**Name: Deepanshu Rathore**

**Superset ID: 6358199**  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Week 3  
**EF Core 8.0 Guided Hands-On Exercises**  
Lab 1: Understanding ORM with a Retail Inventory System

**What is ORM?**

ORM stands for Object-Relational Mapping. It’s a technique that allows us to interact with a relational database using object-oriented code instead of writing raw SQL queries. In C#, we define classes like Product or Category, and ORM tools like Entity Framework Core take care of mapping these classes to tables in the database. Each class becomes a table, and each property in the class becomes a column. This mapping makes it easier to work with databases because we can just create objects, set their properties, and save them and the ORM will handle converting that into proper SQL commands behind the scenes.

The main benefits of using an ORM are increased productivity and better maintainability of the code. Since we don’t have to write SQL for every database operation, development becomes faster. Also, changes to the data model can be managed in code, which makes it easier to maintain. ORM also abstracts the database layer, so we can focus on our application logic instead of worrying about SQL syntax or database-specific quirks.

**2. EF Core vs EF Framework**

Entity Framework (EF) comes in two main versions, EF Core and the older EF Framework (also known as EF6). EF Core is the newer, modern version. It’s lightweight and designed to work across different platforms like Windows, Linux, and macOS. It supports modern features like asynchronous queries with async/await, LINQ (Language Integrated Query), and compiled queries for better performance. EF Core is also constantly evolving and getting new features with each release.

On the other hand, EF Framework (EF6) is more mature but limited to Windows and the .NET Framework. It has been around longer and might have more stability in legacy systems, but it lacks the flexibility and performance improvements that EF Core provides. For most modern applications, especially those built with .NET Core or .NET 6/7/8, EF Core is the recommended option.

**3. EF Core 8.0 Features**

EF Core 8.0 introduced several new features that make it more powerful and efficient. One of the key features is **JSON column mapping**, which allows storing and querying JSON data directly in database columns, useful when dealing with semi-structured data. This helps when we want to store nested or dynamic information without creating a separate table.

Another improvement is **compiled models**, which boost performance by reducing the startup time of the app. Instead of building the model in memory every time the application runs, EF Core can precompile it ahead of time, making the application start faster and use fewer resources.

Lastly, EF Core 8.0 has improved **interceptors and bulk operations**. Interceptors allow us to hook into database operations like SaveChanges to perform logging, validation, or custom logic. The improvements in bulk operations also help when inserting or updating large amounts of data at once, which is very useful in real-world applications dealing with big datasets.

**Code:**  
Program.cs  
using System;

using System.Linq;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Data;

using RetailInventory.Models;

namespace RetailInventory

{

internal class Program

{

private static void Main(string[] args)

{

// 1. Create DbContext instance

using var db = new RetailContext();

// 2. READ: List all products with their category names

Console.WriteLine("---- PRODUCTS BEFORE ADDING ----");

var allProducts = db.Products.Include(p => p.Category).ToList();

foreach (var p in allProducts)

{

Console.WriteLine($"{p.ProductId}: {p.Name} ({p.Category.Name}) - Stock: {p.Stock} - Price: {p.Price:C}");

}

// 3. CREATE: Add a new product under “Beverages”

var beverageCategory = db.Categories.Single(c => c.Name == "Beverages");

var newProduct = new Product

{

Name = "Tea",

Price = 2.75m,

Stock = 150,

CategoryId = beverageCategory.CategoryId

};

db.Products.Add(newProduct);

db.SaveChanges(); // Persist the new product

Console.WriteLine("\nAdded new product: Tea");

// 4. UPDATE, Increase the stock of “Chips” by 50

var chips = db.Products.Single(p => p.Name == "Chips");

Console.WriteLine($"\nPrevious Chips stock: {chips.Stock}");

chips.Stock += 50;

db.SaveChanges();

Console.WriteLine($"Updated Chips stock: {chips.Stock}");

// 5. DELETE- Remove the “Snacks” category and its products (cascade)

var snacksCategory = db.Categories.Include(c => c.Products)

.Single(c => c.Name == "Snacks");

db.Categories.Remove(snacksCategory);

db.SaveChanges();

Console.WriteLine($"\nDeleted category: Snacks (and its products)");

// 6. Show remaining products

Console.WriteLine("\n---- PRODUCTS AFTER CHANGES ----");

var finalProducts = db.Products.Include(p => p.Category).ToList();

foreach (var p in finalProducts)

{

Console.WriteLine($"{p.ProductId}: {p.Name} ({p.Category.Name}) - Stock: {p.Stock}");

}

}

}

}  
  
RetailContext.cs  
using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory.Data

{

public class RetailContext : DbContext

{

public DbSet<Category> Categories { get; set; } = null!;

public DbSet<Product> Products { get; set; } = null!;

protected override void OnConfiguring(DbContextOptionsBuilder options)

=> options.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=RetailInvDb;Trusted\_Connection=True;");

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Category>().HasData(

new Category { CategoryId = 1, Name = "Beverages" },

new Category { CategoryId = 2, Name = "Snacks" }

);

modelBuilder.Entity<Product>().HasData(

new Product { ProductId = 1, Name = "Coffee", Price = 3.50m, Stock = 100, CategoryId = 1 },

new Product { ProductId = 2, Name = "Chips", Price = 1.25m, Stock = 200, CategoryId = 2 }

);

}  
**Output:**  
A screenshot of a computer

AI-generated content may be incorrect.

Lab 2: Setting Up the Database Context for a Retail Store  
**Code:**  
Program.cs:

using System;

using System.Linq;

using RetailStore.Data;

using RetailStore.Models;

namespace RetailStore

{

internal class Program

{

private static void Main(string[] args)

{

using var db = new AppDbContext();

// Ensure DB exists (creates if missing)

db.Database.EnsureCreated();

// Seed one category & product if empty

if (!db.Categories.Any())

{

var c = new Category { Name = "Default Category" };

db.Categories.Add(c);

db.SaveChanges();

db.Products.Add(new Product

{

Name = "Sample Product",

Price = 9.99m,

CategoryId = c.Id

});

db.SaveChanges();

}

// Read & print

var items = db.Products

.OrderBy(p => p.Id)

.Select(p => $"{p.Name} (Category: {p.Category!.Name})")

.ToList();

Console.WriteLine("Products in DB:");

items.ForEach(Console.WriteLine);

}

}

}

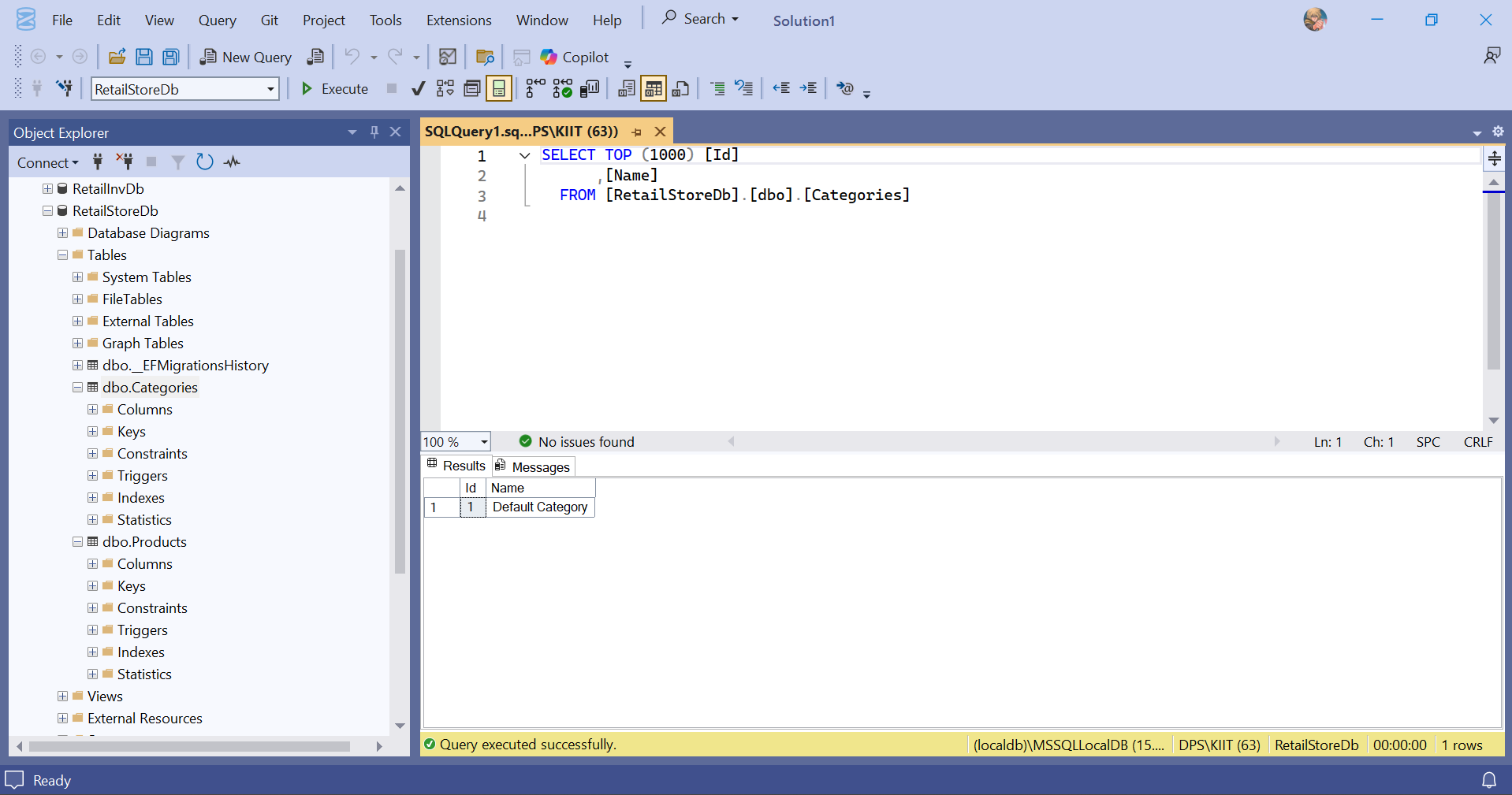
**Output:** **A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

Lab 3: Using EF Core CLI to Create and Apply Migrations  
**Terminal:** PS E:\KIIT\Cognizant\Programs\week\_3\RetailStore> dotnet tool install --global dotnet-ef

**Output:** ****

**A screenshot of a computer program

AI-generated content may be incorrect.**

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

using Microsoft.EntityFrameworkCore;

using RetailStore.Data;

using RetailStore.Models;

using System;

using System.Linq;

using System.Threading.Tasks;

namespace RetailStore

{

internal class Program

{

private static async Task Main(string[] args)

{

await using var db = new AppDbContext();

await db.Database.EnsureCreatedAsync();

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

{

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

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

await db.Categories.AddRangeAsync(electronics, groceries);

var product1 = new Product

{

Name = "Laptop",

Price = 75000m,

Category = electronics

};

var product2 = new Product

{

Name = "Rice Bag",

Price = 1200m,

Category = groceries

};

await db.Products.AddRangeAsync(product1, product2);

await db.SaveChangesAsync();

}

var items = await db.Products

.Include(p => p.Category)

.OrderBy(p => p.Id)

.Select(p => $"{p.Name} (Category: {p.Category!.Name})")

.ToListAsync();

Console.WriteLine("Products in DB:");

items.ForEach(Console.WriteLine);

}

}

}  
**Output:** **A screenshot of a computer program

AI-generated content may be incorrect.**

Lab 5: Retrieving Data from the Database  
**Code:** using Microsoft.EntityFrameworkCore;

using RetailStore.Data;

using RetailStore.Models;

using System;

using System.Linq;

using System.Threading.Tasks;

namespace RetailStore

{

internal class Program

{

private static async Task Main(string[] args)

{

await using var db = new AppDbContext();

await db.Database.EnsureCreatedAsync();

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

{

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

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

await db.Categories.AddRangeAsync(electronics, groceries);

var product1 = new Product

{

Name = "Laptop",

Price = 75000m,

Category = electronics

};

var product2 = new Product

{

Name = "Rice Bag",

Price = 1200m,

Category = groceries

};

await db.Products.AddRangeAsync(product1, product2);

await db.SaveChangesAsync();

Console.WriteLine("Seed data inserted successfully.");

}

var allProducts = await db.Products.ToListAsync();

Console.WriteLine("\nAll Products:");

foreach (var p in allProducts)

{

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

}

var productById = await db.Products.FindAsync(1);

Console.WriteLine($"\nFound (ID = 1): {productById?.Name ?? "Not Found"}");

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

Console.WriteLine($"\nExpensive Product: {expensive?.Name ?? "None over ₹50000"}");

var productsWithCategories = await db.Products

.Include(p => p.Category)

.OrderBy(p => p.Id)

.ToListAsync();

Console.WriteLine("\nProducts with Categories:");

foreach (var p in productsWithCategories)

{

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

}

}

}

}  
**Output:** **A screenshot of a computer program

AI-generated content may be incorrect.**