**Entity Framework Core 8.0 Hands-on**

**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.

**Objective:**  
Understand what ORM is and how EF Core helps bridge the gap between C# objects and relational tables.

Product.cs

public class Product

{

public int ProductId { get; set; }

public string Name { get; set; }

public int StockLevel { get; set; }

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

}

Category.cs

using System.Collections.Generic;

public class Category

{

public int CategoryId { get; set; }

public string Name { get; set; }

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

}

RetailContext.cs

using Microsoft.EntityFrameworkCore;

public class RetailContext : 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;");

}

}

Program.cs

using System;

using System.Linq;

using Microsoft.EntityFrameworkCore;

class Program

{

static void Main()

{

using (var context = new RetailContext())

{

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

var product = new Product

{

Name = "Smartphone",

Price = 25000,

StockLevel = 10,

Category = category

};

context.Categories.Add(category);

context.Products.Add(product);

context.SaveChanges();

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

foreach (var p in products)

{

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

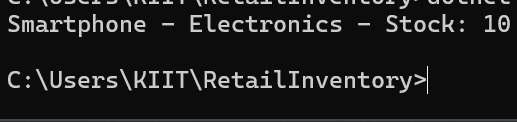
}

}

}

}

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.

#### ****Objective:****

Configure DbContext and connect to SQL Server.

### ****Steps:****

#### ****1. Create Models:****

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

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; }

}

#### 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

using System.Collections.Generic;

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=RetailStoreDB;Trusted\_Connection=True;TrustServerCertificate=True;");

}

}

#### Program.cs

using System;using System.Linq;using Microsoft.EntityFrameworkCore;

class Program

{

static void Main()

{

using (var context = new AppDbContext())

{

if (!context.Categories.Any())

{

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

context.Categories.Add(category);

context.SaveChanges();

var product = new Product { Name = "Smartphone", Price = 45000, CategoryId = category.Id };

context.Products.Add(product);

context.SaveChanges();

}

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

foreach (var product in products)

{

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

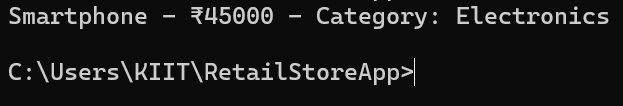
}

}

}

}

**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.

### ****Objective:****

Learn how to use **EF Core CLI** to manage database schema changes.

### ****Steps:****

#### ****1. Install EF Core CLI (if not already):****

dotnet tool install --global dotnet-ef

#### ****2. Create Initial Migration:****

dotnet ef migrations add InitialCreate

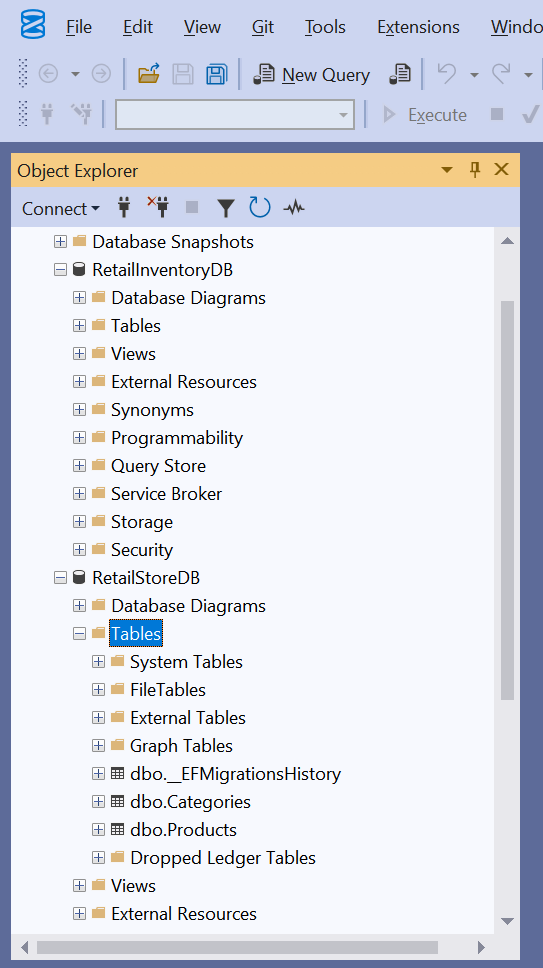
This generates a Migrations folder with code that represents the schema.

#### ****3. Apply Migration to Create Database:****

#### ****4. Verify in SQL Server:****

Open **SQL Server Management Studio (SSMS)** or **Azure Data Studio** and confirm that the database was created successfully.

**Output:**



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

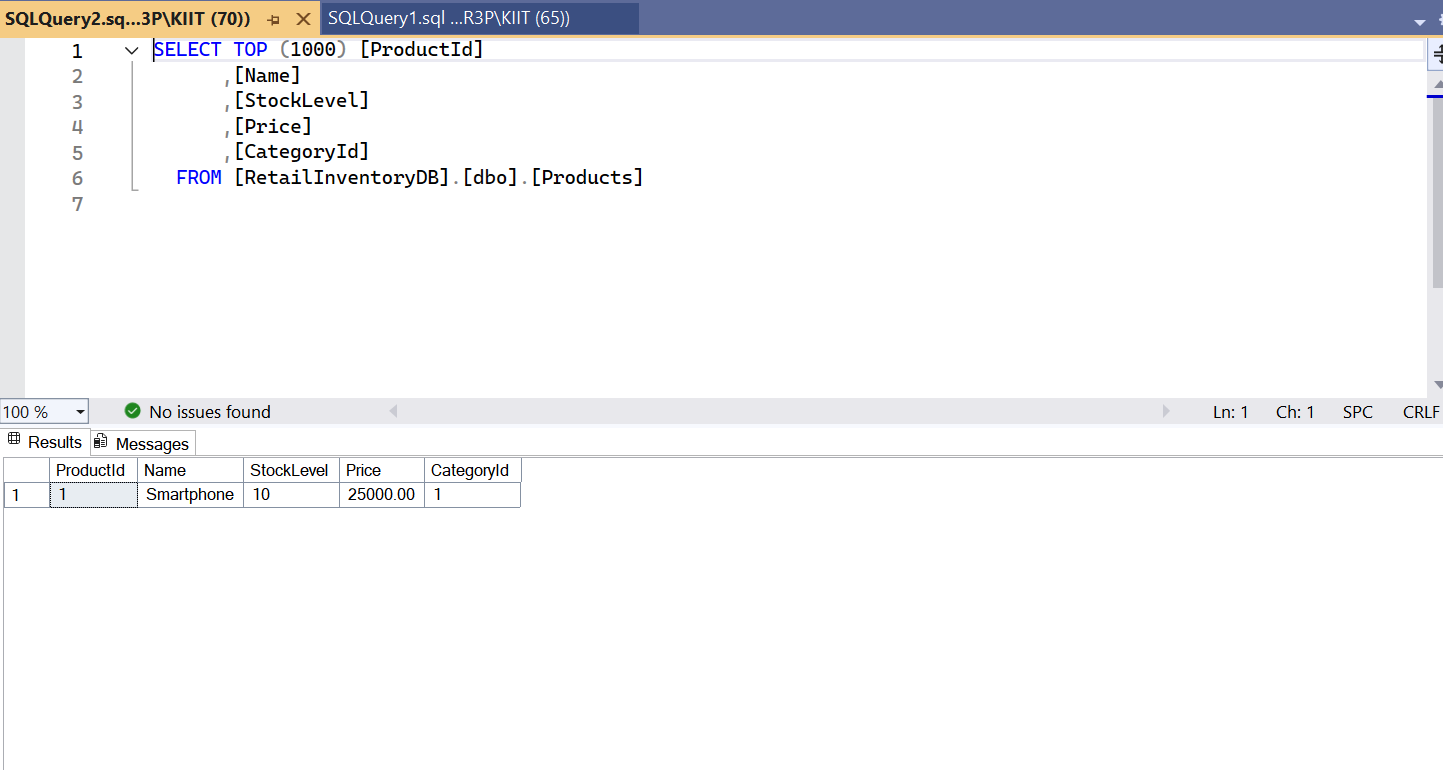
#### **Scenario:**

The store manager wants to initialize the system with some basic product categories and products, such as "Electronics" and "Groceries", along with a few sample items like a laptop and a rice bag.

#### **Objective:**

To learn how to use EF Core methods like AddRangeAsync() and SaveChangesAsync() to insert records into a SQL Server database.

**Output:**



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

#### ****Scenario:****

The retail store now wants to display product details on their dashboard, such as product names and prices.

#### ****Objective:****

Learn how to **retrieve data** from SQL Server using Entity Framework Core methods like ToListAsync(), FindAsync(), and FirstOrDefaultAsync().

**Output:**

