**Lab 1: Understanding ORM with a Retail Inventory System**

**1. What is ORM?**

**ORM** is a programming technique that allows you to interact with a relational database using object-oriented code. Instead of writing raw SQL, you work with **C# classes and objects**, and the ORM handles translating these into SQL queries.

**Mapping C# to SQL**

| **C# Class/Property** | **SQL Equivalent** |
| --- | --- |
| class Product | Products table |
| int Id | INT primary key |
| string Name | VARCHAR or NVARCHAR |
| decimal Price | DECIMAL |

ORM automatically handles:

* Table creation based on your classes.
* CRUD operations (Insert, Update, Delete, Select).
* Relationships (e.g., foreign keys) through navigation properties.

**Benefits of ORM**

* **Productivity**: Less boilerplate SQL, more focus on business logic.
* **Maintainability**: Changes in models automatically sync with DB schema (via migrations).
* **Abstraction**: No need to worry about SQL dialects or connection management.

**2. EF Core vs. EF Framework**

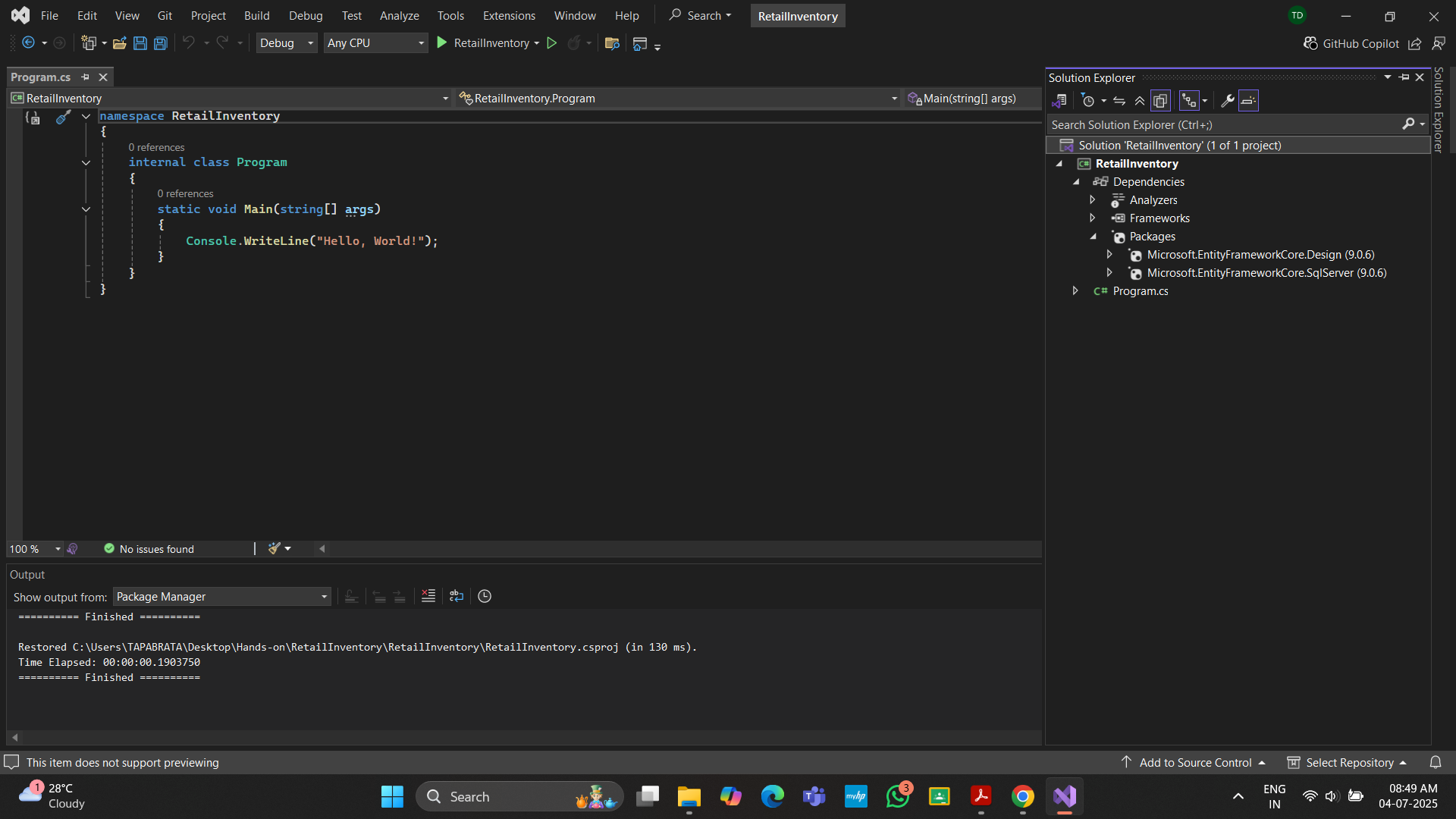
| **Feature** | **EF Core** | **Entity Framework (EF6)** |
| --- | --- | --- |
| Platform | Cross-platform | Windows-only |
| Performance | High (leaner, faster) | Slower but stable |
| Features | Async LINQ, compiled queries, JSON | Mature, but less modern |
| Support | Actively developed | In maintenance mode |
| Migration & Tooling | CLI & Visual Studio support | Visual Studio only |

**3. EF Core 8.0 Features**

EF Core 8.0 introduced powerful features:

* **JSON column mapping**: Store structured data like { "color": "red", "size": "L" } directly in a single column.
* **Compiled models**: Improve app startup time and runtime performance by pre-compiling metadata.
* **Interceptors**: Hook into EF Core's internal operations (e.g., logging or modifying queries).
* **Better bulk operations**: Improved performance for inserting/updating large datasets.

After creating the console app and installing Packages :



**Lab 2: Setting Up the Database Context for a Retail Store**

Category.cs :

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

public List<Product> Products { get; set; }

}

Product.cs :

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

}

AppDbContext.cs :

using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext

{

public DbSet<Product> Products { get; set; }

public DbSet<Category> Categories { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer(

"Server=.\\SQLEXPRESS;Database=RetailInventoryDB;Trusted\_Connection=True;

TrustServerCertificate=True;" // I have express edition of SQL SERVER

);

}

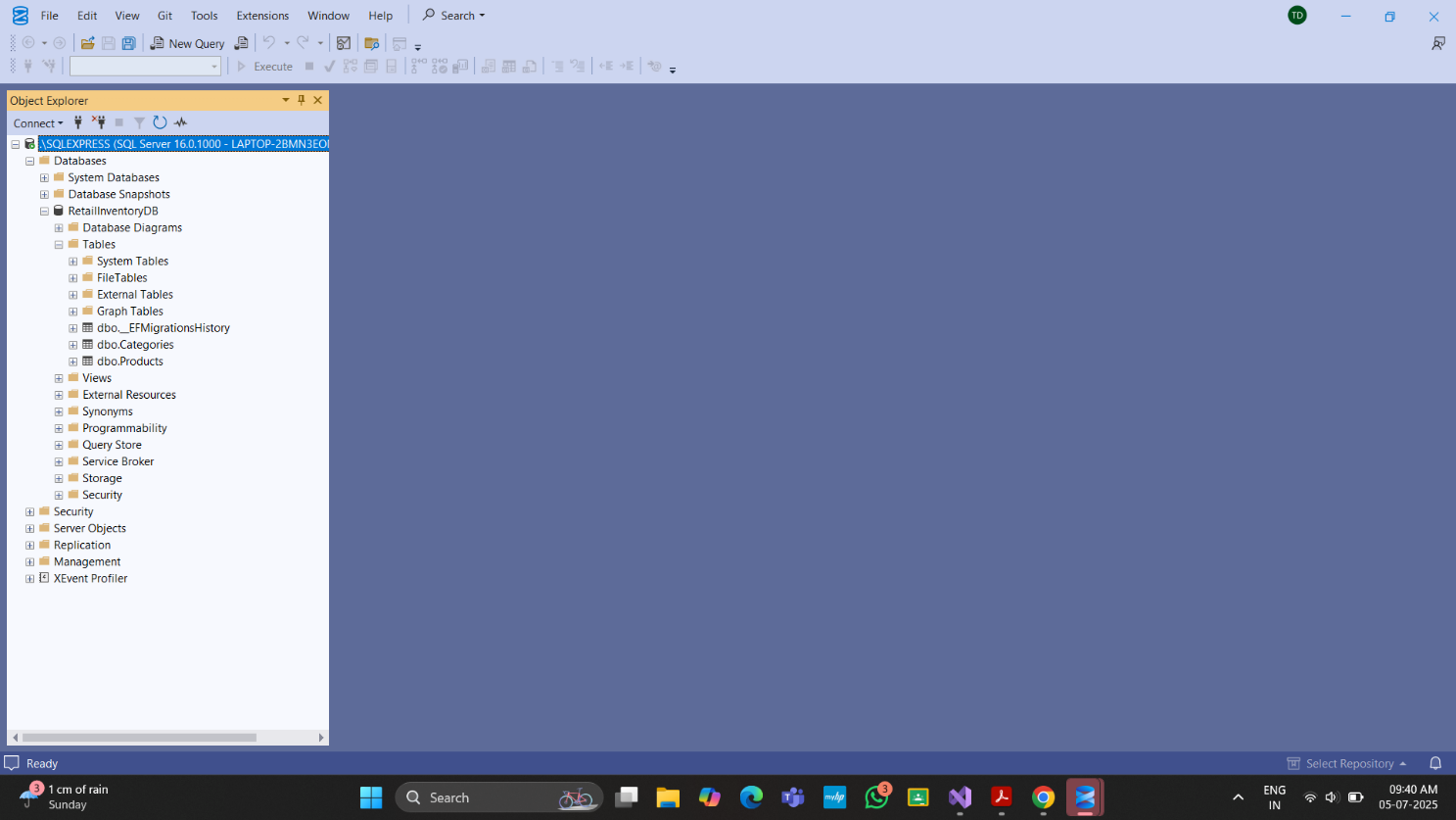
}

**Lab 3: Using EF Core CLI to Create and Apply Migrations**

For Migration: dotnet ef migrations add InitialCreate

For Updation: dotnet ef database update

Tables created :



**Lab 4: Inserting Initial Data into the Database**

Program.cs :

using System.Threading.Tasks;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

var electronics = new Category { Name = "Electronics" };

var groceries = new Category { Name = "Groceries" };

await context.Categories.AddRangeAsync(electronics, groceries);

var product1 = new Product { Name = "Laptop", Price = 75000, Category = electronics };

var product2 = new Product { Name = "Rice Bag", Price = 1200, Category = groceries };

await context.Products.AddRangeAsync(product1, product2);

await context.SaveChangesAsync();

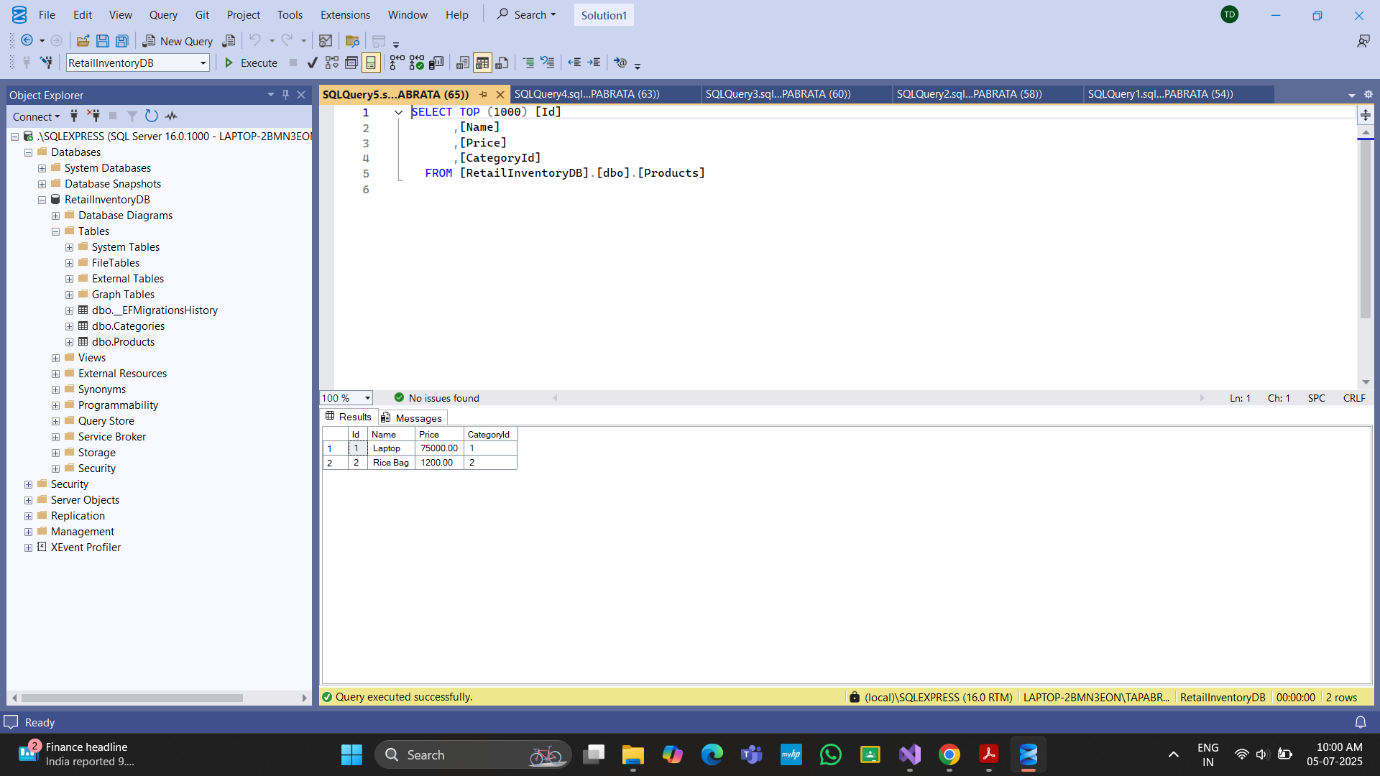
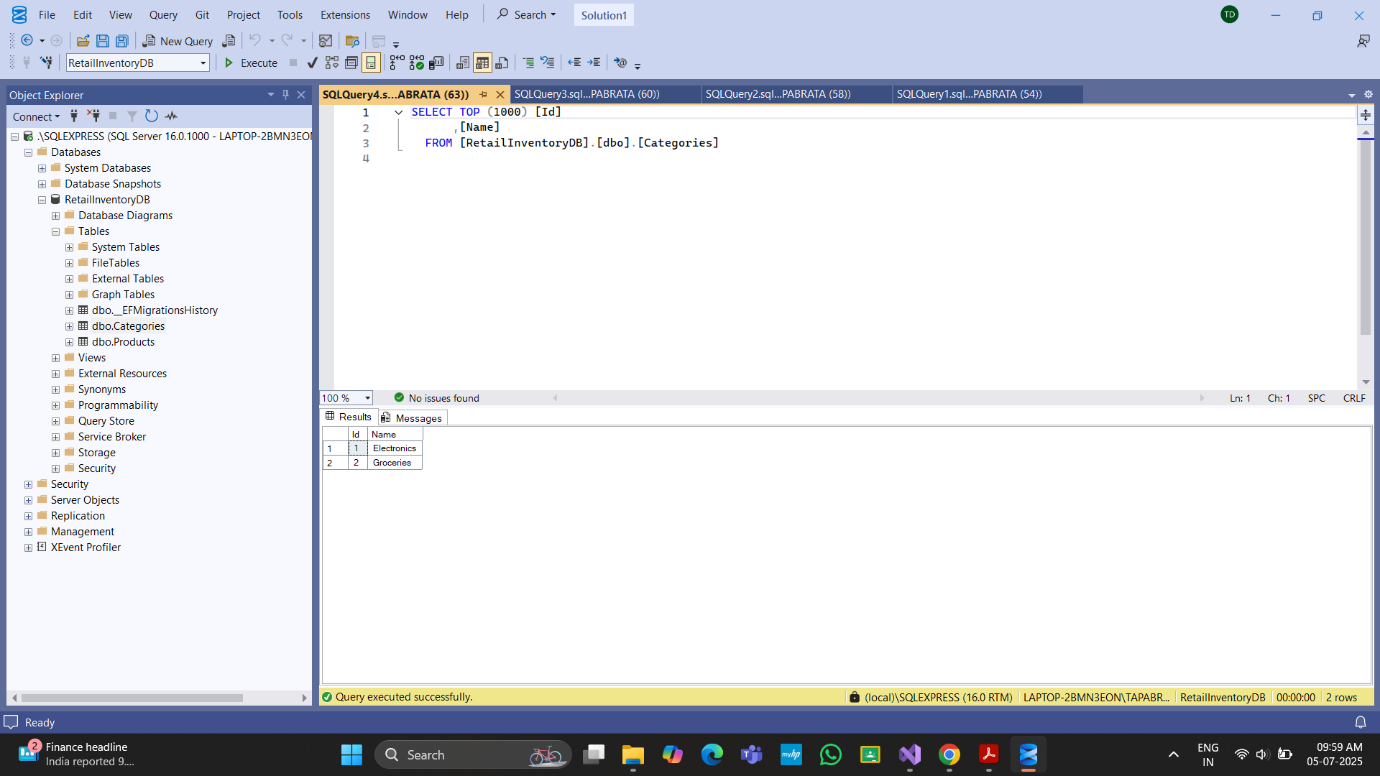
var service = new ProductService();

await service.DisplayDashboardDataAsync();

}

}

Data Inserted :



**Lab 5: Retrieving Data from the Database**

ProductService.cs :

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

public class ProductService

{

private readonly AppDbContext \_context;

public ProductService()

{

\_context = new AppDbContext();

}

public async Task DisplayDashboardDataAsync()

{

// 1. Retrieve All Products

var products = await \_context.Products.ToListAsync();

Console.WriteLine("All Products:");

foreach (var p in products)

Console.WriteLine($"{p.Name} - ₹{p.Price}");

Console.WriteLine();

// 2. Find by ID

var product = await \_context.Products.FindAsync(1);

Console.WriteLine($"Found by ID = 1: {product?.Name}");

Console.WriteLine();

// 3. FirstOrDefault with Condition

var expensive = await \_context.Products.FirstOrDefaultAsync(p => p.Price > 50000);

Console.WriteLine($"First Product with Price > ₹50000: {expensive?.Name}");

}

}

OutPut :

