EF Core 8.0 Guided Hands-On Exercises

Lab 1: Understanding ORM with a Retail Inventory System

Scenario:

You’re building an inventory management system for a retail store. The store wants to

track products, categories, and stock levels in a SQL Server database.

Ans) Code:-

1. Create a .NET Console App:

dotnet new console -n RetailInventory

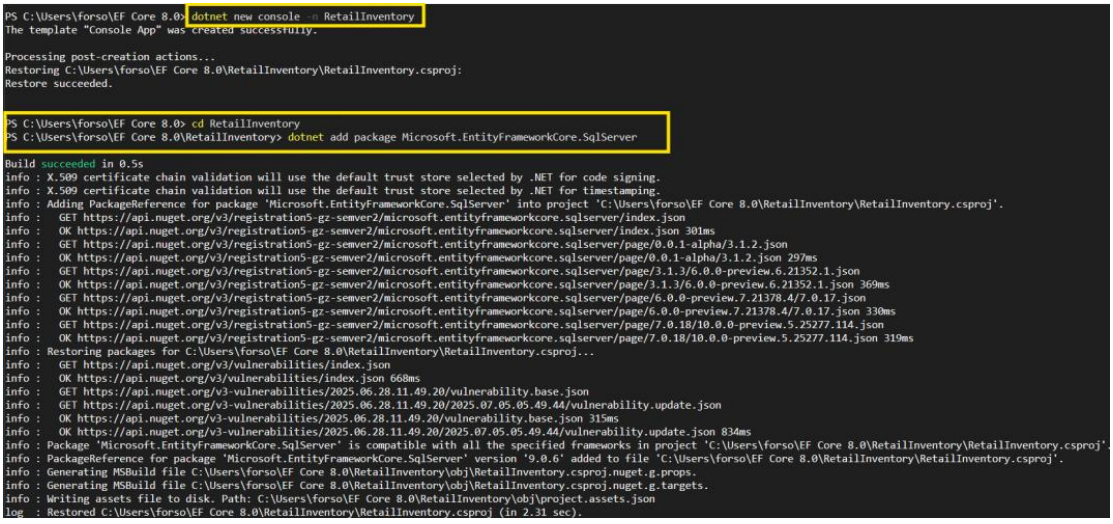
cd RetailInventory

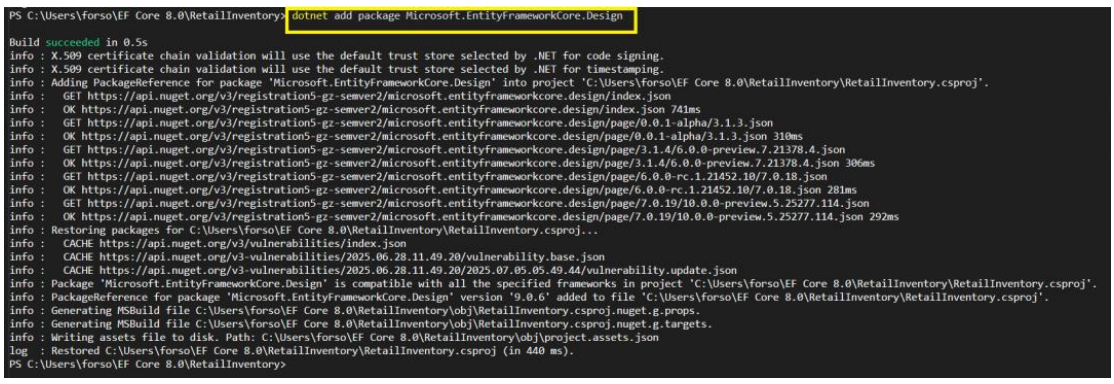
2. Install EF Core Packages:

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Design

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.

Ans) Code:-

Models/Product.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace RetailInventory.Models
{
    public class Product
    {
        public int Id { get; set; }

        [Required]
        [MaxLength(200)]
        public string Name { get; set; } = string.Empty;

        [Column(TypeName = "decimal(18,2)")]
        public decimal Price { get; set; }

        public int CategoryId { get; set; }
        public Category Category { get; set; } = null!;
    }
}
```

Models/Category.cs

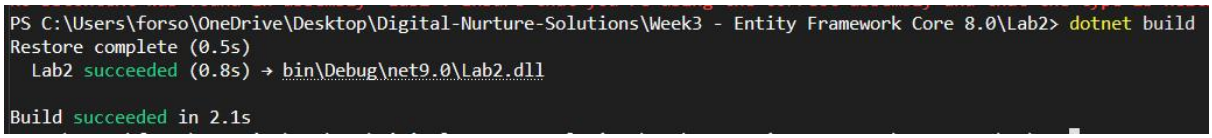
```
using System.ComponentModel.DataAnnotations;

namespace RetailInventory.Models
{
    public class Category
    {
        public int Id { get; set; }

        [Required]
        [MaxLength(100)]
        public string Name { get; set; } = string.Empty;

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

Data/AppDbContext.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(DbContextOptionsBuilder
optionsBuilder)
        {

optionsBuilder.UseSqlServer("Server=.;Database=RetailInventoryDB;Trusted_C
onnection=true;TrustServerCertificate=true;");
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {

            modelBuilder.Entity<Product>()
                .HasOne(p => p.Category)
                .WithMany(c => c.Products)
                .HasForeignKey(p => p.CategoryId);
        }
    }
}
```

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.

Ans) Code:-

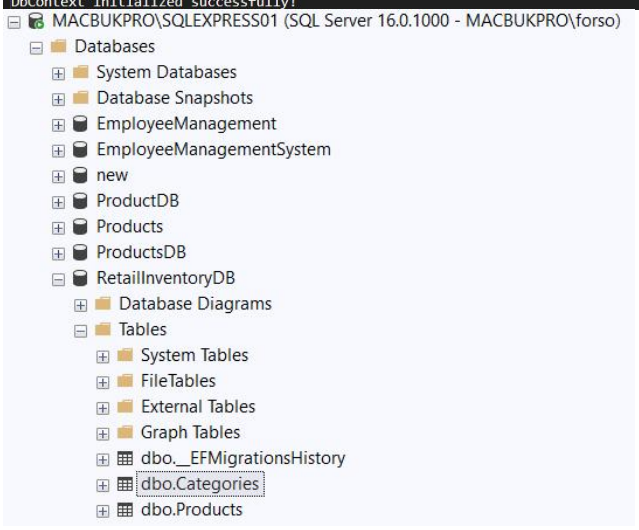
Program.cs

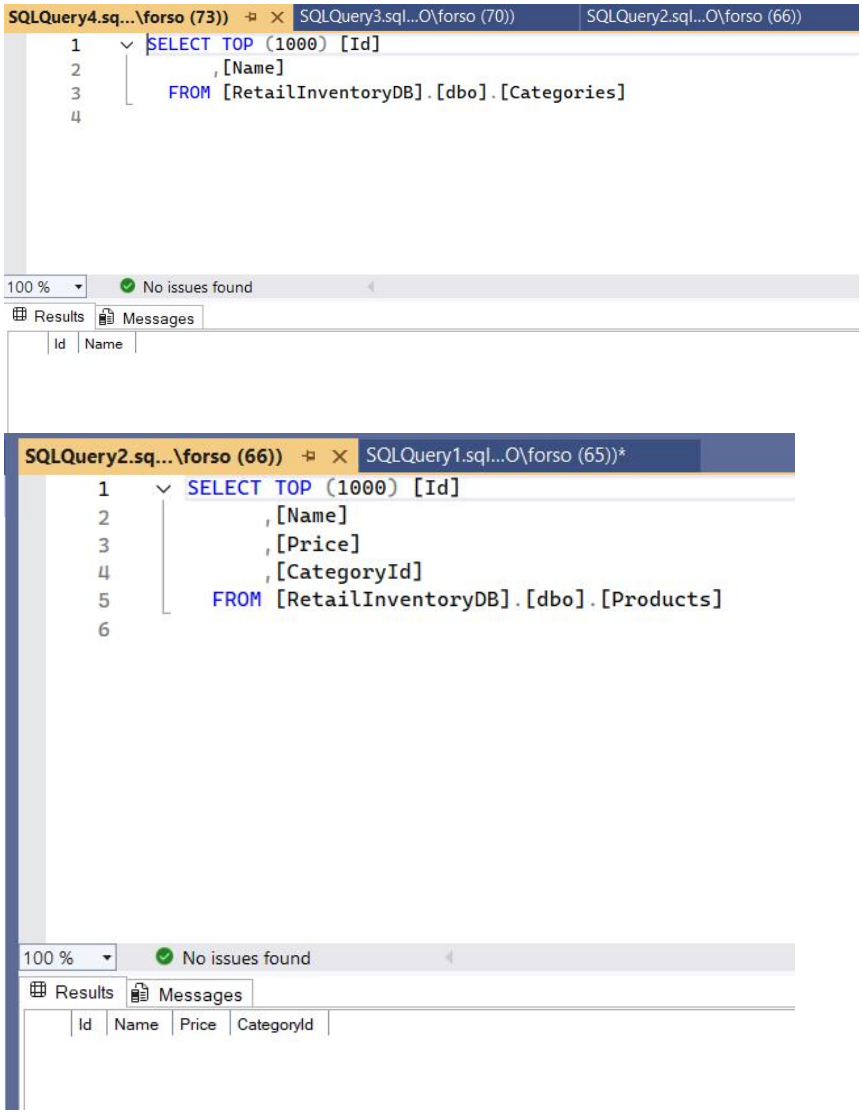
```
using Lab2.Data;
using Lab2.Models;
using Microsoft.EntityFrameworkCore;

Console.WriteLine("=== Retail Inventory System ===\n");

try
{
    using var context = new AppDbContext();
    await context.Database.EnsureCreatedAsync();
    if (!await context.Categories.AnyAsync())
    {
        Console.WriteLine("Inserting initial data...\n");

        var electronics = new Category { Name = "Electronics" };
        var groceries = new Category { Name = "Groceries" };
        var clothing = new Category { Name = "Clothing" };

        await context.Categories.AddRangeAsync(electronics, groceries,
clothing);

        var products = new List<Product>
        {
```

```
            new Product { Name = "Laptop", Price = 75000, Category =
electronics },
            new Product { Name = "Smartphone", Price = 45000, Category =
electronics },
            new Product { Name = "Rice Bag (25kg)", Price = 1200, Category
= groceries },
            new Product { Name = "Cooking Oil (1L)", Price = 180, Category
= groceries },
            new Product { Name = "T-Shirt", Price = 899, Category =
clothing },
            new Product { Name = "Jeans", Price = 2499, Category =
clothing }
        };

        await context.Products.AddRangeAsync(products);
        int recordsAffected = await context.SaveChangesAsync();

        Console.WriteLine($"☑ Successfully inserted {recordsAffected}
records!\n");
    }
    else
    {
        Console.WriteLine("⚠  Data already exists. Skipping
insertion.\n");
    }
}
catch (Exception ex)
{
    Console.WriteLine($"❌ Error: {ex.Message}");
}
```

Output:-

```
PS C:\Users\forso\OneDrive\Desktop\Digital-Nurture-Solutions\Week3 - Entity Framework Core 8.0\Lab2> dotnet build
Restore complete (0.3s)
  Lab2 succeeded (0.2s) → bin\Debug\net9.0\Lab2.dll

Build succeeded in 1.0s
PS C:\Users\forso\OneDrive\Desktop\Digital-Nurture-Solutions\Week3 - Entity Framework Core 8.0\Lab2> dotnet ef migrations add InitialCreate
Build started...
Build succeeded.
The name 'InitialCreate' is used by an existing migration.
PS C:\Users\forso\OneDrive\Desktop\Digital-Nurture-Solutions\Week3 - Entity Framework Core 8.0\Lab2> dotnet ef database update
Build started...
Build succeeded.
Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.
No migrations were applied. The database is already up to date.
Done.
PS C:\Users\forso\OneDrive\Desktop\Digital-Nurture-Solutions\Week3 - Entity Framework Core 8.0\Lab2> dotnet run
>>
DbContext initialized successfully!
```





Lab 4: Inserting Initial Data into the Database

Scenario:

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

Ans) Code:-

Program.cs

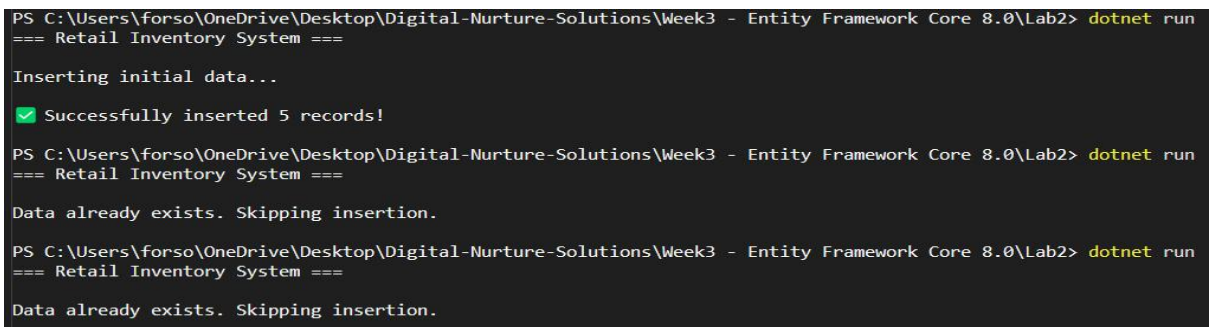
```
using Lab2.Data;
using Lab2.Models;
using Microsoft.EntityFrameworkCore;

Console.WriteLine("=== Retail Inventory System ===\n");

try
{
    using var context = new AppDbContext();
```

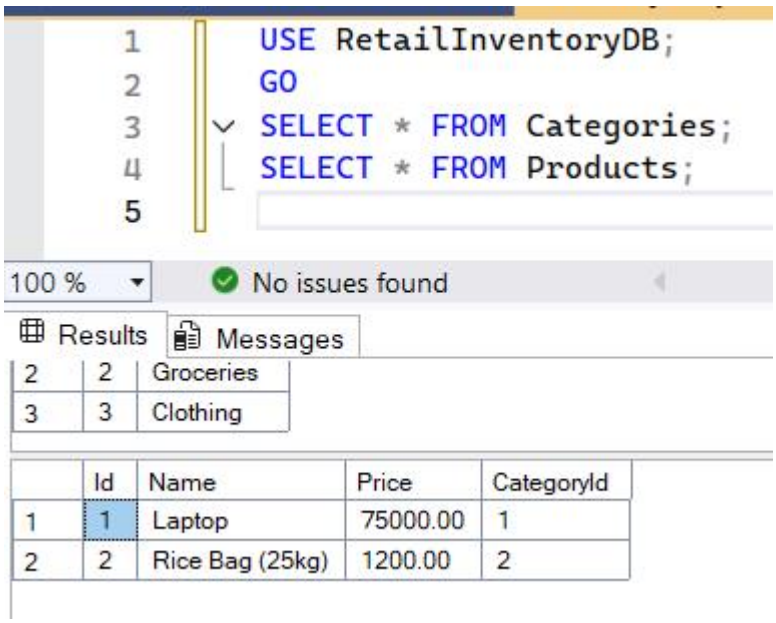
```
    await context.Database.EnsureCreatedAsync();

        if (!await context.Categories.AnyAsync())
    {
        Console.WriteLine("Inserting initial data...\n");

              var electronics = new Category { Name = "Electronics" };
        var groceries = new Category { Name = "Groceries" };


        var clothing = new Category { Name = "Clothing" };

        await context.Categories.AddRangeAsync(electronics, groceries,
clothing);

        var products = new List<Product>
        {
            new Product { Name = "Laptop", Price = 75000, Category =
electronics },
            new Product { Name = "Rice Bag (25kg)", Price = 1200, Category
= groceries },
            new Product { Name = "Jeans", Price = 2499, Category =
clothing },
        };

        await context.Products.AddRangeAsync(products);


        int recordsAffected = await context.SaveChangesAsync();
        Console.WriteLine($"Successfully inserted {recordsAffected}
records!\n");
    }
    else
    {
        Console.WriteLine("Data already exists. Skipping insertion.\n");
    }
}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}
```

RetailInventoryDB:

```
USE RetailInventoryDB;
GO
SELECT * FROM Categories;
SELECT * FROM Products;
```

Output:-





Lab 5: Retrieving Data from the Database

Scenario:

The store wants to display product details on the dashboard.

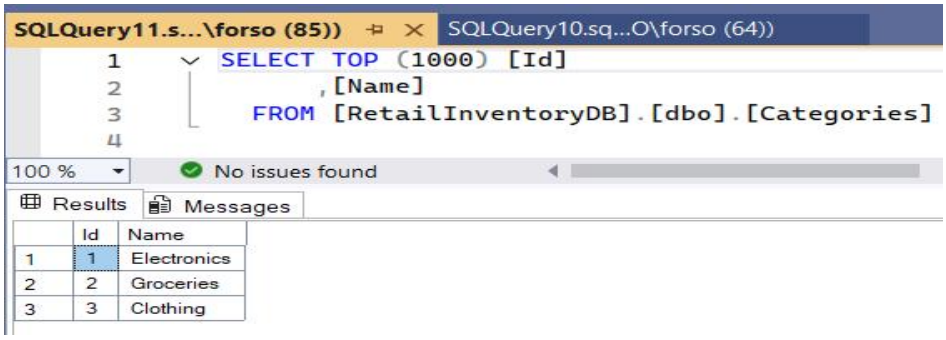
Ans) Code:-

Lab5/Program.cs

```
using Lab2.Data;
using Lab2.Models;
using Microsoft.EntityFrameworkCore;

Console.WriteLine("=== Data Retrieval Examples ===\n");

try
{
    using var context = new AppDbContext();

        Console.WriteLine("📋 ALL PRODUCTS:");
    Console.WriteLine("".PadRight(50, '-'));

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

    foreach (var product in products)
    {
        Console.WriteLine($"{product.Name} - ₹{product.Price:N0}
({product.Category.Name})");
    }

        Console.WriteLine($"\n🔍 FIND BY ID (ID: 1):");
    Console.WriteLine("".PadRight(50, '-'));

    var productById = await context.Products
        .Include(p => p.Category)
```

```
        .FirstOrDefaultAsync(p => p.Id == 1);

    if (productById != null)
    {
        Console.WriteLine($"Found: {productById.Name} -
{productById.Price:N0}");
    }
    else
    {
        Console.WriteLine("Product not found!");
    }

        Console.WriteLine($"\n EXPENSIVE PRODUCTS (Price > ₹50,000):");
    Console.WriteLine("".PadRight(50, '-'));

    var expensiveProducts = await context.Products
        .Include(p => p.Category)
        .Where(p => p.Price > 50000)
        .ToListAsync();

    if (expensiveProducts.Any())
    {
        foreach (var product in expensiveProducts)
        {
            Console.WriteLine($" {product.Name} - ₹{product.Price:N0}");
        }
    }
    else
    {
        Console.WriteLine("No expensive products found!");
    }

       Console.WriteLine($"\n STATISTICS:");
    Console.WriteLine("".PadRight(50, '-'));

    var totalProducts = await context.Products.CountAsync();
    var totalCategories = await context.Categories.CountAsync();
    var avgPrice = await context.Products.AverageAsync(p => p.Price);

    Console.WriteLine($"Total Products: {totalProducts}");
    Console.WriteLine($"Total Categories: {totalCategories}");
    Console.WriteLine($"Average Price: ₹{avgPrice:N2}");
}
catch (Exception ex)
{
    Console.WriteLine($"Error: {ex.Message}");
}
```

Output:-



SQLQuery10.s...\forso (64)) | SQLQuery9.sql...O\forso (79))

```
SELECT TOP (1000) [Id]
      ,[Name]
      ,[Price]
      ,[CategoryId]
  FROM [RetailInventoryDB].[dbo].[Products]
```

100 % — No issues found

Results | Messages

	Id	Name	Price	CategoryId
1	1	Laptop	75000.00	1
2	2	Rice Bag (25kg)	1200.00	2

```
PS C:\Users\forso\OneDrive\Desktop\Digital-Nurture-Solutions\Week3 - Entity Framework Core 8.0\Lab5> dotnet run
=== Data Retrieval Examples ===

 ALL PRODUCTS:
--------------------------------------------------
Laptop - ₹75,000 (Electronics)
Rice Bag (25kg) - ₹1,200 (Groceries)

 FIND BY ID (ID: 1):
--------------------------------------------------
Found: Laptop - 75,000

 EXPENSIVE PRODUCTS (Price > ₹50,000):
--------------------------------------------------
 Laptop - ₹75,000

 STATISTICS:
--------------------------------------------------
Total Products: 2
Total Categories: 3
Average Price: ₹38,100.00
```