EF Core 8.0 Guided Hands-On Exercises

**Lab 1: Understanding ORM with a Retail Inventory System**

1.What is ORM?

ORM (Object-Relational Mapping) is a programming technique that allows developers to interact with a relational database using C# classes instead of writing raw SQL queries.

ORM tools like Entity Framework Core automatically map:

* C# classes → Database tables
* Properties → Table columns
* Object relationships → Foreign keys and joins

Benefits of ORM:

| Benefit | Explanation |
| --- | --- |
| Productivity | No need to write SQL manually; use LINQ to query your data. |
| Maintainability | Changes to the model are automatically reflected in the database with migrations. |
| Abstraction | Keeps business logic separate from database logic. |
| Security | Prevents SQL injection by using parameterized queries. |

2.EF Core vs EF Framework:

| Feature | EF Core | EF Framework (EF6) |
| --- | --- | --- |
| Cross-platform | Yes (Windows, Linux, macOS) | Windows-only |
| Performance | Faster, lightweight | Slower due to legacy support |
| LINQ Support | Full support | Full support |
| Compiled Queries | Supported | Not supported |
| Asynchronous Queries | Native async support | Limited |
| Migrations | Built-in via CLI & tools | Supported but Windows-centric |
| Third-party extensibility | Better with modular architecture | Limited |
| Maturity | Newer (more evolving) | Older, very stable |

* 1. 3.EF Core 8.0 Features:

a. JSON Column Mapping

* Map C# classes or objects to a single JSON column in SQL Server (or PostgreSQL).
* Useful for semi-structured data.

b. Compiled Models and Queries

* EF Core 8 allows you to compile the entire model ahead-of-time to speed up startup time.
* You can also compile LINQ queries for better runtime performance:

c. Interceptors & Better Bulk Operations

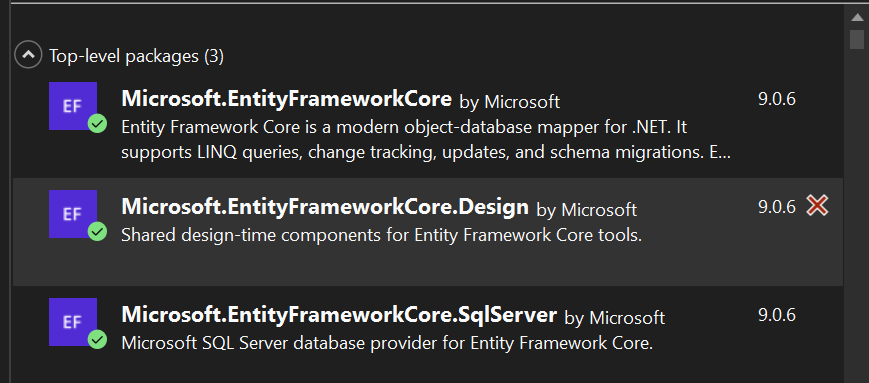
* Interceptors let you intercept and log/modify database operations like command execution, saving, etc.
* New improvements to bulk operations make it easier and faster to insert/update/delete many rows.

4.Create a .NET Console App:

A screenshot of a computer

AI-generated content may be incorrect.

5.Install EF Core Packages:



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

AppDbContext.cs :

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

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.UseSqlServer(@"Server=localhost\MSSQLSERVER01;Database=RetailInventoryDB;Trusted\_Connection=True;Encrypt=False;");

}

}

}

Category.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RetailInventory.Models

{

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

}

Products.cs :

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RetailInventory.Models

{

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; }

}

}

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

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

Terminal:

dotnet ef migrations add InitialCreate

dotnet ef database update

dotnet ef database update

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

**Lab 4: Inserting Initial Data into the Database**

Program.cs:

using RetailInventory.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RetailInventory

{

public class Program

{

public static async Task Main(string[] args)

{

using var context = new AppDbContext();

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();

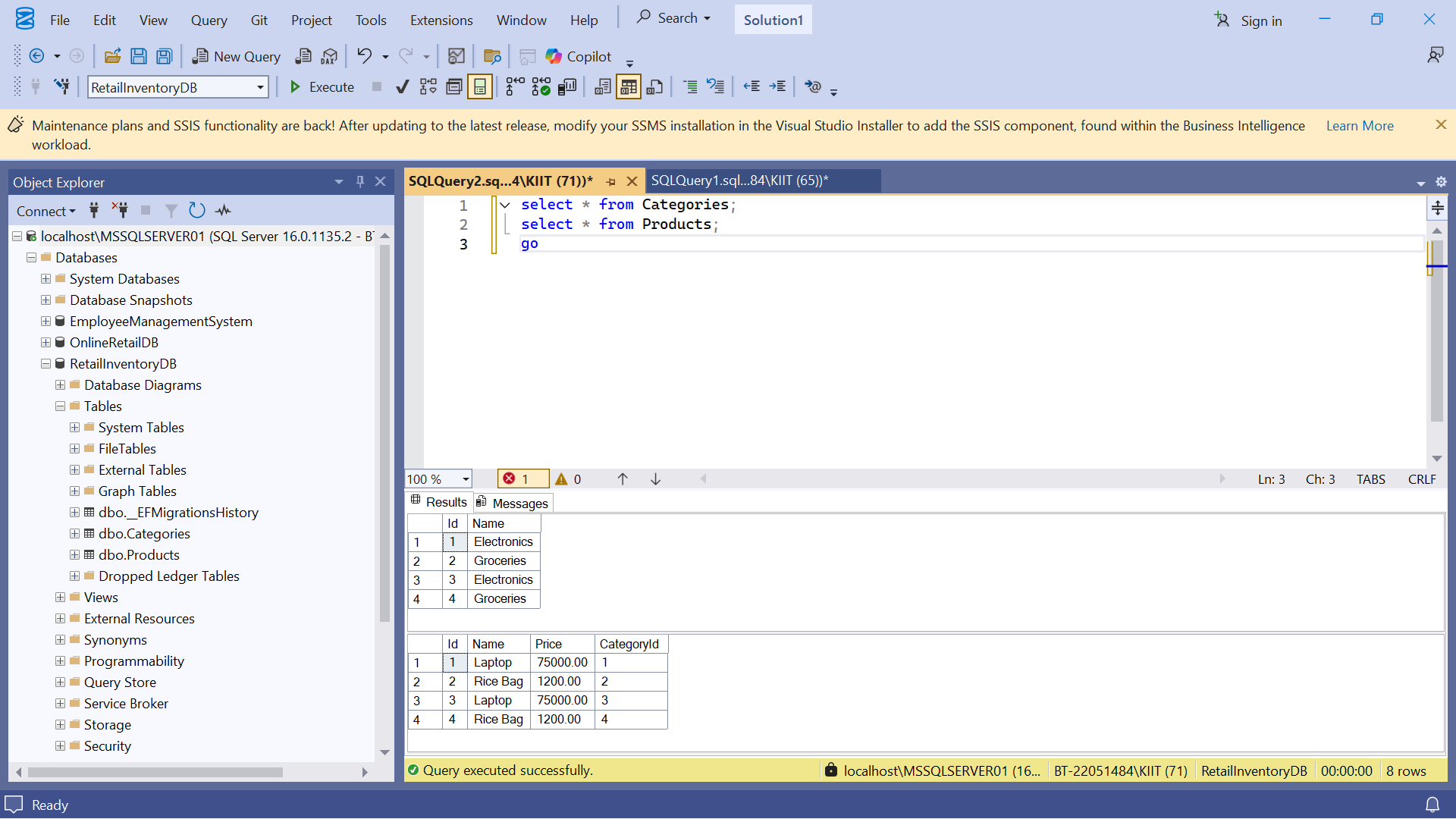
Console.WriteLine("Initial data seeded.");

}

}

}

OUTPUT:



**Lab 5: Retrieving Data from the Database**

Program.cs:

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RetailInventory

{

public class Program

{

public static async Task Main(string[] args)

{

using var context = new AppDbContext();

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

foreach (var product in products)

{

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

}

// Find by ID

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

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

// FirstOrDefault

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

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

}

}

}

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.