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 relational database using object-oriented programming languages like C#. Instead of writing raw SQL queries, you can use objects to perform CRUD operations.

How ORM maps C# classes to DB tables:

- A C# class (e.g., *Product*) becomes a database table.
- Class properties (e.g., *Name, Price*) become columns in that table.
- EF Core handles the conversion between your C# objects and the SQL data behind the scenes.

2. Benefits of Using ORM

- **Productivity**: Write less boilerplate code.
- **Maintainability**: Centralize logic in your models.
- **Abstraction**: Avoid complex SQL; use LINQ instead.
- **Portability**: Easily switch databases.

3. EF Core vs EF Framework

Feature	EF Core	EF 6 (EF Framework)
Cross-platform	Yes	No
Performance	Better	Slower
Modular Architecture	Yes	Monolithic
.NET Compatibility	.NET Core, .NET 5+	.NET Framework only
Development Focus	Active Development	Maintenance Only
Reverse Engineering (Scaffold)	Yes	Yes
Lazy Loading	Yes (from EF Core 2.1+)	Yes
Designer/Visual Tools	Limited	Better support

4. EF Core 8.0 New Features

● **JSON Column Mapping**: Store complex objects directly in a single column.
● **Compiled Models**: Speeds up startup performance for large databases.
● **Interceptors**: Hook into database calls for logging or validation.
● **Bulk Operations Improvements**: More efficient insert/update/delete.

5. Project Setup

Create a .NET Console App:

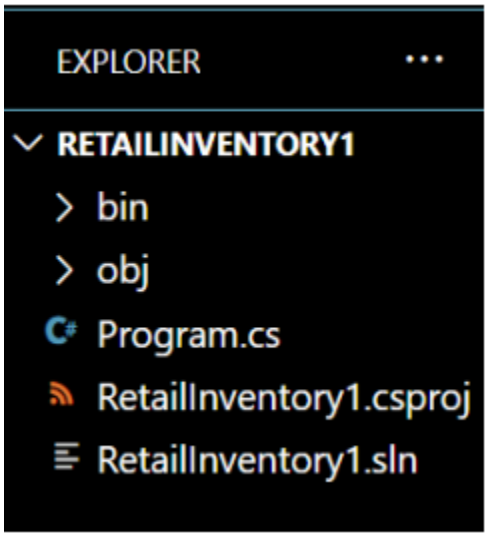
dotnet new console -n RetailInventory

Install EF Core Packages:

dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Design

This sets up EF Core in your project and prepares it to work with a SQL Server database.

After Setup:



Lab - 2

1. **AppDbConntextlab2.cs**

```
using Microsoft.EntityFrameworkCore;
using RetailInventory.Models;

namespace RetailInventory
{
   public class AppDbContext : DbContext
   {
      public DbSet<Product> Products { get; set; }
      public DbSet<Category> Categories { get; set; }

      protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
      {
         optionsBuilder.UseSqlite("Data Source=retail.db");
```

```
    }
  }
}
```

2. **Categorylab2.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();
    }
}
```

3. **Productlab2.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!;
    }
}
```

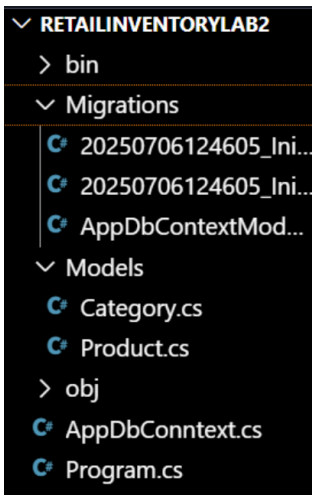
4. **Programlab2.cs**

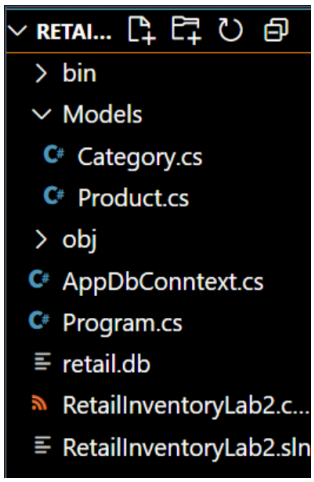
```
using System;
using System.Linq;
using RetailInventory.Models;

namespace RetailInventory
{
   class Program
   {
```

```
        static void Main(string[] args)
        {
            using var context = new AppDbContext();

            // Ensure database is created
            context.Database.EnsureCreated();

            // Add sample data if none exists
            if (!context.Categories.Any())
            {
                var electronics = new Category { Name = "Electronics" };
                var groceries = new Category { Name = "Groceries" };

                context.Categories.AddRange(electronics, groceries);

                context.Products.Add(new Product { Name = "Laptop", Price = 60000,
Category = electronics });
                context.Products.Add(new Product { Name = "Smartphone", Price = 30000,
Category = electronics });
                context.Products.Add(new Product { Name = "Rice", Price = 50, Category =
groceries });

                context.SaveChanges();
            }

            // Fetch and print all products with categories
            var products = context.Products
                .Select(p => new { p.Name, p.Price, CategoryName = p.Category.Name })
                .ToList();

            Console.WriteLine("Products in RetailInventory:");
            foreach (var p in products)
            {
                Console.WriteLine($"{p.Name} - ₹{p.Price} - Category: {p.CategoryName}");
            }
        }
    }
}
```

```
PS C:\Users\KIIT\RetailInventory> dotnet run
>>
Products in RetailInventory:
Laptop - ₹60000.0 - Category: Electronics
Smartphone - ₹30000.0 - Category: Electronics
Rice - ₹50.0 - Category: Groceries
```

Lab - 3






```
>> dotnet tool install --global dotnet-ef
>>
Tool 'dotnet-ef' is already installed.
PS C:\Users\KIIT\RetailInventoryLab2> dotnet ef migrations add InitialCreate
Build started...
Build succeeded.
```

```
PS C:\Users\KIIT\RetailInventoryLab2> dotnet ef database update
Build started...
Build succeeded.
```

Lab - 4

```
using Microsoft.EntityFrameworkCore;
using RetailInventory.Models;

namespace RetailInventory
{
    public class AppDbContext : DbContext
    {
```

```
        public DbSet<Product> Products { get; set; }
        public DbSet<Category> Categories { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlite("Data Source=retail.db");

        }
    }
}
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();
    }
}
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!;
    }
}
using System;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using RetailInventory.Models;

namespace RetailInventory
{
```

```
class Program
{
    static async Task Main(string[] args)
    {
        using var context = new AppDbContext();

        // Optional: Only insert data if empty
        if (await context.Categories.CountAsync() == 0)
        {
            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.");
        }
        else
        {
            Console.WriteLine("Data already exists. Skipping insertion.");
        }

        // Optional: Show products
        var products = context.Products.Include(p => p.Category);

        Console.WriteLine("\nProducts:");
        await foreach (var p in products.AsAsyncEnumerable())
        {
            Console.WriteLine($"{p.Name} - ₹{p.Price} - Category: {p.Category.Name}");
        }
    }
}
```

Lab 5

```
using Microsoft.EntityFrameworkCore;
using RetailInventory.Models;

namespace RetailInventory
{
    public class AppDbContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Category> Categories { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlite("Data Source=retail.db");
        }
    }
}
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();
    }
}
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!;
    }
```

```
}
using System;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using RetailInventory.Models;

namespace RetailInventory
{
    class Program
    {
        static async Task Main(string[] args)
        {
            using var context = new AppDbContext();

            // Seed data if none exists
            if (!await context.Categories.AnyAsync())
            {
                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("Seeded initial data.");
            }

            // Now retrieve data
            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($"Found product with ID=1: {product?.Name ?? "Not
found"}");

        Console.WriteLine();

        var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price >
50000);
        Console.WriteLine($"First expensive product (>₹50000): {expensive?.Name ??
"None found"}");
      }
   }
}
```