**Name-Masheera Afrin.**

**Superset Id- 6362071**

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

1. **What is ORM?**

**ORM (Object-Relational Mapping)** is a technique that allows developers to interact with a database using object-oriented programming principles. It maps **C# classes** to **SQL database tables**, making it easier to perform CRUD operations (Create, Read, Update, Delete) without writing raw SQL queries.

**Benefits:**

**Productivity**: Write less code — no need for manual SQL.

**Maintainability**: Centralized logic using C# classes.

**Abstraction**: Focus on business logic, not SQL syntax.

2.**EF Core vs EF FrameWorK**

| **Feature** | **EF Core** | **EF Framework (EF6)** |
| --- | --- | --- |
| Platform | Cross-platform (.NET Core) | Windows-only (.NET Framework) |
| Performance | Lightweight & fast | Heavier, slower in large apps |
| LINQ & Async | Fully supported | Limited support |
| Flexibility | Supports modern practices | More mature but less flexible |
| Open-source | Yes | Yes |

**3.EF Core 8.0 Features**

**JSON Column Mapping**: Store and query JSON documents directly in database columns.

**Compiled Models**: Boosts performance by pre-compiling the entity model during build time.

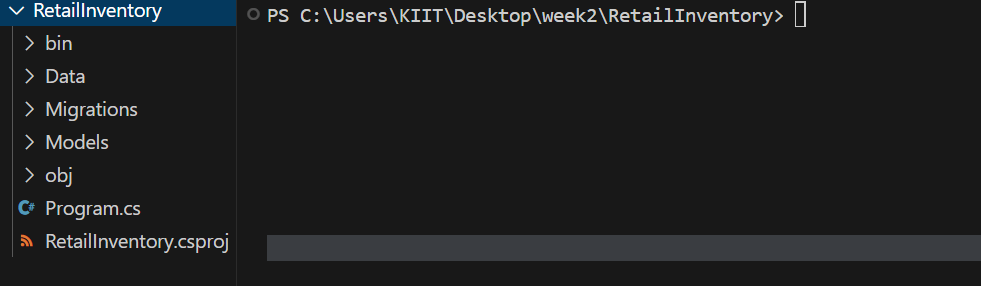
**Interceptors**: Hook into database operations (e.g., logging, validation).

**Improved Bulk Operations**: Efficiently insert or update many rows with fewer DB calls.

1. **Create a .NET Console App**

dotnet new console -n RetailInventory

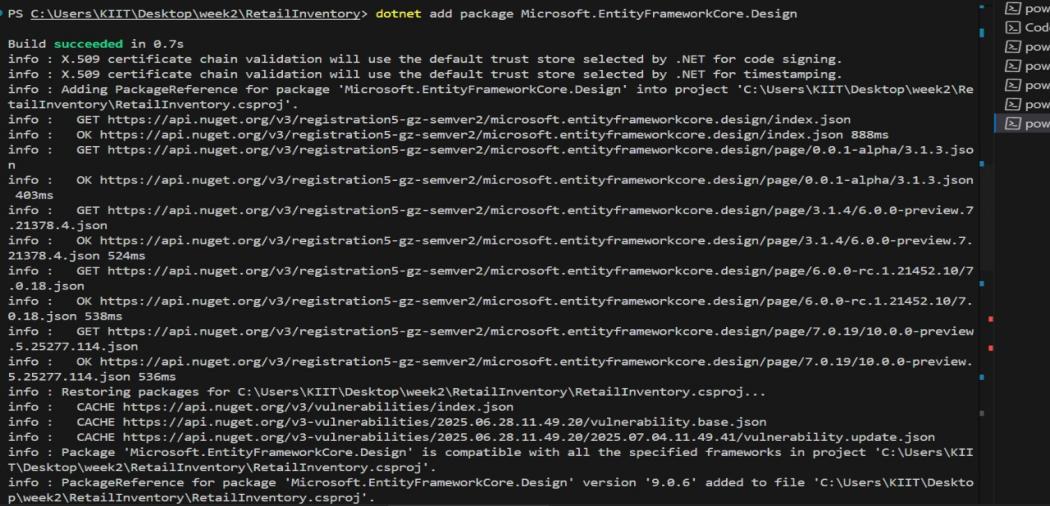
cd RetailInventory

**OUTPUT**

1. **Install EF Core Packages:**

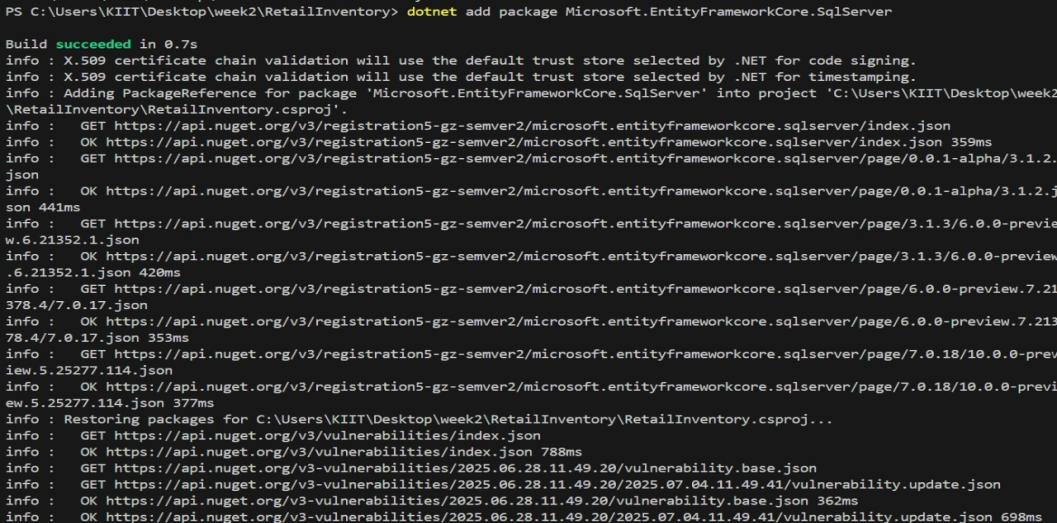
dotnet add package Microsoft.EntityFrameCore.Design

**OUTPUT:**

****

Dotnet add package Microsoft.EntityFrameworkCore.SqlServer

**OUTPUT:**

****

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

**Step1:**

**Models/category.cs**

using System.Collections.Generic;

namespace RetailInventory.Models

{

public class Category {

public int Id { get; set; }

public string Name { get; set; } = string.Empty;

public List<Product> Products { get; set; } = new();

}

}

**Models/Product.cs**

namespace RetailInventory.Models

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; } = string.Empty;

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; } = null!;

}

}

**Step2:**

**AppContextDb.cs**

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory.Data

{

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilderoptionsBuilder)

{

optionsBuilder.UseSqlServer("Server=localhost;Database=RetailStoreDB;Trusted\_Connection=True;TrustServerCertificate=True;");

}

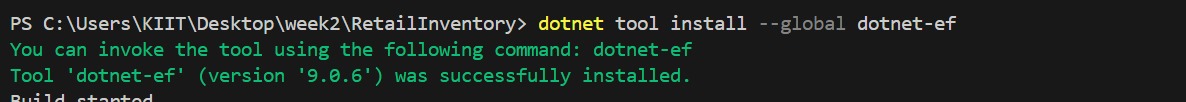
}

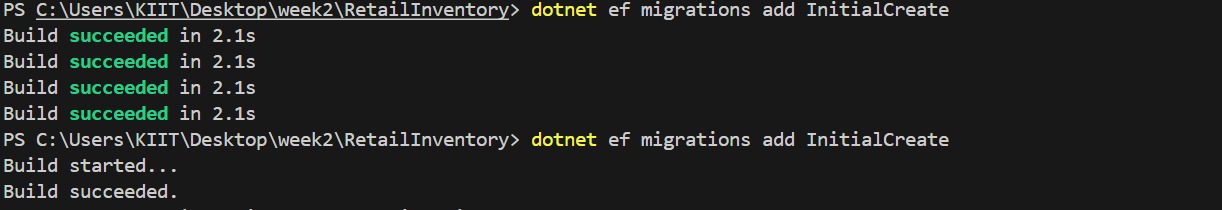
}

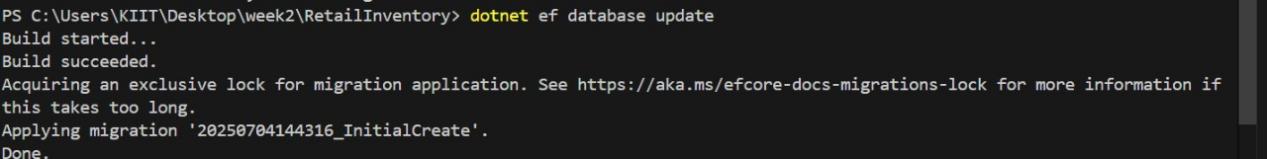
**Lab 3: Using EF Core CLI to Create and Apply Migrations**

**Steps:**

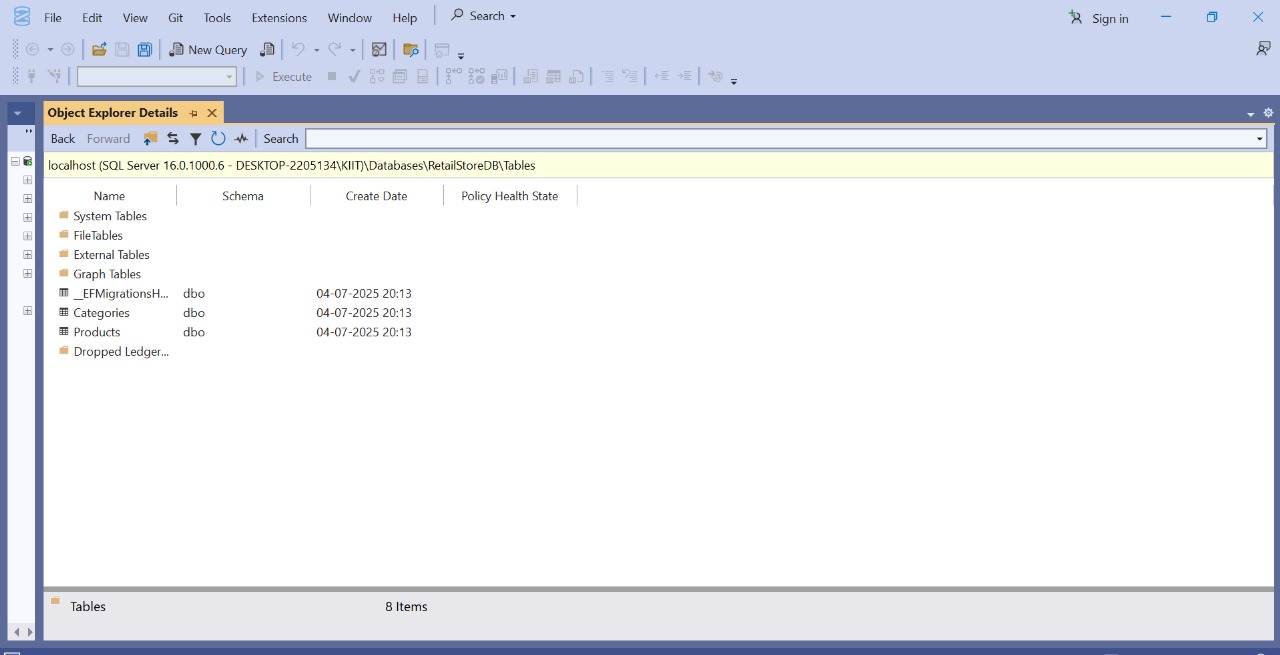
**1.OUTPUT:**

****

**2.OUTPUT:****3.OUTPUT:**



4.**OUTPUT:**



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

**Program.cs**

using System;

using System.Threading.Tasks;

using RetailInventory.Data;

using RetailInventory.Models;

namespace RetailInventory

{

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();

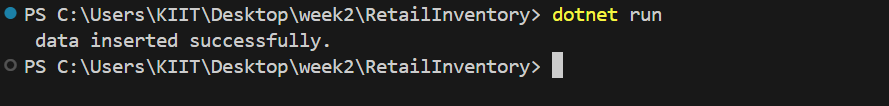
Console.WriteLine(" data inserted successfully.");

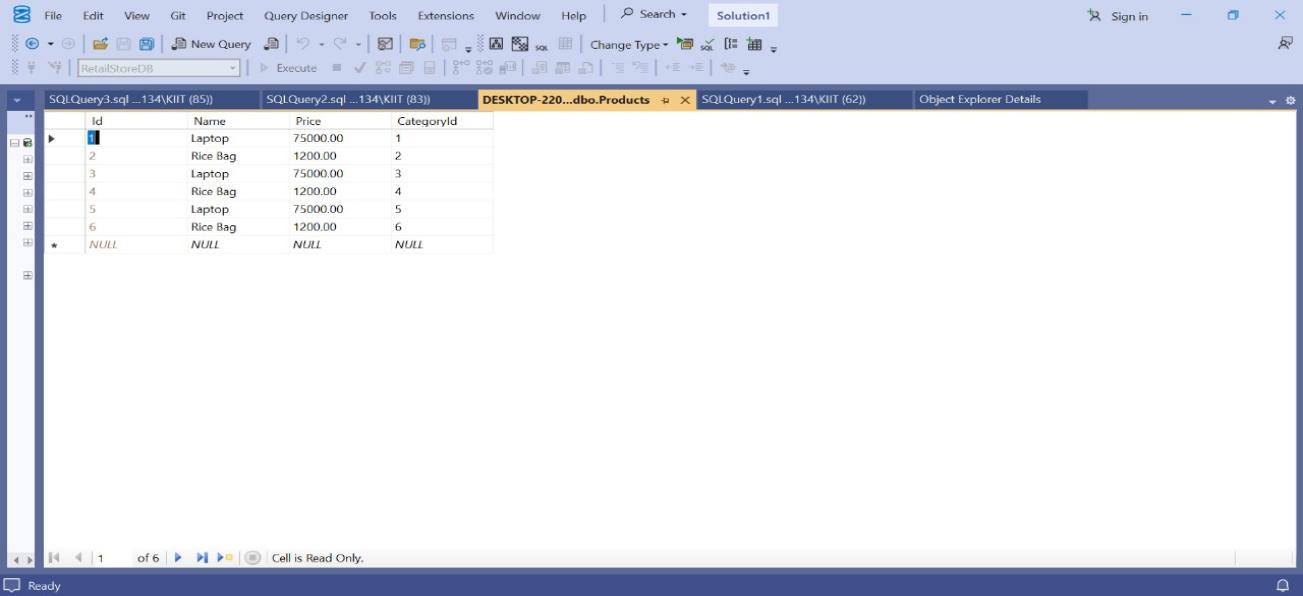
}

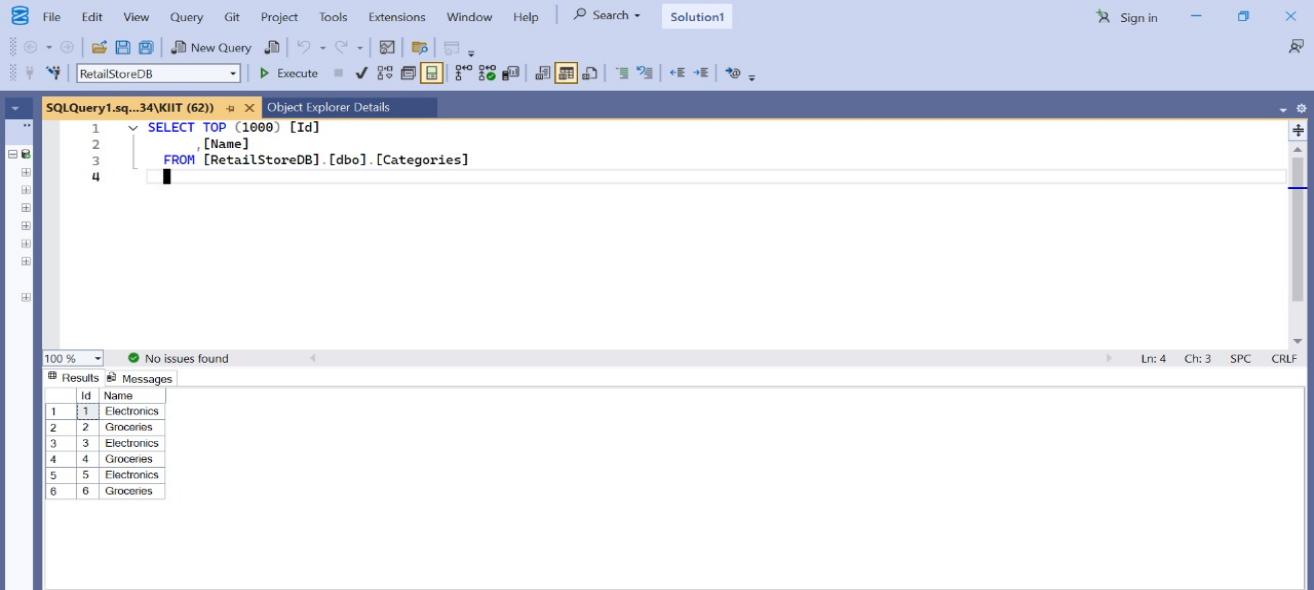
}

}

**OUTPUT:**







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

Steps:

**1.Retrieve All Products**

**Program.cs**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Data;

using RetailInventory.Models;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

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

Console.WriteLine("All Products:");

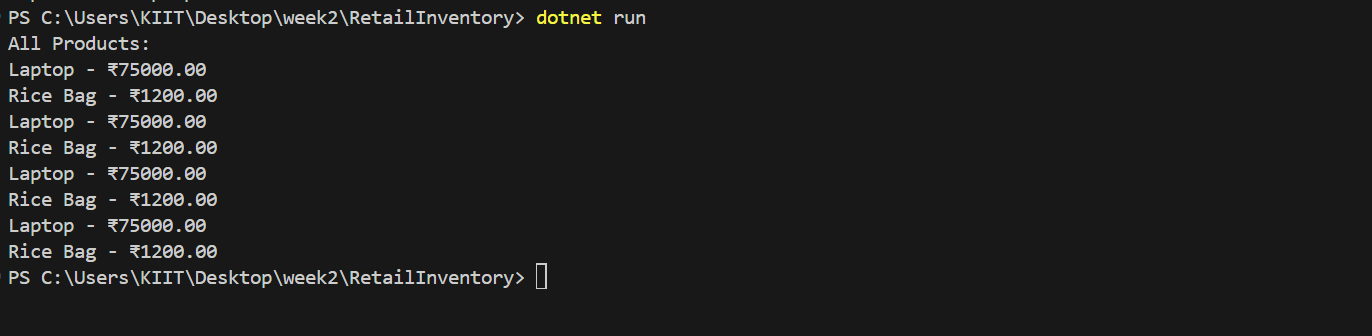
foreach (var p in products)

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

}

}

**OUTPUT:**



1. **Find by ID**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Data;

using RetailInventory.Models;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

var product = await context.Products.FindAsync(1);

Console.WriteLine($"\nFound: {product?.Name}");

}

}

**OUTPUT:**



**3.FirstOrDefault with Condition**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Data;

using RetailInventory.Models;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

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

Console.WriteLine($"\nExpensive: {expensive?.Name}");

}

}

**OUTPUT:**