**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

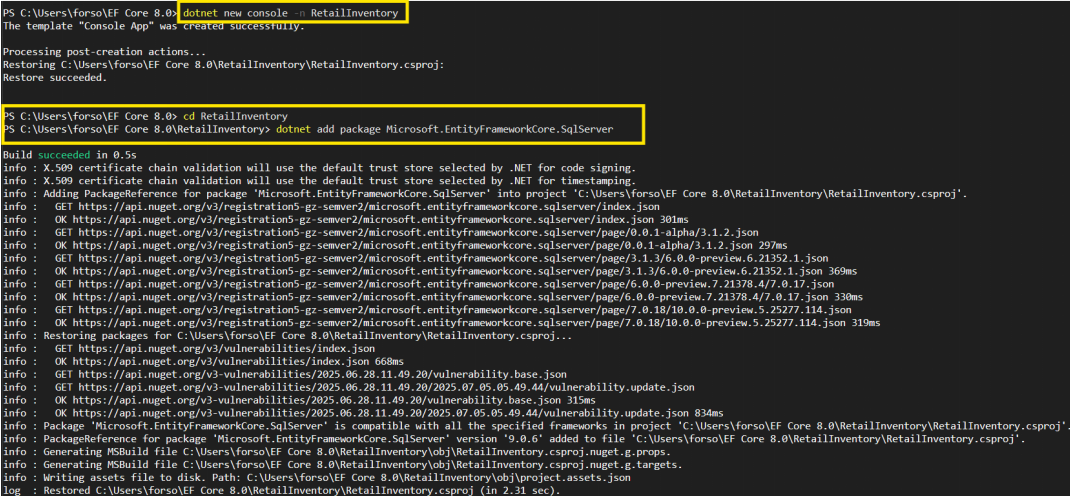
cd RetailInventory

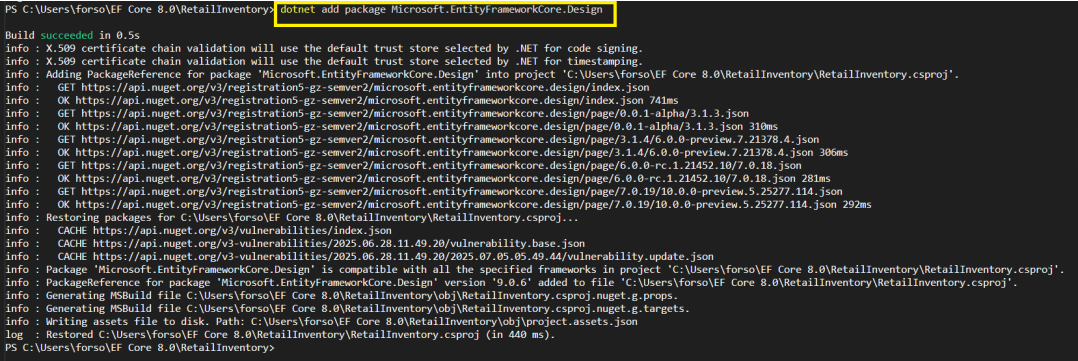
2. Install EF Core Packages:

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)

{

// Replace with your actual connection string

optionsBuilder.UseSqlServer("Server=.;Database=RetailInventoryDB;Trusted\_Connection=true;TrustServerCertificate=true;");

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Configure relationships

modelBuilder.Entity<Product>()

.HasOne(p => p.Category)

.WithMany(c => c.Products)

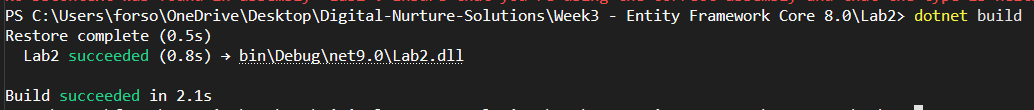
.HasForeignKey(p => p.CategoryId);

}

}

}

**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");

}

}

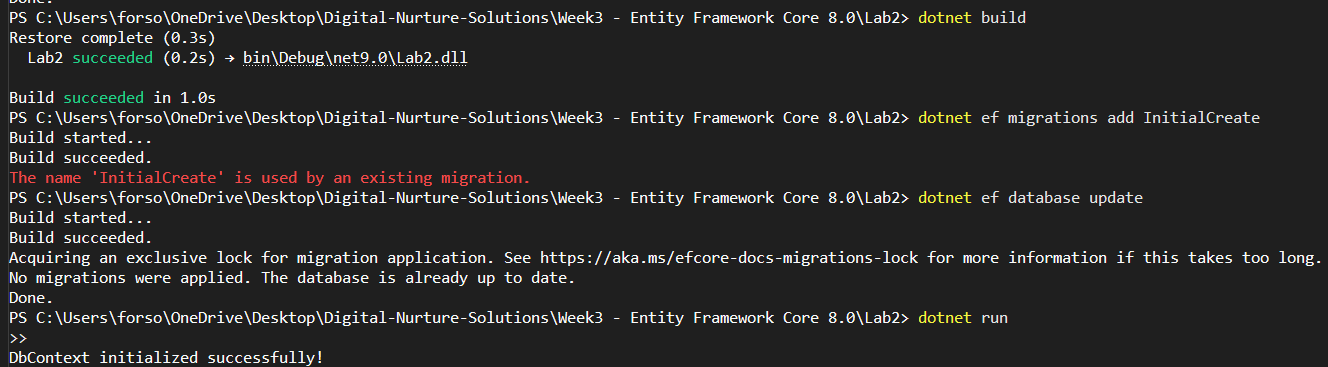
catch (Exception ex)

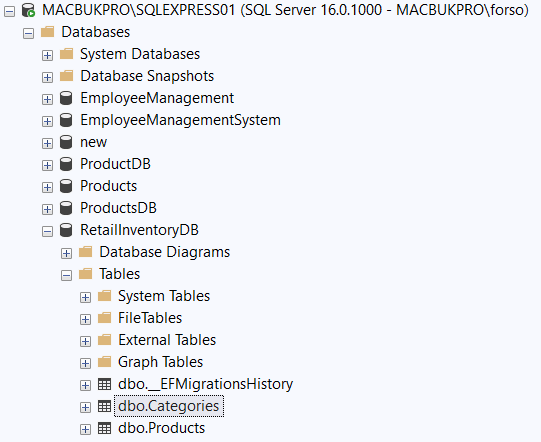
{

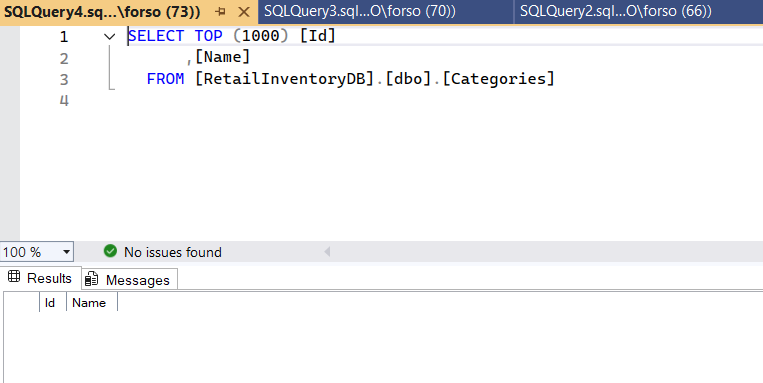
Console.WriteLine($"❌ Error: {ex.Message}");

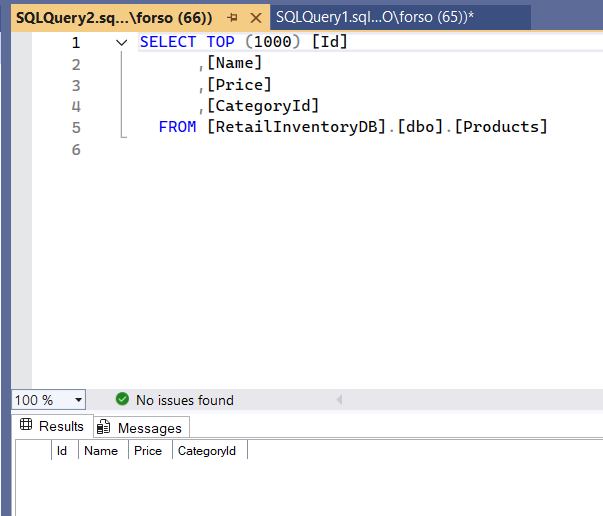
}

**Output:-**

****

****

****



**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();

// Ensure database is created

await context.Database.EnsureCreatedAsync();

// Check if data already exists

if (!await context.Categories.AnyAsync())

{

Console.WriteLine("Inserting initial data...\n");

// Create categories

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);

// Create products

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);

// Save changes

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:**

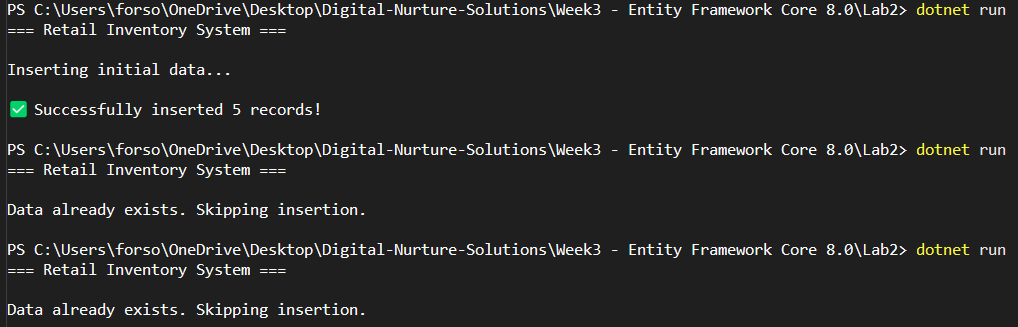
USE RetailInventoryDB;

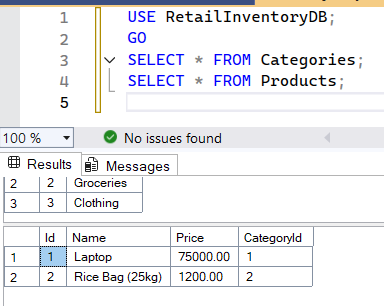
GO

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();

// 1. Retrieve All Products

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})");

}

// 2. Find by ID

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!");

}

// 3. FirstOrDefault with Condition

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!");

}

// 4. Count and Statistics

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}");

}

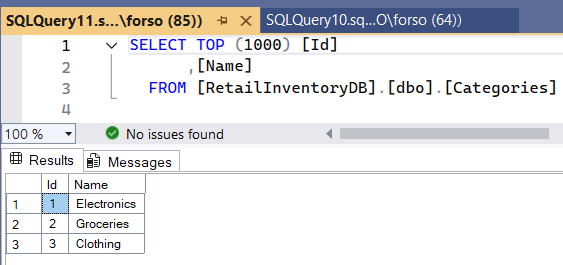
catch (Exception ex)

{

Console.WriteLine($"Error: {ex.Message}");

}

**Output:-**

****

