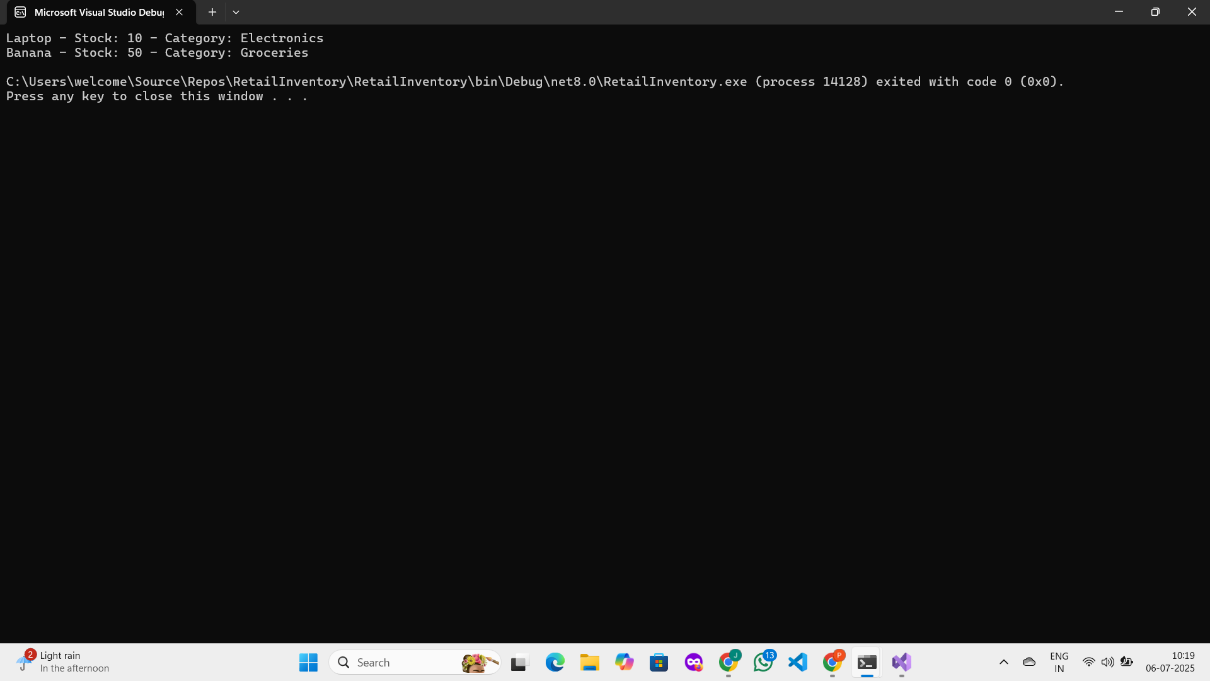
**Lab 1: Understanding ORM with a Retail Inventory System**

**Objective**

To understand what Object-Relational Mapping (ORM) is and how Entity Framework Core (EF Core) helps bridge the gap between C# objects and SQL Server relational tables.

public class Product  
{  
 public int ProductId { get; set; }  
 public string Name { get; set; } = string.Empty;  
 public int Stock { get; set; }  
 public Category Category { get; set; } = null!;  
}  
  
public class Category  
{  
 public int CategoryId { get; set; }  
 public string Name { get; set; } = string.Empty;  
 public List<Product> Products { get; set; } = new();  
}  
  
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=(localdb)\MSSQLLocalDB;Database=RetailDb;Trusted\_Connection=True;");  
 }  
}  
  
class Program  
{  
 static void Main(string[] args)  
 {  
 using var context = new AppDbContext();  
  
 if (!context.Categories.Any())  
 {  
 var electronics = new Category { Name = "Electronics" };  
 var groceries = new Category { Name = "Groceries" };  
  
 context.Categories.AddRange(electronics, groceries);  
  
 context.Products.AddRange(  
 new Product { Name = "Laptop", Stock = 10, Category = electronics },  
 new Product { Name = "Banana", Stock = 50, Category = groceries }  
 );  
  
 context.SaveChanges();  
 }  
  
 foreach (var product in context.Products.Include(p => p.Category))  
 {  
 Console.WriteLine($"{product.Name} - Stock: {product.Stock} - Category: {product.Category?.Name}");  
 }  
 }  
}

**Output**



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

**Scenario**: The retail store wants to store product and category data in SQL Server.

**Objective:** Configure DbContext and connect to SQL Server.

public class Category {

public int Id { get; set; }

public string Name { get; set; }

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

}

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; }

}

public class AppDbContext : DbContext {

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {

optionsBuilder.UseSqlServer("Your\_Connection\_String\_Here");

}

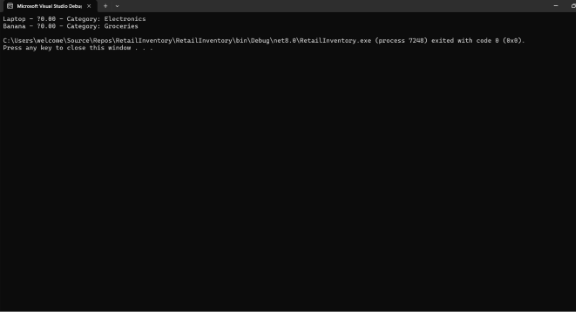
}

dotnet ef migrations add InitialCreate

dotnet ef database update

dotnet ef migrations add AddPriceColumn

dotnet ef database update

OUTPUT :  


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

**Scenario: The retail store's database needs to be created based on the models you've defined. You’ll use EF Core CLI to generate and apply migrations.**

**Objective: Learn how to use EF Core CLI to manage database schema changes.**

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

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; }

}

using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext {

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

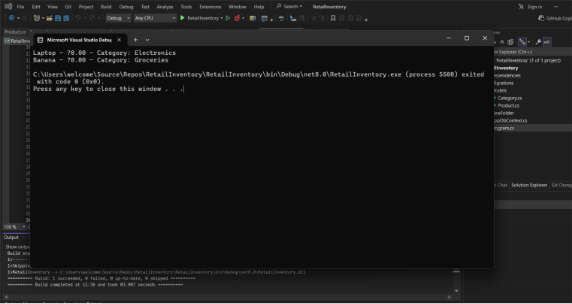
public DbSet<Category> Categories { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {

optionsBuilder.UseSqlServer("Your\_Connection\_String\_Here");

}

}

**OUTPUT :**  


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

**Scenario: The store manager wants to add initial product categories and products to the system.**

**Objective: Use EF Core to insert records using AddAsync and SaveChangesAsync.**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

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();

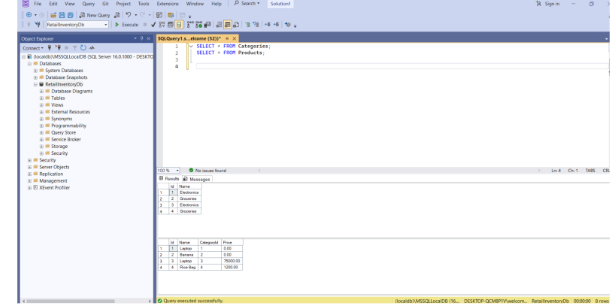
Console.WriteLine("Initial data inserted successfully!");

}

}

}

**OUTPUT :**



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

**Scenario: The store wants to display product details on the dashboard.**

**Objective: Use Find, FirstOrDefault, and ToListAsync to retrieve data.**

using System;  
using System.Threading.Tasks;  
using Microsoft.EntityFrameworkCore;  
  
namespace RetailInventory  
{  
 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}");  
  
 Console.WriteLine();  
  
 var product = await context.Products.FindAsync(1);  
 Console.WriteLine($"Product with ID 1: {product?.Name}");  
  
 Console.WriteLine();  
  
 var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);  
 Console.WriteLine($"First Expensive Product: {expensive?.Name}");  
 }  
 }

**OUTPUT :**

