COGNIZANT DN - 4.0 DEEP SKILLING

HANDS ON WEEK-3

Entity Framework Core 8.0

Lab 1: Understanding ORM with a Retail Inventory System

1. **What is ORM ?**

**Solution:**

ORM is a technique that allows you to interact with a relational database (like SQL Server) using object-oriented code (C# classes), instead of writing raw SQL queries.

### How it works:

* C# classes map to database tables.
* Properties in C# map to columns in the database.
* Navigation properties handle relationships like one-to-many, etc.

### Benefits of ORM:

* **Productivity**: Write less boilerplate SQL code.
* **Maintainability**: Change C# classes, and EF will track the schema changes.
* **Abstraction**: Avoid raw SQL; use LINQ (Language Integrated Query).

1. **EF Core vs EF Framework:**

**Solution:**

| **Feature** | **EF Core** | **EF Framework (EF6)** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| Platform | Cross-platform (.NET Core) | Windows-only |

|  |  |  |
| --- | --- | --- |
| Performance | Better (especially EF Core 8) | Moderate |

|  |  |  |
| --- | --- | --- |
| LINQ Support | Full | Full |

|  |  |  |
| --- | --- | --- |
| Async Support | Full | Partial |

|  |  |  |
| --- | --- | --- |
| JSON Support | Yes(EF Core 8) | No |

|  |  |  |
| --- | --- | --- |
| Compiled Queries | Yes No |  |

|  |  |  |
| --- | --- | --- |
| Community and Updates | Active (latest version) | Legacy |

1. **EF Core 8.0 Features:**

* **JSON column mapping**: Map C# objects to JSON columns in SQL Server.
* **Compiled models**: Boost performance by compiling the model at build time.
* **Interceptors**: Add logic during DB operations (e.g., logging, auditing).
* **Bulk operations**: Improved handling of inserts, updates, and deletes in large volumes.

1. **Create a .NET Console App:**

****

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

**1.Create Models**

using System.Collections.Generic;

namespace RetailInventory.Models

{

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

}

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; }

}

}

**2. Create AppDbContext:**

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

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=RetailDB;Trusted\_Connection=True;");

}

}

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

### Install EF Core CLI (if not already)

### 

### Create Initial Migration:

### dotnet ef migrations add InitialCreate

### Apply Migration to Create Database:

### 

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

### 1.Insert data in program.cs

using RetailInventory; // Adjust namespace if needed

using System;

using System.Threading.Tasks;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

// Create categories

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

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

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

// Create products and link them to categories

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);

// Save changes to database

await context.SaveChangesAsync();

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

}

}

Lab 5: Retrieving Data from the Database

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RetailInventory; // Adjust if your namespace is different

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

// 1. Retrieve All Products

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

Console.WriteLine("All Products:");

foreach (var p in products)

{

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

}

// 2. Find by ID

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

Console.WriteLine($"\nFound by ID: {product?.Name}");

// 3. FirstOrDefault with Condition

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

Console.WriteLine($"\nExpensive Product (Price > ₹50,000): {expensive?.Name}");

}

}





