

****

****

**Перетворити модуль “Модель” з шаблону MVC РГР у вигляд об’єктно-реляційної проекції (ORM)**

Технологія EntityFrameworkCore. Сутності мають звязки M-N та 1-M через колекцію об'єктів та об'єкт.

**Контекст:**

using System;

using System.Collections.Generic;

using Microsoft.EntityFrameworkCore;

namespace Persistance.Entities;

public partial class Lab2Context : DbContext

{

public Lab2Context()

{

}

public Lab2Context(DbContextOptions<Lab2Context> options)

: base(options)

{

}

public virtual DbSet<Client> Clients { get; set; }

public virtual DbSet<Flight> Flights { get; set; }

public virtual DbSet<Hotel> Hotels { get; set; }

public virtual DbSet<Journey> Journeys { 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("host=localhost;port=5433;database=Lab2;user id=postgres;password=pass12345");

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Client>(entity =>

{

entity.HasKey(e => e.ClientId).HasName("Clients\_pkey");

entity.Property(e => e.ClientId).HasColumnName("Client\_ID");

entity.Property(e => e.Email).HasMaxLength(256);

entity.Property(e => e.FirstName)

.HasMaxLength(100)

.HasColumnName("First\_Name");

entity.Property(e => e.LastName)

.HasMaxLength(100)

.HasColumnName("Last\_Name");

});

modelBuilder.Entity<Flight>(entity =>

{

entity.HasKey(e => e.FlightId).HasName("Flights\_pkey");

entity.Property(e => e.FlightId).HasColumnName("Flight\_ID");

entity.Property(e => e.Departure).HasColumnType("timestamp without time zone");

entity.Property(e => e.FlightNumber).HasColumnName("Flight\_Number");

});

modelBuilder.Entity<Hotel>(entity =>

{

entity.HasKey(e => e.HotelId).HasName("Hotels\_pkey");

entity.Property(e => e.HotelId).HasColumnName("Hotel\_ID");

entity.Property(e => e.Address).HasMaxLength(100);

entity.Property(e => e.HotelName)

.HasMaxLength(100)

.HasColumnName("Hotel\_Name");

entity.Property(e => e.JourneyId).HasColumnName("Journey\_ID");

entity.Property(e => e.StarRating).HasColumnName("Star\_Rating");

entity.HasOne(d => d.Journey).WithMany(p => p.Hotels)

.HasForeignKey(d => d.JourneyId)

.HasConstraintName("Hotels\_Journey\_ID\_fkey");

entity.HasMany(d => d.Flights).WithMany(p => p.Hotels)

.UsingEntity<Dictionary<string, object>>(

"HotelFlight",

r => r.HasOne<Flight>().WithMany()

.HasForeignKey("FlightId")

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("Hotel-Flights\_Flight\_ID\_fkey"),

l => l.HasOne<Hotel>().WithMany()

.HasForeignKey("HotelId")

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("Hotel-Flights\_Hotel\_ID\_fkey"),

j =>

{

j.HasKey("HotelId", "FlightId").HasName("Hotel-Flights\_pkey");

j.ToTable("Hotel-Flights");

j.IndexerProperty<long>("HotelId").HasColumnName("Hotel\_ID");

j.IndexerProperty<long>("FlightId").HasColumnName("Flight\_ID");

});

});

modelBuilder.Entity<Journey>(entity =>

{

entity.HasKey(e => e.JourneyId).HasName("Journeys\_pkey");

entity.Property(e => e.JourneyId).HasColumnName("Journey\_ID");

entity.Property(e => e.ClientId).HasColumnName("Client\_ID");

entity.Property(e => e.Cost).HasColumnType("money");

entity.Property(e => e.DepartureDate)

.HasColumnType("timestamp without time zone")

.HasColumnName("Departure\_Date");

entity.Property(e => e.JourneyName)

.HasMaxLength(100)

.HasColumnName("Journey\_Name");

entity.HasOne(d => d.Client).WithMany(p => p.Journeys)

.HasForeignKey(d => d.ClientId)

.OnDelete(DeleteBehavior.ClientSetNull)

.HasConstraintName("Journeys\_Client\_ID\_fkey");

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

**Сутності:**

using System;

using System.Collections.Generic;

namespace Persistance.Entities;

public partial class Client

{

public long ClientId { get; set; }

public string FirstName { get; set; } = null!;

public string LastName { get; set; } = null!;

public string? Email { get; set; }

public virtual ICollection<Journey> Journeys { get; set; } = new List<Journey>();

}

using System;

using System.Collections.Generic;

namespace Persistance.Entities;

public partial class Flight

{

public long FlightId { get; set; }

public int FlightNumber { get; set; }

public DateTime Departure { get; set; }

public int Destination { get; set; }

public virtual ICollection<Hotel> Hotels { get; set; } = new List<Hotel>();

}

using System;

using System.Collections.Generic;

namespace Persistance.Entities;

public partial class Hotel

{

public long HotelId { get; set; }

public string HotelName { get; set; } = null!;

public string Address { get; set; } = null!;

public int? StarRating { get; set; }

public long? JourneyId { get; set; }

public virtual Journey? Journey { get; set; }

public virtual ICollection<Flight> Flights { get; set; } = new List<Flight>();

}

using System;

using System.Collections.Generic;

namespace Persistance.Entities;

public partial class Journey

{

public long JourneyId { get; set; }

public string JourneyName { get; set; } = null!;

public DateTime DepartureDate { get; set; }

public decimal Cost { get; set; }

public long ClientId { get; set; }

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

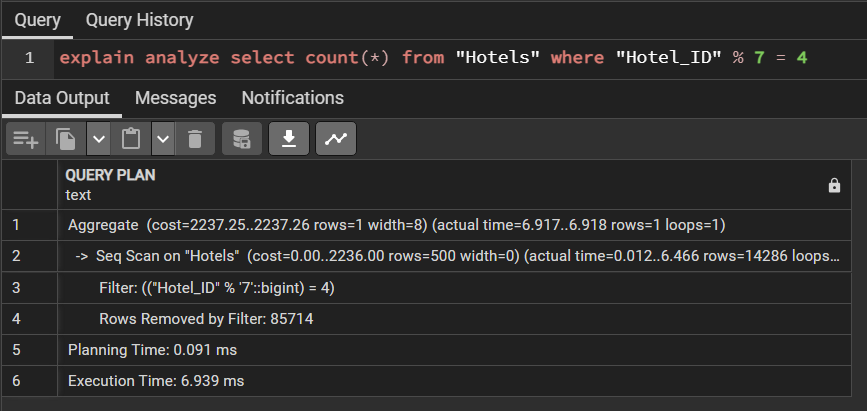
public virtual ICollection<Hotel> Hotels { get; set; } = new List<Hotel>();

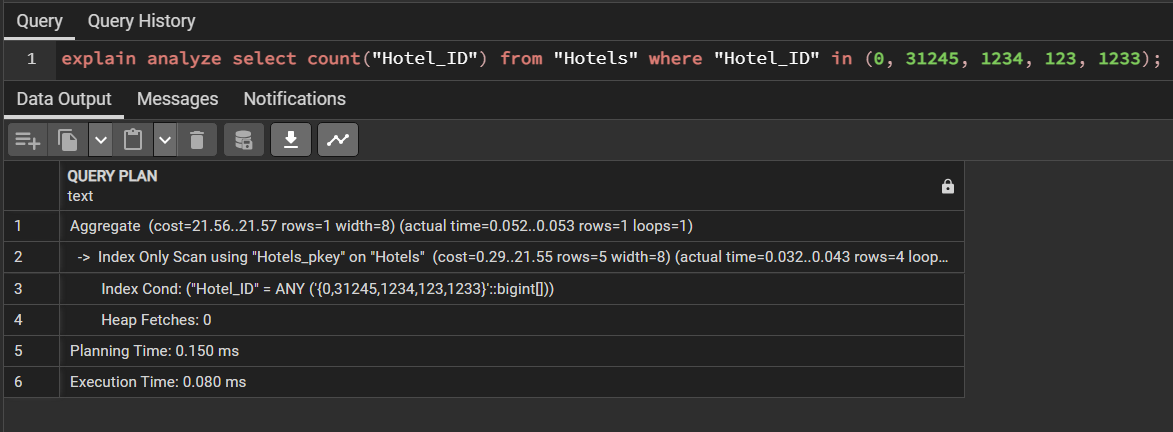
}

**Створити та проаналізувати різні типи індексів у PostgreSQL**

|  |  |  |
| --- | --- | --- |
| *21* | *Btree, Hash* | *before delete, update* |

**Тестування без індексів**

****

****

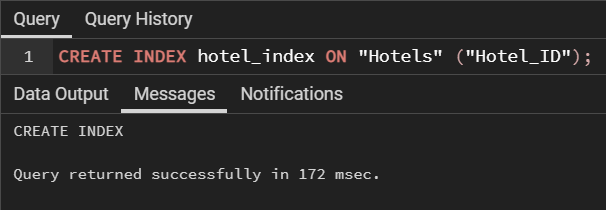
****

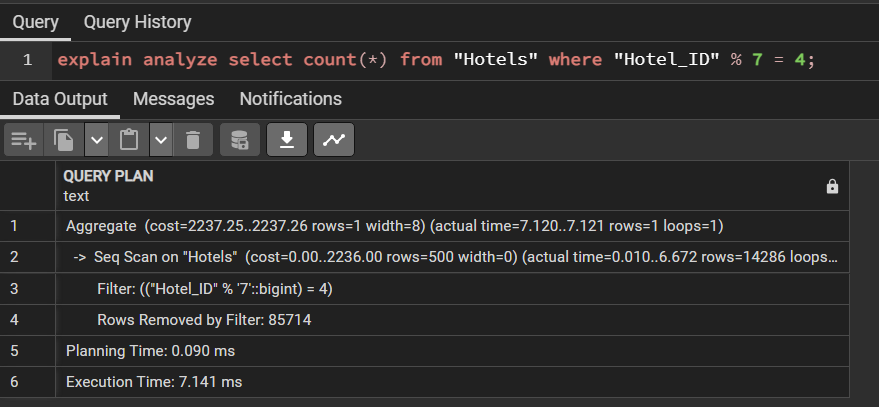
**Індекс BTree**

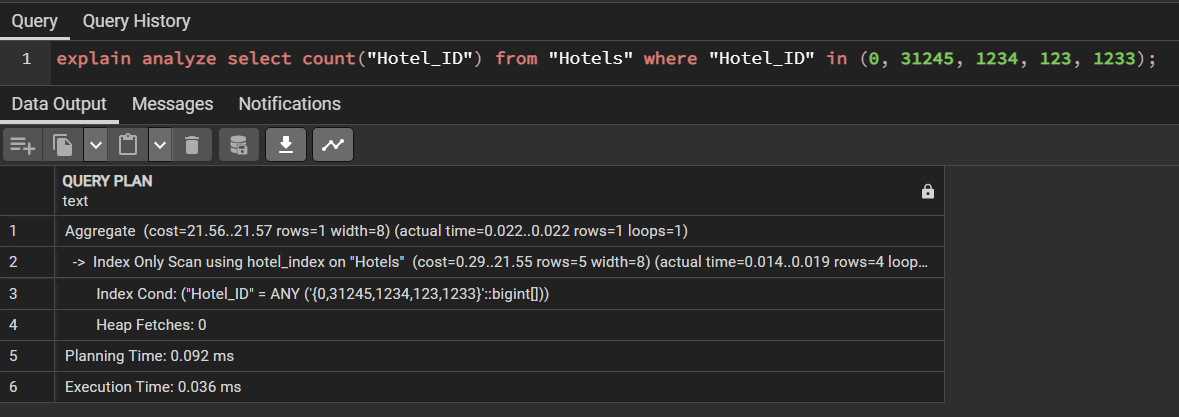
Індекс **BTree** є найбільш поширеним типом індексу для ефективного зберігання та отримання даних у PostgreSQL. Це типовий вид індексу. Кожного разу, коли використовується команда CREATE INDEX без вказівки типу індексу, PostgreSQL автоматично створить індекс B-дерева для таблиці або стовпця.

Індекс **BTree** організований у вигляді структури, схожої на дерево. Починаючи з кореневого вузла із вказівниками на дочірні вузли, кожен вузол дерева, як правило, містить набір ключ-значення. Ключі використовуються для індексації, а значення вказують на відповідні дані в таблиці.

Створимо індекс:







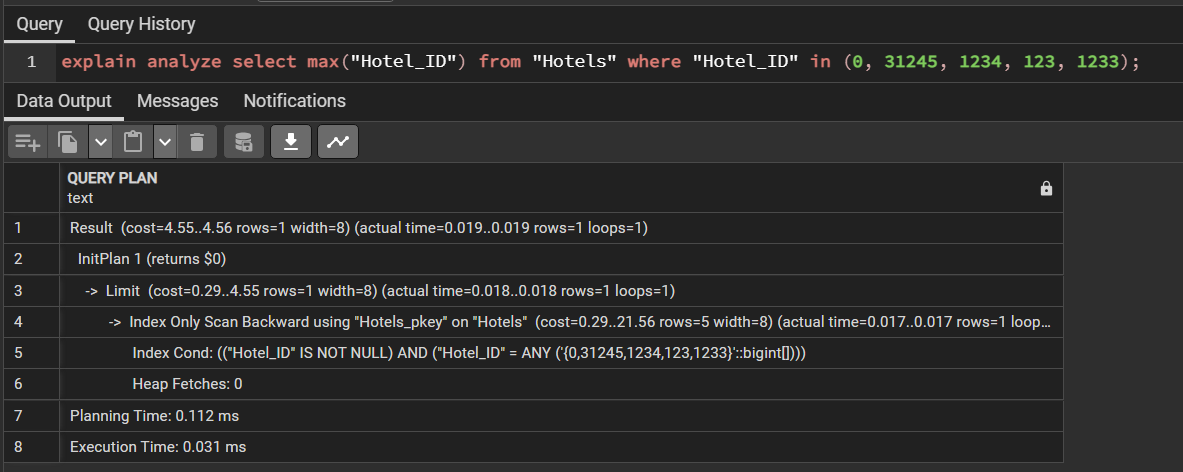


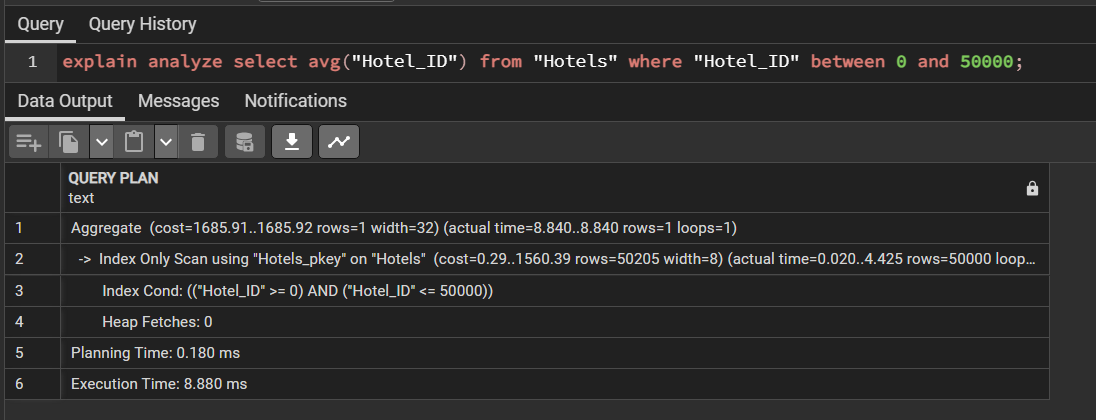
**Індекс HASH**

Хеш-індекси розроблені для швидкого пошуку значень за ключем. Коли умова запиту передбачає порівняння на рівність з індексованими стовпцями, хеш-індекси можуть забезпечити надзвичайно швидке витягування, оскільки хеш-функція безпосередньо визначає місцезнаходження потрібних даних. Ці індекси найбільш ефективні для операцій порівняння на рівність, таких як операції = або IN.

Хоча схожі на інші типи індексів, хеш-індекси повинні підтримувати модифікації даних (вставки, оновлення та видалення), щоб забезпечити консистентність даних. Проте обслуговування хеш-індексів може бути витратнішим через необхідність вирішення колізій та перехешування даних порівняно з обслуговуванням індексів B-дерева.

****

****

****

**Висновки**

Можна побачити, що швидкодія запитів приблизно однакова і особливого приросту ми не отримали. Якщо дивитись на звичайний запит без агрегаційних функцій, то можна побачити, що приріст є, але не критичний. В інших випадках все впирається в функції. При 100000 записах особливих проблем не спостерігається, тому, що швидкодія достатня навіть без індексів, але як показують результати, індекси в даній ситуації як мінімум не роблять гірше, а навіть покращують.