**EF Core 8.0 Guided Hands-On Exercises**

Lab 1: Understanding ORM with a Retail Inventory System

What is ORM?

-> ORM (Object-Relational Mapping) is a technique that maps C# classes to database tables. It allows developers to interact with databases using C# objects instead of writing raw SQL.

Console App creation

#dotnet new console -n RetailInventory

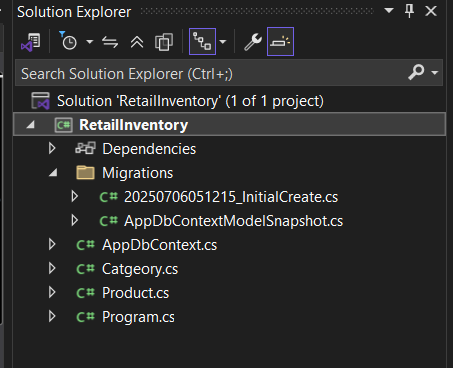
cd RetailInventory

EF Core Packages Installation

#dotnet add package Microsoft.EntityFrameworkCore.SqlServer

#dotnet add package Microsoft.EntityFrameworkCore.Design

**OUTPUT**



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

**Category.cs**

using System.Collections.Generic;

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)

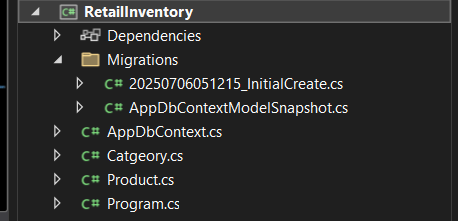
{

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

}

}

OUTPUT



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

Installed EF Core CLI Tool

#dotnet tool install --global dotnet-ef

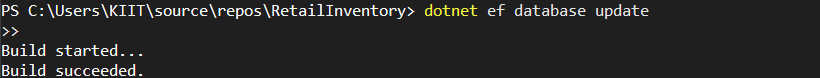
Created Initial Migration:

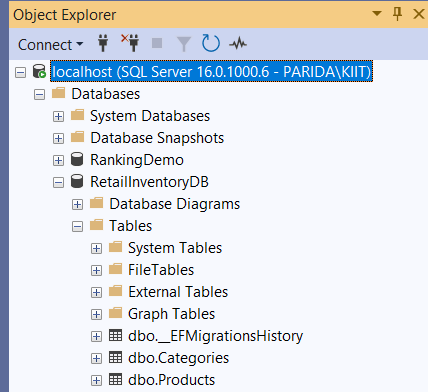
#dotnet ef migrations add InitialCreate

Applied Migration to Create the Database:

#dotnet ef database update

OUTPUT





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

Modified *Program.cs* to Insert Data

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

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.");

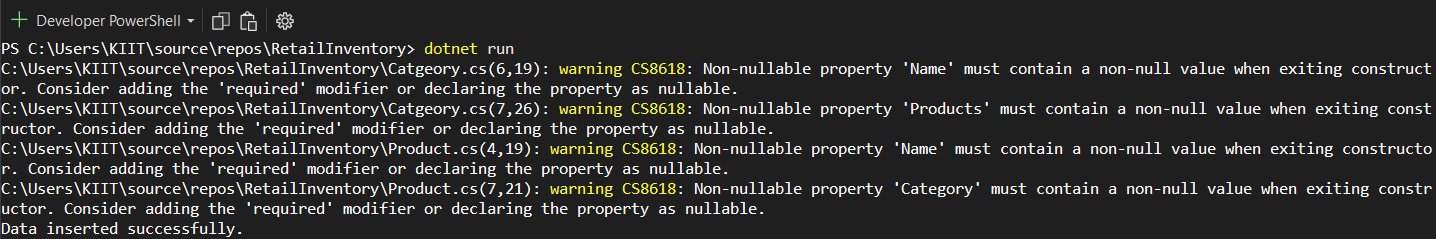
}

}

Ran the Console App

#dotnet run

OUTPUT





(N.B. I have done lab 6 and lab 7 so it’s showing only 1 product)

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

Modified *Program.cs* to Insert Data

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

// 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}");

// 2. Find by ID

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

Console.WriteLine($"\nProduct with ID 1: {product?.Name}");

// 3. FirstOrDefault with condition

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

Console.WriteLine($"\nFirst Expensive Product (> ₹50,000): {expensive?.Name}");

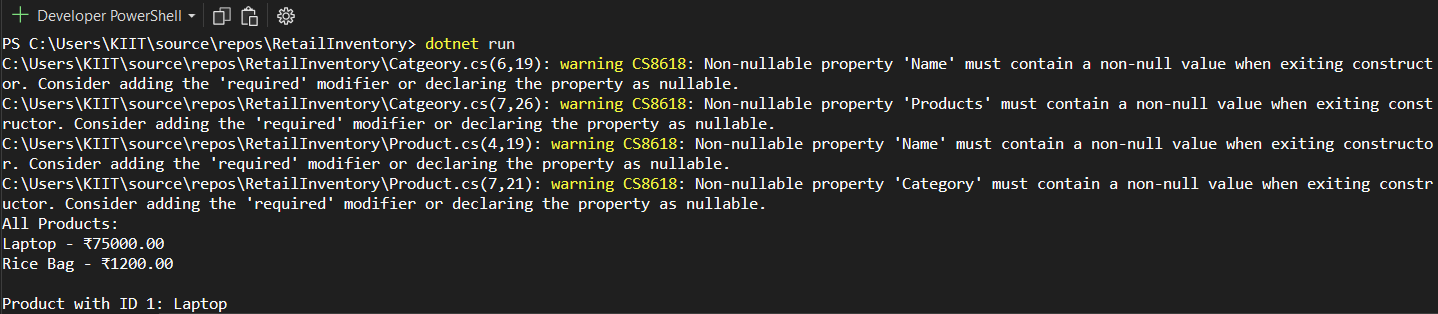
}

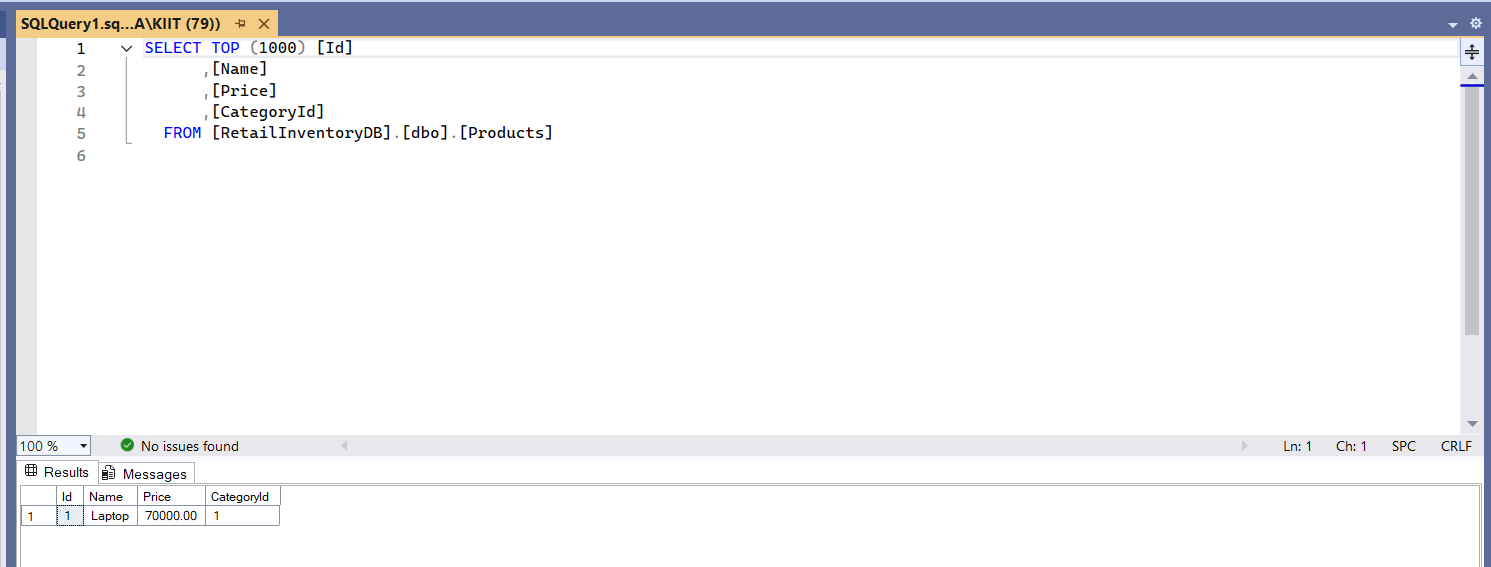
}

Ran the Console App

#dotnet run

OUPUT





(N.B. I have done lab 6 and lab 7 so it’s showing only 1 product)

**Lab 6: Updating and Deleting Records**

Modified *Program.cs* to Update & Delete Records

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

// --- Update Product Price ---

var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");

if (product != null)

{

product.Price = 70000;

await context.SaveChangesAsync();

Console.WriteLine($"Updated {product.Name}'s price to ₹{product.Price}");

}

// --- Delete Discontinued Product ---

var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Rice Bag");

if (toDelete != null)

{

context.Products.Remove(toDelete);

await context.SaveChangesAsync();

Console.WriteLine($"Deleted product: {toDelete.Name}");

}

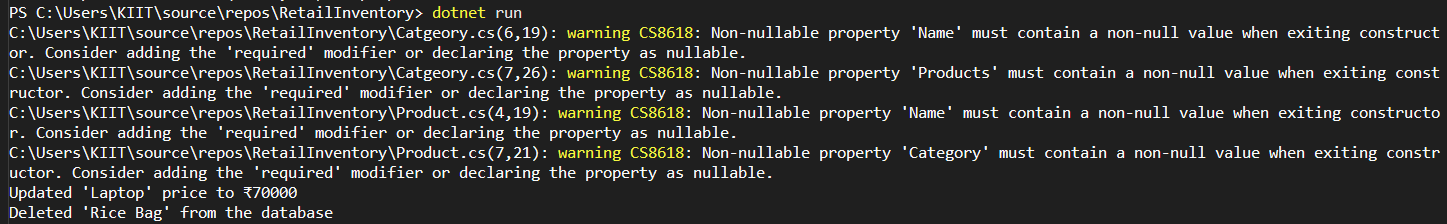
}

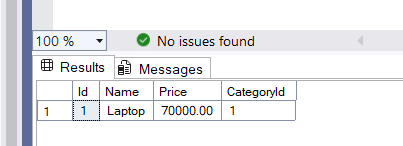
}

Ran the Console App

#dotnet run

OUTPUT





(Rice Bag is gone)

**Lab 7: Writing Queries with LINQ**

Modified *Program.cs* to Perform LINQ Queries

using System;

using System.Threading.Tasks;

using System.Linq;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

// --- Filter and Sort Products ---

var filtered = await context.Products

.Where(p => p.Price > 1000)

.OrderByDescending(p => p.Price)

.ToListAsync();

Console.WriteLine("Filtered & Sorted Products (Price > ₹1000):");

foreach (var p in filtered)

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

// --- Project into DTO (Anonymous Object) ---

var productDTOs = await context.Products

.Select(p => new { p.Name, p.Price })

.ToListAsync();

Console.WriteLine("\nProduct DTOs:");

foreach (var dto in productDTOs)

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

}

}

Ran the Console App

#dotnet run

OUTPUT

