WEEK-3

Entity Framework Core 8.0

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

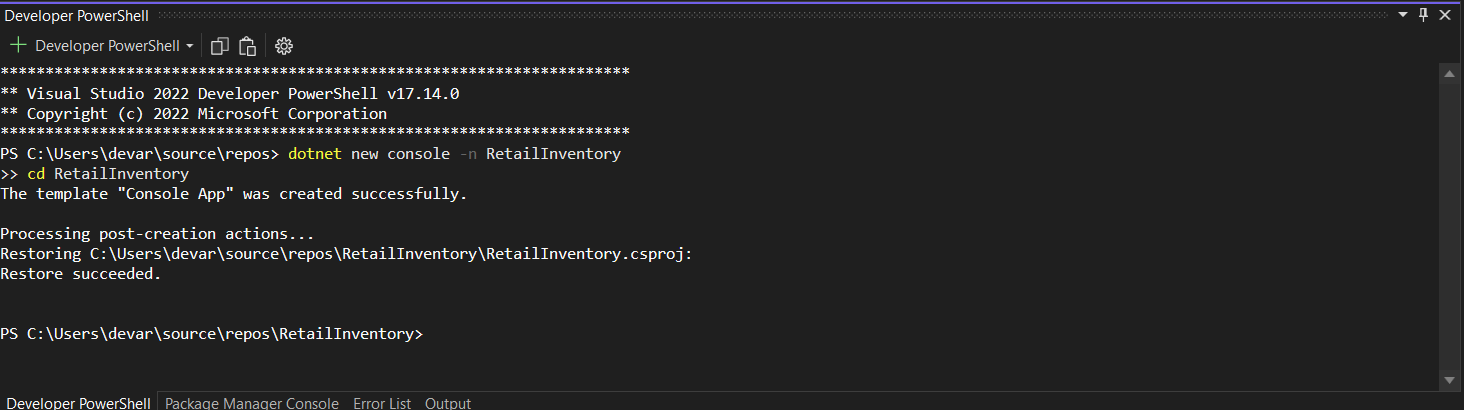
1. What is ORM?

An **Object-Relational Mapper (ORM)** is a programming technique that acts as a bridge between your application's code (like C# classes) and a relational database (like SQL Server).

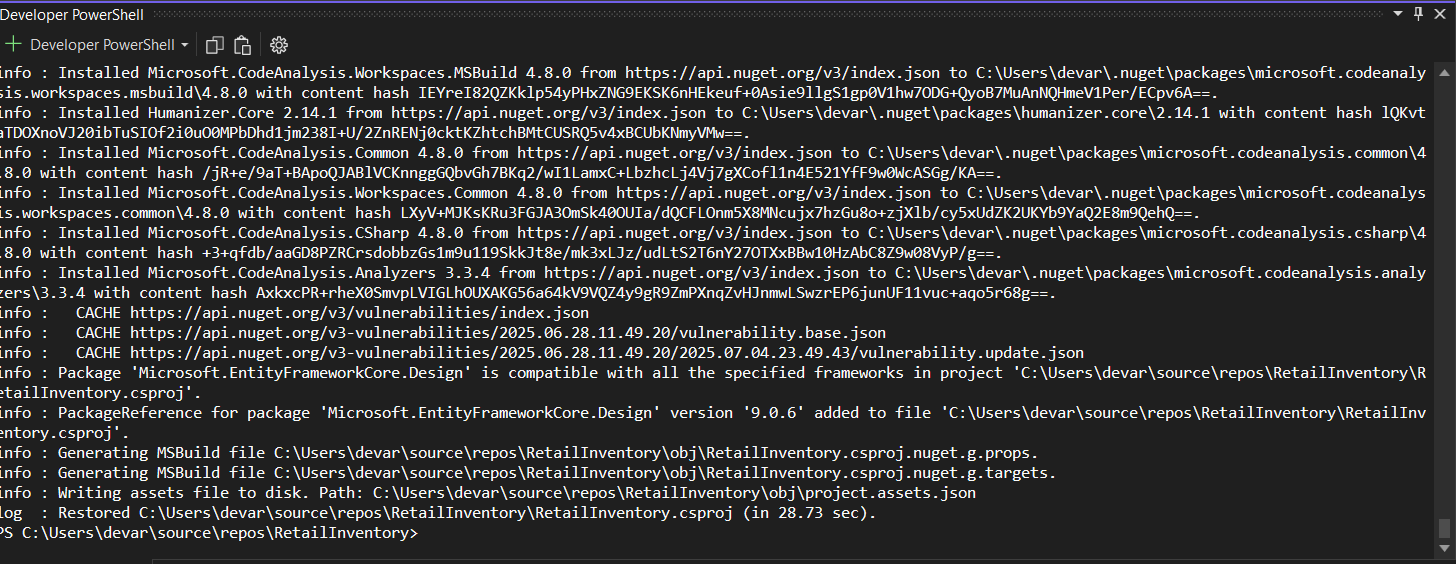
Instead of writing raw SQL queries to perform database operations (CREATE, READ, UPDATE, DELETE), you interact with your data as if it were regular C# objects. The ORM, in this case EF Core, handles the translation behind the scenes.

* **How it Works:** You define a C# class, say Product, and the ORM maps this class to a Products table in your database. The properties of the class (e.g., ProductId, Name, Price) correspond to the columns in the table.
* **Key Benefits:**
  + **Productivity:** You write less boilerplate code for data access, which speeds up development.
  + **Maintainability:** Your data access logic is cleaner and easier to manage because it's written in C# rather than being a mix of C# and SQL strings.
  + **Abstraction:** You can switch the underlying database (e.g., from SQL Server to PostgreSQL) with minimal changes to your application code.

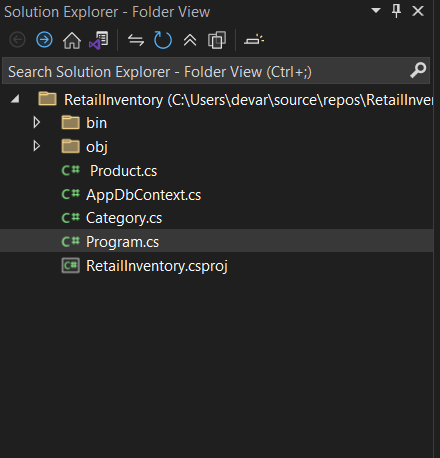
2. Create a .NET Console App:



3. Install EF Core Packages:

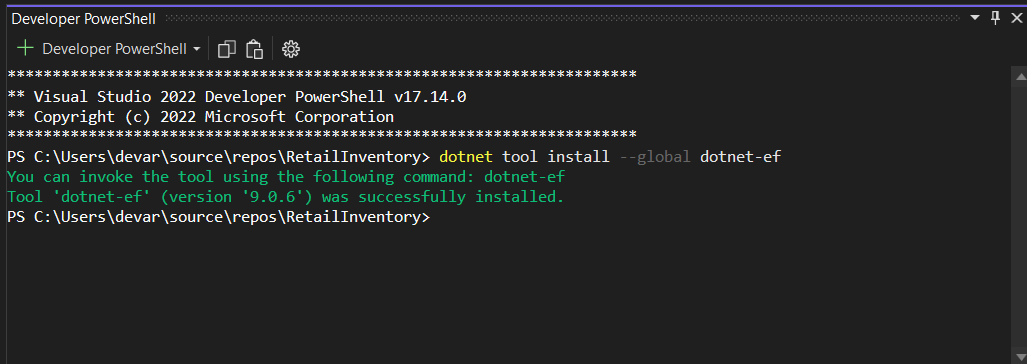


Lab 2: Setting Up the Database Context for a Retail Store:-

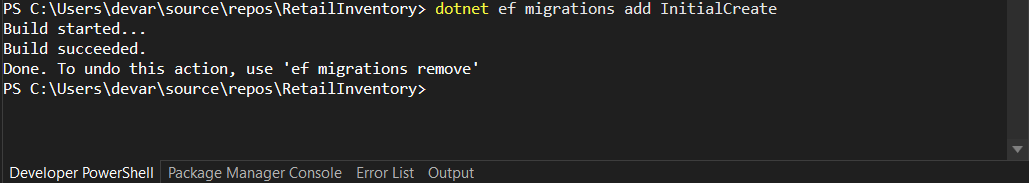


Lab 3: Using EF Core CLI to Create and Apply Migrations:-

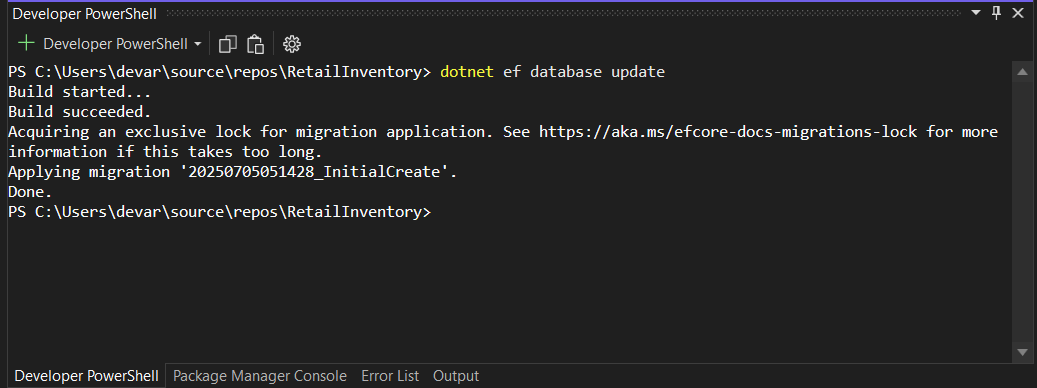
1. Install EF Core CLI (if not already):



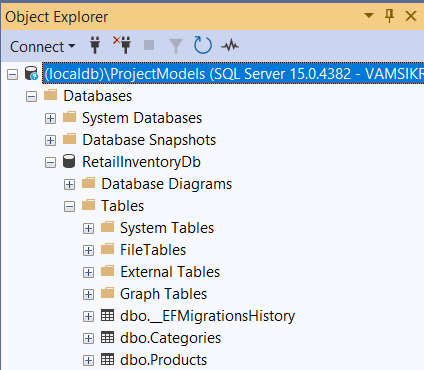
2. Create Initial Migration:-



3. Apply Migration to Create Database:-



4. Verify in SQL Server:-

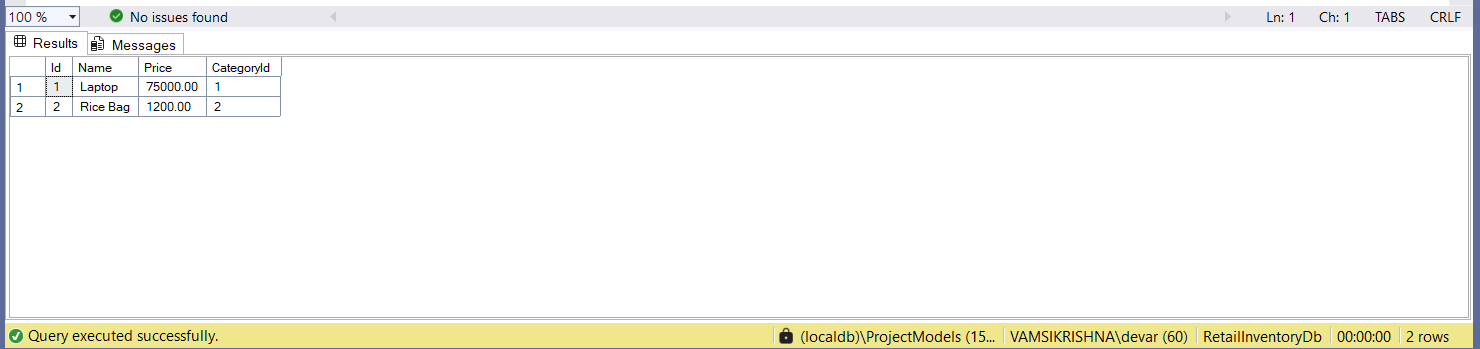


Lab 4: Inserting Initial Data into the Database:-

Verify in SQL Server:-

A screenshot of a computer

AI-generated content may be incorrect.



Lab 5: Retrieving Data from the Database:-

**1. Retrieve All Products:-**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

var products = await context.Products.ToListAsync();

foreach (var p in products)

{

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

}

}

}

A screenshot of a computer

AI-generated content may be incorrect.

**2. Find by ID:-**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using var context = new AppDbContext();

int idToFind = 1;

var product = await context.Products.FindAsync(idToFind);

if (product != null)

{

Console.WriteLine($"Found: {product.Name} - ₹{product.Price}");

}

else

{

Console.WriteLine("Product not found.");

}

OUTPUT:-

A screenshot of a computer

AI-generated content may be incorrect.

**3. FirstOrDefault with Condition:-**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using var context = new AppDbContext();

var expensiveProduct = await context.Products.FirstOrDefaultAsync(p => p.Price >5000);

if (expensiveProduct != null)

{

Console.WriteLine($"Expensive: {expensiveProduct.Name} - {expensiveProduct.Price}");

}

else

{

Console.WriteLine("No expensive product found.");

}

OUTPUT:-

**A black screen with white text

AI-generated content may be incorrect.**