Entity Framework

Lab 1: Understanding ORM with a Retail Inventory System.

Solution:

**1. What is ORM (Object-Relational Mapping)?**

* It lets you treat database rows as C# objects.
* EF Core takes care of generating SQL and managing data.

**2. EF Core vs EF Framework:**

| **Feature** | **EF Core 8.0** | **EF Framework (EF6)** |
| --- | --- | --- |
| Platform | Cross-platform | Windows-only |
| Features | LINQ, async, compiled models | More mature, fewer new features |
| Performance | Faster (especially with compiled models) | Slower |

**3. EF Core 8.0 Features:**

* JSON column mapping (store complex data types as JSON).
* Compiled models = faster startup.
* Interceptors for logging, validation, etc.

dotnet new console -n RetailInventory

cd RetailInventory

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

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

Solution:

Product.cs:

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; }

}

Category.cs:

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

AppDbContext.cs

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(@"Server=localhost;Database=RetailInventoryDb;Trusted\_Connection=True;TrustServerCertificate=True;");

}

}

Dbseeder.cs

public static class DbSeeder

{

public static void Seed(AppDbContext db)

{

// Only seed if no categories/products exist

if (!db.Categories.Any() && !db.Products.Any())

{

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

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

db.Categories.AddRange(electronics, groceries);

db.Products.AddRange(

new Product { Name = "Laptop", Price = 75000, Category = electronics },

new Product { Name = "Rice Bag", Price = 1200, Category = groceries }

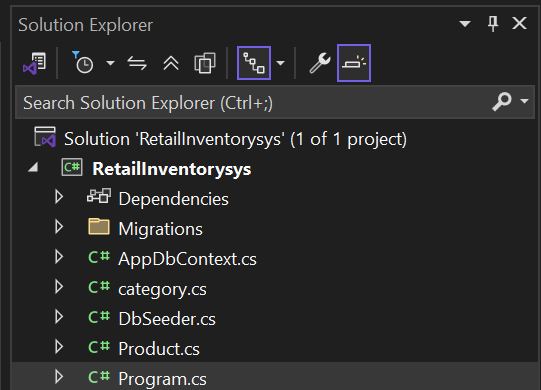
);

db.SaveChanges();

}

}

}



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

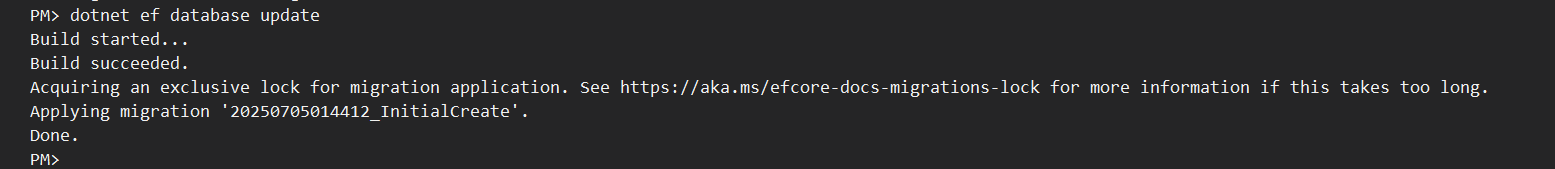
Solution:

A black screen with white text

AI-generated content may be incorrect.

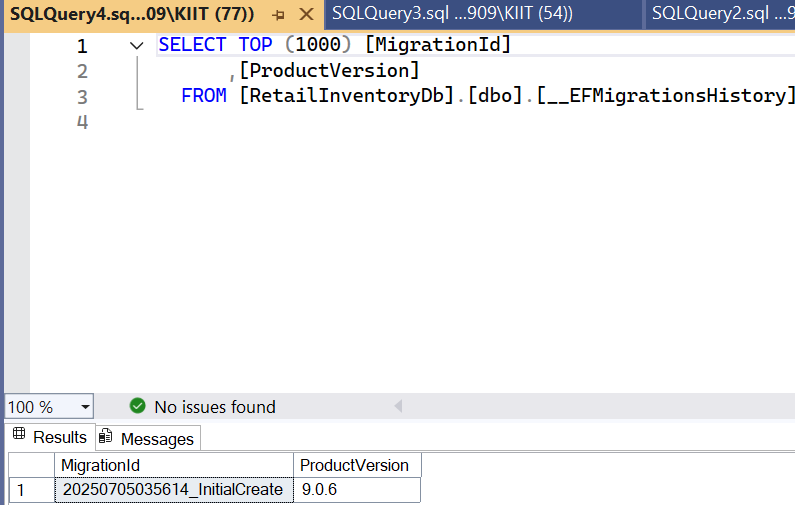
A screenshot of a computer

AI-generated content may be incorrect.



Snapshots of SQL server Management Studio:

A screenshot of a computer

AI-generated content may be incorrect. 

A screenshot of a computer

AI-generated content may be incorrect.

Lab 4: Inserting Initial Data into the Database.

Solution:

Program.cs

using System;

using System.Linq;

using System.Threading.Tasks;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

// Prevent duplicate seeding

if (!context.Categories.Any() && !context.Products.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

{

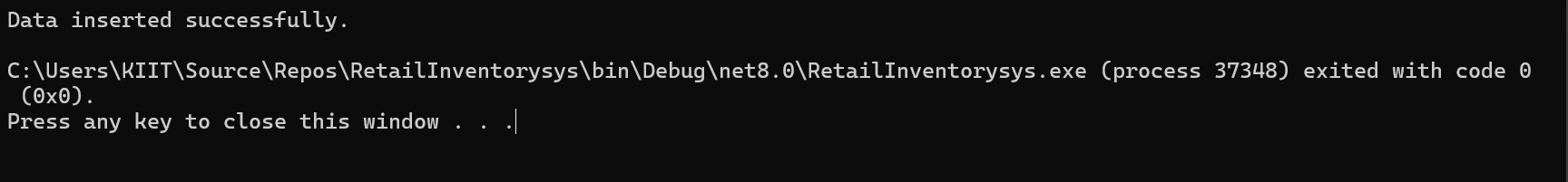
Console.WriteLine("Data already exists. Skipping seeding.");

}

}

}

Output:



Lab 5: Retrieve and Display Data

Updated Program.cs

using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

// ✅ 1. Print all products with category name

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

foreach (var p in products)

Console.WriteLine($"{p.Name} - INR{p.Price} - {p.Category?.Name}");

// ✅ 2. Find product by ID

var product = await context.Products.FindAsync(1);

Console.WriteLine($"Found: {product?.Name}");

// ✅ 3. Get first expensive product (Price > 50,000)

var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);

Console.WriteLine($"Expensive: {expensive?.Name}");

}

}

Output:

