**WEEK 3**

**ENTITY FRAMEWORK CORE 8.0**

**EXERCISE 1: UNDERSTANDING ORM WITH A RETAIL INVENTORY SYSTEM**

Understand what ORM is and how EF Core helps bridge the gap between C# objects and relational tables.

Category.cs

using System.Collections.Generic;

public class Category

{

    public int CategoryId { get; set; }

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

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

}

Product.cs

public class Product

{

    public int ProductId { get; set; }

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

    public int Stock { get; set; }

    public int CategoryId { get; set; }

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

}

AppDbContext.cs

using Microsoft.EntityFrameworkCore;

namespace RetailInventory;

public class AppDbContext : DbContext

{

    public DbSet<Product> Products => Set<Product>();

    public DbSet<Category> Categories => Set<Category>();

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

    {

        // You can replace this with your working server if LocalDB is not installed

        optionsBuilder.UseSqlServer(@"Server=(localdb)\MSSQLLocalDB;Database=RetailDb;Trusted\_Connection=True;");

    }

}

Program.cs

using System;

using System.Linq;

using Microsoft.EntityFrameworkCore;

using RetailInventory;

namespace RetailInventory;

public class Program

{

    public static void Main()

    {

        Console.WriteLine("EF Core App is Running...\n");

        using var context = new AppDbContext();

        // Add category

        if (!context.Categories.Any())

        {

            var category = new Category { CategoryName = "Electronics" };

            context.Categories.Add(category);

            context.SaveChanges();

            Console.WriteLine("Category 'Electronics' added.");

        }

        // Add product

        if (!context.Products.Any())

        {

            var product = new Product

            {

                Name = "Laptop",

                Stock = 10,

                CategoryId = context.Categories.First().CategoryId

            };

            context.Products.Add(product);

            context.SaveChanges();

            Console.WriteLine("Product 'Laptop' added.");

        }

        // Display data

        var products = context.Products.Include(p => p.Category).ToList();

        Console.WriteLine("\n--- Product List ---");

        foreach (var p in products)

        {

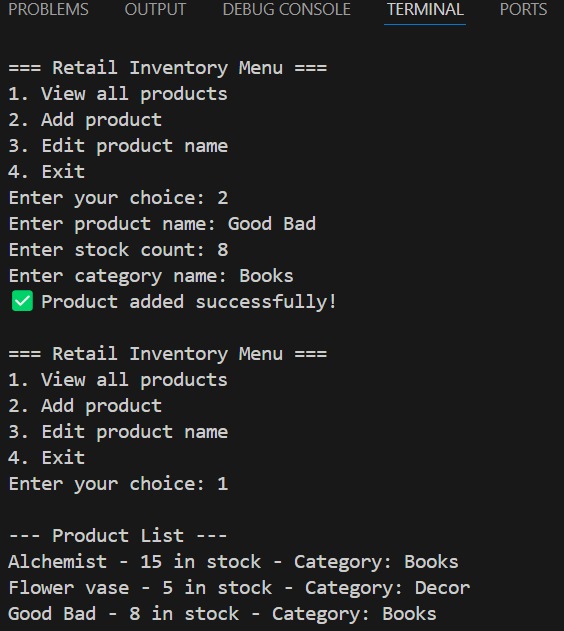
            Console.WriteLine($"{p.Name} - {p.Stock} in stock - Category: {p.Category.CategoryName}");

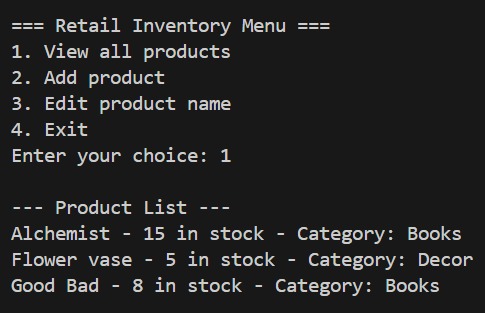
        }

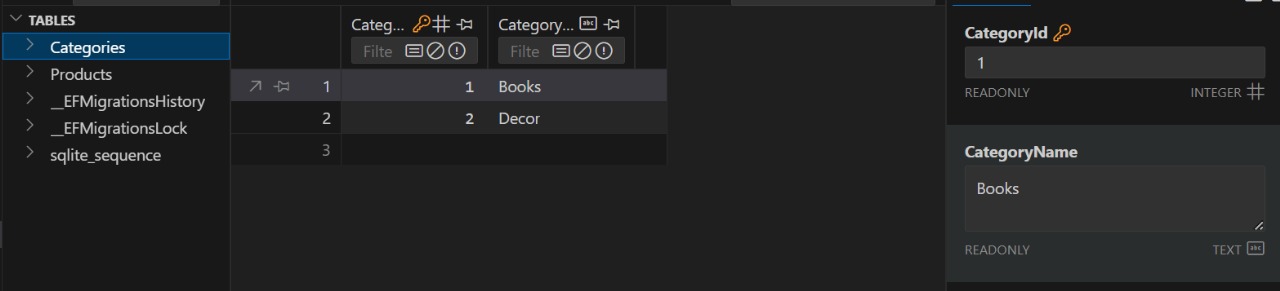
    }

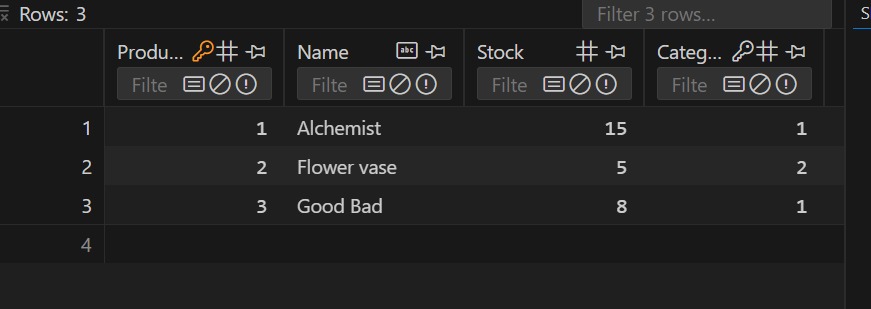
}

**OUTPUT:**









**EXERCISE 2: SETTING UP THE DATABASE CONTEXT FOR A RETAIL STORE**

Configure DbContext and connect to SQL Server.

AppDbContext.cs

using Microsoft.EntityFrameworkCore;

namespace RetailStoreSQL;

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\\SQLEXPRESS;Database=RetailStoreDb;Trusted\_Connection=True;TrustServerCertificate=True;"

);

    }

}

Category.cs

namespace RetailStoreSQL;

public class Category

{

    public int Id { get; set; }

    public required string Name { get; set; }

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

}

Product.cs

namespace RetailStoreSQL;

public class Product

{

    public int Id { get; set; }

    public required string Name { get; set; }

    public decimal Price { get; set; }

    public int CategoryId { get; set; }

    public required Category Category { get; set; }

}

Program.cs

using System;

using System.Linq;

using Microsoft.EntityFrameworkCore;

using RetailStoreSQL;

class Program

{

    static void Main()

    {

        using var context = new AppDbContext();

        context.Database.EnsureCreated();

        if (!context.Categories.Any())

        {

            var books = new Category { Name = "Books" };

            context.Categories.Add(books);

            var book = new Product

            {

                Name = "Learn SQL in 1 Day",

                Price = 499,

                Category = books

            };

            context.Products.Add(book);

            context.SaveChanges();

        }

        var products = context.Products.Include(p => p.Category).ToList();

        foreach (var p in products)

        {

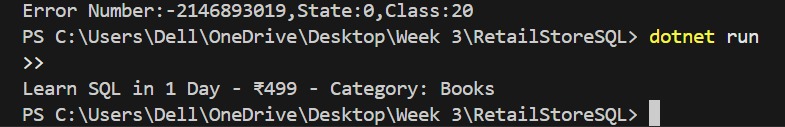
            Console.WriteLine($"{p.Name} - ₹{p.Price} - Category: {p.Category.Name}");

        }

    }

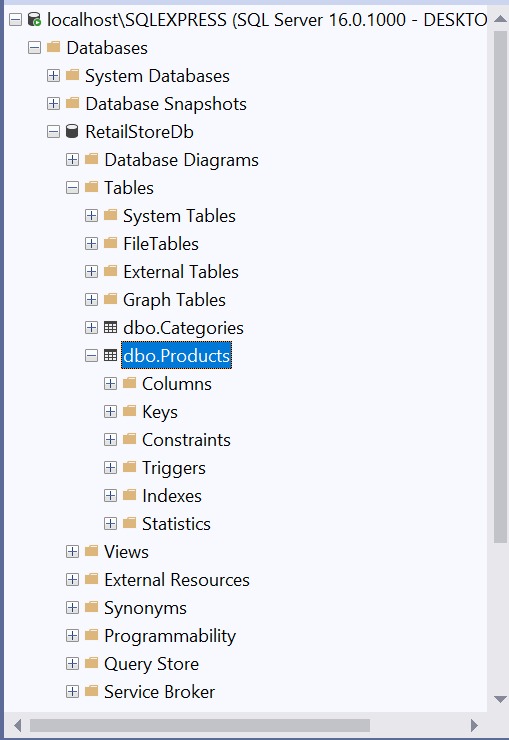
}

**OUTPUT:**



**EXERCISE 3: USING EF CORE CLI TO CREATE AND APPLY MIGRATIONS**

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



**EXERCISE 4: INSERTING INITIAL DATA INTO THE DATABASE**

Use EF Core to insert records using AddAsync and SaveChangesAsync.

Program.cs

using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RetailStoreSQL;

class Program

{

    public static async Task Main(string[] args)

    {

        using var context = new AppDbContext();

        // Optional: Apply migrations automatically (for testing only)

        await context.Database.MigrateAsync();

        if (!context.Categories.Any())

        {

            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!");

        }

        else

        {

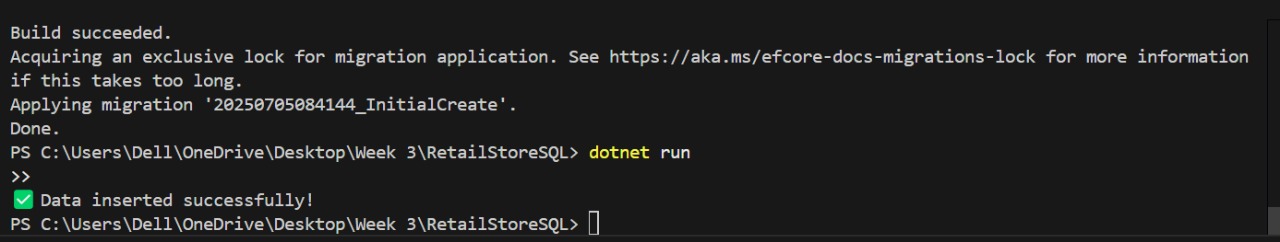
            Console.WriteLine("ℹ️ Data already exists in the database.");

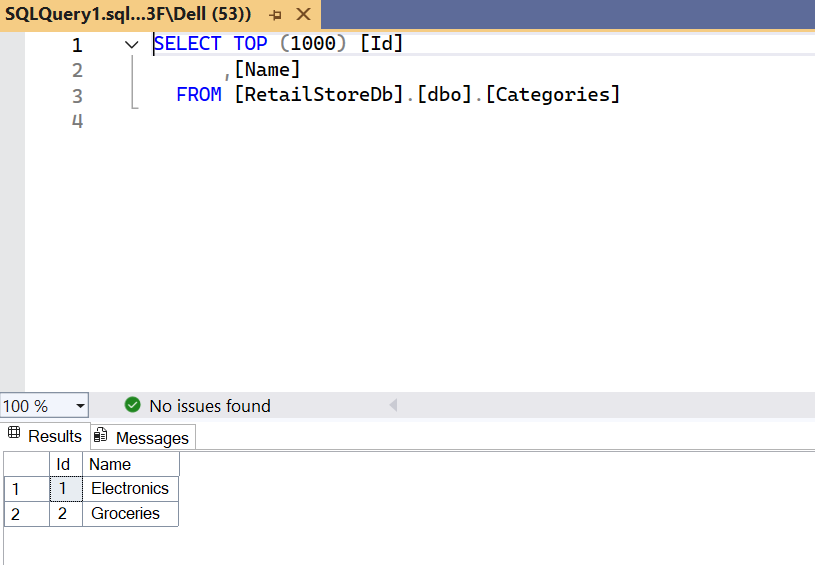
        }

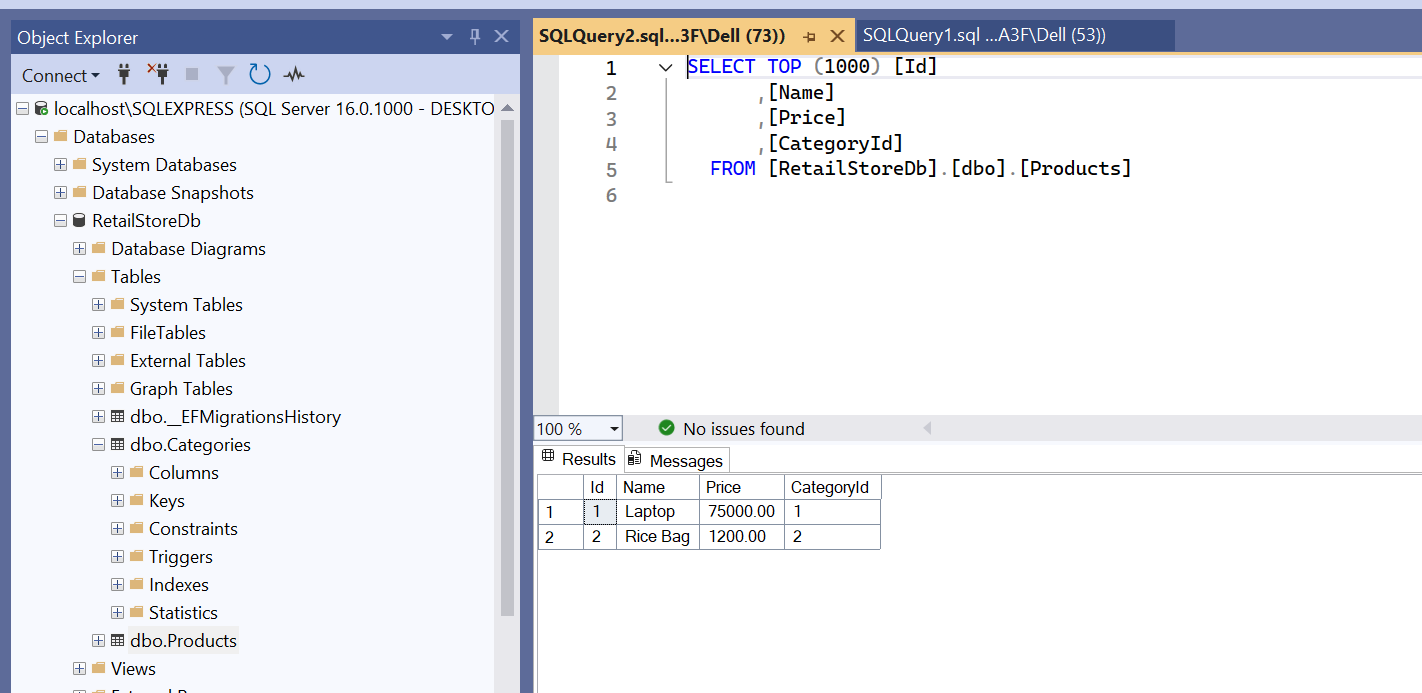
    }

}

Output:







**EXERCISE 5: RETRIEVING DATA FROM THE DATABASE**

Use Find, FirstOrDefault, and ToListAsync to retrieve data.

Program.cs

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RetailStoreSQL;

class Program

{

    static async Task Main(string[] args)

    {

        using var context = new AppDbContext();

        // 1. Retrieve All Products

        var products = await context.Products

                                    .Include(p => p.Category) // Include category details

                                    .ToListAsync();

        Console.WriteLine("All Products:");

        foreach (var p in products)

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

        Console.WriteLine(); // blank line for spacing

        // 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($"Expensive Product (> ₹50000): {expensive?.Name}");

    }

}

**OUTPUT:**

