**Entity Framework Core 8.0**

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

**1** **What is ORM**

**Ans :ORM** stands for **Object-Relational Mapping**. It is a technique that allows developers to interact with a **relational database** (like SQL Server, MySQL, etc.) using **object-oriented programming (OOP)** concepts. Instead of writing raw SQL queries, developers can use objects, classes, and methods to perform database operations like insert, update, delete, and read.

In C#, ORM tools like **Entity Framework Core** automatically map classes to database tables using a process called **convention-based mapping** or **attribute-based configuration**.

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

**Ans: Entity Framework Core (EF Core)** is a modern, lightweight, and cross-platform ORM built from scratch to support .NET Core and beyond. It offers advanced features like full async support, LINQ, and compiled queries for better performance and flexibility. In contrast, **Entity Framework 6 (EF6)** is a mature ORM limited to the Windows-only .NET Framework. While EF6 has a larger set of legacy features and broader stability for older applications, it lacks the performance optimizations and cross-platform support found in EF Core. Therefore, EF Core is ideal for new, scalable, and cloud-based applications, whereas EF6 is better suited for maintaining existing Windows-based systems.

**3.EF Core 8.0 Features:**

**1. JSON Column Mapping**

Allows storing and querying structured data (like dictionaries or nested objects) directly in a JSON column in SQL Server.

**2. Compiled Models**

Improves startup and runtime performance by compiling your entity models ahead of time.

**3. Interceptors**

You can hook into EF Core’s behavior (like logging SQL queries, auditing, or preventing certain writes).

**4. Better Bulk Operations**

More efficient insert/update/delete operations in bulk.

4&5 Question:

Create .NET Console App

dotnet new console -n RetailInventory

cd RetailInventory

Install EF Core Packages

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Design

Define Models

Models/Product.cs

namespace RetailInventory.Models

{

public class Product

{

public int ProductId { get; set; }

public string Name { get; set; }

public int Stock { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

}

}

Models/Category.cs

namespace RetailInventory.Models

{

public class Category

{

public int CategoryId { get; set; }

public string CategoryName { get; set; }

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

}

}

Define DbContext

RetailDbContext.cs

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

public class RetailDbContext : DbContext

{

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

public DbSet<Category> Categories { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("Your\_Connection\_String\_Here");

}

}

Create Initial Migration

dotnet ef migrations add InitialCreate

dotnet ef database update

Example Usage

using RetailInventory.Models;

class Program

{

static void Main()

{

using var context = new RetailDbContext();

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

context.Categories.Add(electronics);

var product = new Product { Name = "Smartphone", Stock = 50, Category = electronics };

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.Stock} units - Category: {p.Category.CategoryName}");

}

}

Run the App

dotnet run

**OUTPUT:**

****

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

**Prerequisites**

Ensure you have:

* [.NET 6+ SDK](https://dotnet.microsoft.com/en-us/download)
* SQL Server or SQL Server Express installed
* EF Core CLI Tools installed:

dotnet tool install --global dotnet-ef

**Project Setup**

dotnet new console -n RetailInventory

cd RetailInventory

*Add EF Core:*

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Design

dotnet add package Microsoft.Extensions.Configuration

dotnet add package Microsoft.Extensions.Configuration.Json

dotnet add package Microsoft.Extensions.DependencyInjection

**Entity Models**

Models/Product.cs

public class Product

{

public int ProductId { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public int Stock { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

}

Models/Category.cs

public class Category

{

public int CategoryId { get; set; }

public string CategoryName { get; set; }

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

}

**Setup AppDbContext**

Data/AppDbContext.cs

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using RetailInventory.Models;

namespace RetailInventory.Data

{

public class AppDbContext : DbContext

{

private readonly IConfiguration \_configuration;

public AppDbContext(IConfiguration configuration)

{

\_configuration = configuration;

}

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

public DbSet<Category> Categories { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

var connectionString = \_configuration.GetConnectionString("DefaultConnection");

optionsBuilder.UseSqlServer(connectionString);

}

}

}

**Add Configuration File**

appsettings.json

{

"ConnectionStrings": {

"DefaultConnection": "Your\_Connection\_String\_Here"

}

}

**Configure Program.cs**

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using RetailInventory.Data;

namespace RetailInventory

{

internal class Program

{

static void Main(string[] args)

{

// Load configuration

var configuration = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json")

.Build();

// Setup DI

var services = new ServiceCollection()

.AddSingleton<IConfiguration>(configuration)

.AddDbContext<AppDbContext>()

.BuildServiceProvider();

using var context = services.GetService<AppDbContext>();

Console.WriteLine("Database context initialized using appsettings.json");

}

}

}

**Migrations and Database Creation**

dotnet ef migrations add InitialCreate

dotnet ef database update

OUTPUT:



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

Models Used

Category.cs

public class Category

{

public int Id { get; set; }

public string? Name { get; set; }

public ICollection<Product>? Products { 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; }

}

EF Core CLI Steps

Install EF Core CLI

dotnet tool install --global dotnet-ef

Verify installation using:

dotnet ef --version

Create Initial Migration

dotnet ef migrations add InitialCreate

Apply Migration and Create Database

dotnet ef database update

Verify in SQL Server

Open SQL Server Management Studio (SSMS) or Azure Data Studio

Connect to:

(localdb)\MSSQLLocalDB

Check that the database RetailDb has been created

Verify that the tables Products and Categories exist under Tables

NOTES

If you encounter the error: "There is already an object named 'Categories' in the database"

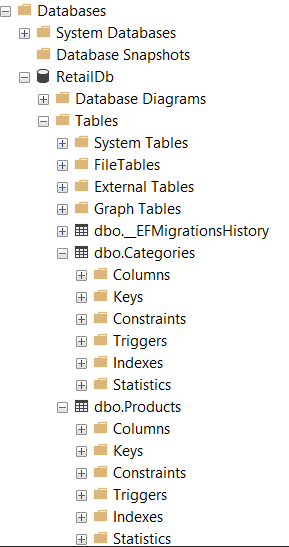
Run:

dotnet ef database drop

dotnet ef database update

Seed initial data in OnModelCreating() using HasData() for testing

OUTPUT:



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

Models Used

Category.cs

public class Category

{

public int Id { get; set; }

public string? Name { get; set; }

public ICollection<Product>? Products { 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; }

}

Steps

Setup and Configuration

Used AppDbContext to connect to the database via connection string in appsettings.json.

Dependency Injection was used to inject DbContext through ServiceCollection.

Inserting Data

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();

Run the Application

dotnet run

Verify Insertion

Open SQL Server Management Studio (SSMS) or Azure Data Studio

Execute the following queries to verify inserted data:

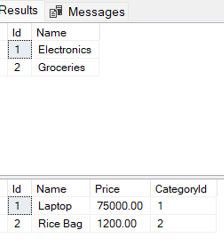
USE RetailDb;

GO

SELECT \* FROM Categories;

SELECT \* FROM Products;

OUTPUT:



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

Models Used

Category.cs

public class Category

{

public int Id { get; set; }

public string? Name { get; set; }

public ICollection<Product>? Products { 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; }

}

Steps

Retrieve All Products

Use ToListAsync() to fetch all product records.

var products = await context.Products.ToListAsync();

foreach (var p in products)

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

Find Product by ID

Use FindAsync() to retrieve a product using its primary key.

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

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

FirstOrDefault with Condition

Use FirstOrDefaultAsync() to get the first product with price greater than ₹50,000.

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

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

OUTPUT:

