EF Core 8.0- Hands-On Exercises Detailed Report & Outputs(Labs 1-7)

Submitted by- Richa Goyal(6365275)

Lab 1: Understanding ORM with a Retail Inventory System

Objective: Understanding what ORM is and how Entity Framework Core helps bridge the gap between C# objects and relational tables.

-What is ORM?

ORM (Object-Relational Mapping) is a technique that allows developers to interact with a database using C# objects instead of SQL queries. In EF Core, C# classes (like Product, Category) are mapped to database tables, and object properties map to table columns.

-Benefits of ORM:

- Productivity: No need to write repetitive SQL queries.
- Maintainability: Strongly-typed models are easier to debug and update. - Abstraction: Developers focus on code, not database structure.

EF Core vs Entity Framework 6 (EF6):

- EF Core is lightweight, cross-platform, and modern. Supports async, compiled queries, LINQ, etc.
- EF6 is mature but Windows-only and less flexible.

EF Core 8.0 New Features:

- JSON column mapping for flexible data storage.
- Improved performance using compiled models.
- Support for interceptors and better bulk operations.

Setup Performed:

- Created a new console project using: dotnet new console -n RetailInventory - Installed necessary EF Core packages:

Microsoft.EntityFrameworkCore.SqlServer
Microsoft.EntityFrameworkCore.Design

Output:

✓ Project created and EF Core packages installed successfully.

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

Objective: Create the data model and set up the database context in C#.

Models Created:

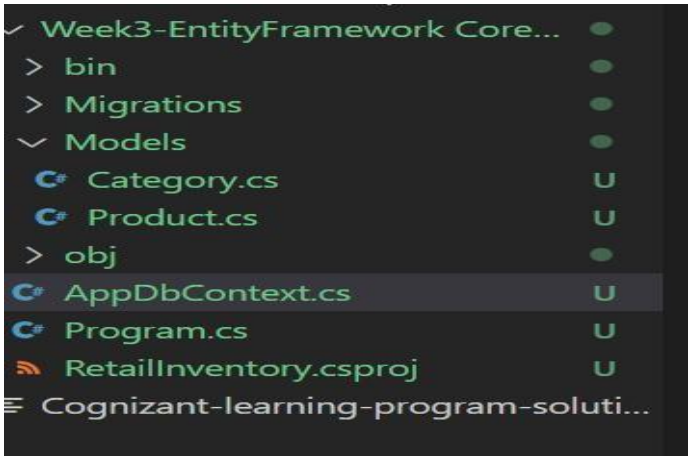
- Category.cs with properties: Id, Name, and List<Product> Products. - Product.cs with properties: Id, Name, Price, CategoryId (foreign key), and navigation property Category.

DbContext Setup:

- AppDbContext.cs created with DbSets for Products and Categories.
- Configured SQL Server connection using OnConfiguring() method. - Replaced connection string with my actual SQL Server connection string as: ***optionsBuilder.UseSqlServer("Server=localhost\\SQLEXPRESS;Database=Retail Db;Trusted_Connection=True;TrustServerCertificate=True;");***

Output:

✓ Models and AppDbContext set up successfully.



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

Objective: Generate database schema using EF Core CLI and create tables in SQL Server.

Steps Performed:

- Installed EF CLI using: dotnet tool install --global dotnet-ef
- Created initial migration: dotnet ef migrations add InitialCreate
- Applied migration: dotnet ef database update

Output:

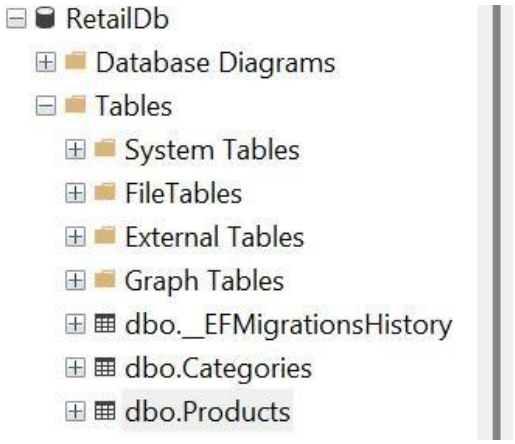
```
PS C:\Users\KIIT\OneDrive\Desktop\projects\cognizant\Cognizant-learning-program-solutions\Week3-EntityFramework Core8.0\Ret
ailInventory> dotnet ef migrations add InitialCreate
Build started...
Build succeeded.
Done. To undo this action, use 'ef migrations remove'
PS C:\Users\KIIT\OneDrive\Desktop\projects\cognizant\Cognizant-learning-program-solutions\Week3-EntityFramework Core8.0\Ret
ailInventory> dotnet ef database update
Build started...
Build succeeded.
Microsoft.Data.SqlClient.SqlException (0x80131904): A network-related or instance-specific error occurred while establishin
g a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and
that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a co
nnection to SQL Server)
 ---> System.ComponentModel.Win32Exception (2): The system cannot find the file specified.
   at Microsoft.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean breakConnection, Action`1 wrap
CloseInAction)
   at Microsoft.Data.SqlClient.TdsParser.ThrowExceptionAndWarning(TdsParserStateObject stateObj, Boolean callerHasConnectio
nLock, Boolean asyncClose)
   at Microsoft.Data.SqlClient.TdsParser.Connect(ServerInfo serverInfo, SqlInternalConnectionTds connHandler, Boolean ignor
eSniOpenTimeout, Int64 timerExpire, SqlConnectionString connectionOptions, Boolean withFailover)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds.AttemptOneLogin(ServerInfo serverInfo, String newPassword, SecureSt
ring newSecurePassword, Boolean ignoreSniOpenTimeout, TimeoutTimer timeout, Boolean withFailover)
   at Microsoft.Data.SqlClient.SqlInternalConnectionTds.LoginNoFailover(ServerInfo serverInfo, String newPassword, SecureSt
ring newSecurePassword, Boolean redirectedUserInstance, SqlConnectionString connectionOptions, SqlCredential credential, Ti
```

Verification:

- Connected to SQL Server using SSMS/Azure Data Studio.
- Verified tables: Products, Categories, and __EFMigrationsHistory were created in the RetailDb database.

Output:

✓ Database schema created and tables verified in SQL Server.



Lab 4: Inserting initial data into the Database

Objective: Insert initial product categories and products into the database using EF Core.

Steps:

- Created new Category objects: Electronics and Groceries.

- Created new Product objects: Laptop (₹75000), Rice Bag (₹1200). - Added using AddRangeAsync and saved with SaveChangesAsync.

```
Week3-EntityFramework Core8.0 > RetailInventory > Program.cs > Program > Main
1   using System.Threading.Tasks;
2   using RetailInventory;
3   using RetailInventory.Models;
4
    0 references
5   class Program
6   {
        0 references
7       static async Task Main(string[] args)
8       {
9           using var context = new AppDbContext();
10          var electronics = new Category { Name = "Electronics" };
11          var groceries = new Category { Name = "Groceries" };
12          await context.Categories.AddRangeAsync(electronics, groceries);
13          var product1 = new Product { Name = "Laptop", Price = 75000, Category = electronics };
14          var product2 = new Product { Name = "Rice Bag", Price = 1200, Category = groceries };
15          await context.Products.AddRangeAsync(product1, product2);
16          await context.SaveChangesAsync();
17          Console.WriteLine("Data inserted successfully.");
18      }
19  }
```

Output:

✓ Inserted: Electronics, Groceries, Laptop, Rice Bag.

```
SQLQuery1.sql - lo...-2206098\KIIT (69))*
USE RetailDb;

SELECT * FROM Categories;
SELECT * FROM Products;
```

100 %

Results | Messages

	Id	Name
1	1	Electronics
2	2	Groceries

	Id	Name	Price	CategoryId
1	1	Laptop	75000.00	1
2	2	Rice Bag	1200.00	2

Query executed successfully. | localhost\SQLEXPRESS (16.0 ... | BT-2206098\KIIT (69) | RetailDb | 00:00:00 | 4 rows

Lab 5: Retrieving Data from the Database

Objective: Use LINQ and EF Core methods to retrieve data from the database.

Steps:

1. Retrieve all products using ToListAsync().

2. Find product by ID using FindAsync(1).

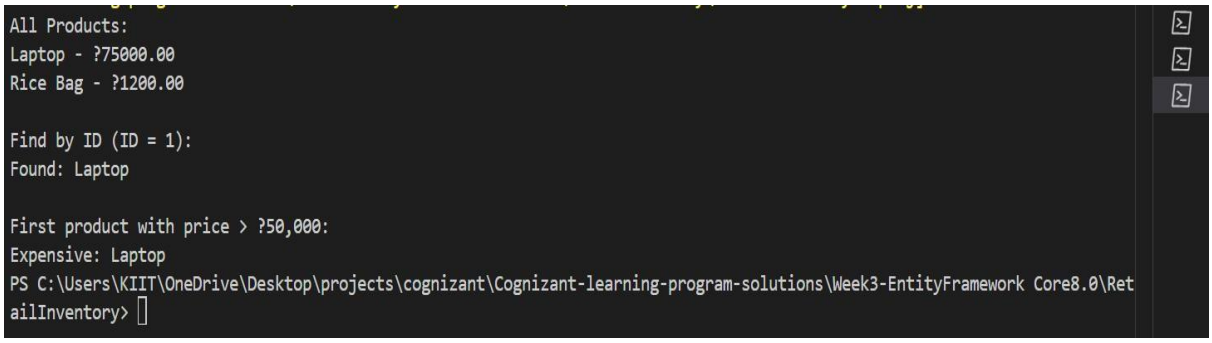
3. Find first product with Price > ₹50000 using FirstOrDefaultAsync. **Code Sample:**

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

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

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

Output:



Lab 6: Updating and Deleting Records

Objective: Update and delete records in the database using EF Core.

Steps:

- Updated Laptop price to ₹70000 using FirstOrDefaultAsync and SaveChangesAsync.
- Deleted 'Rice Bag' using Remove() and SaveChangesAsync.

Code Sample:

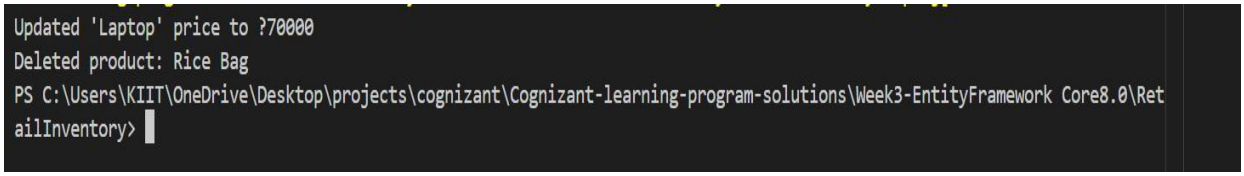
var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");

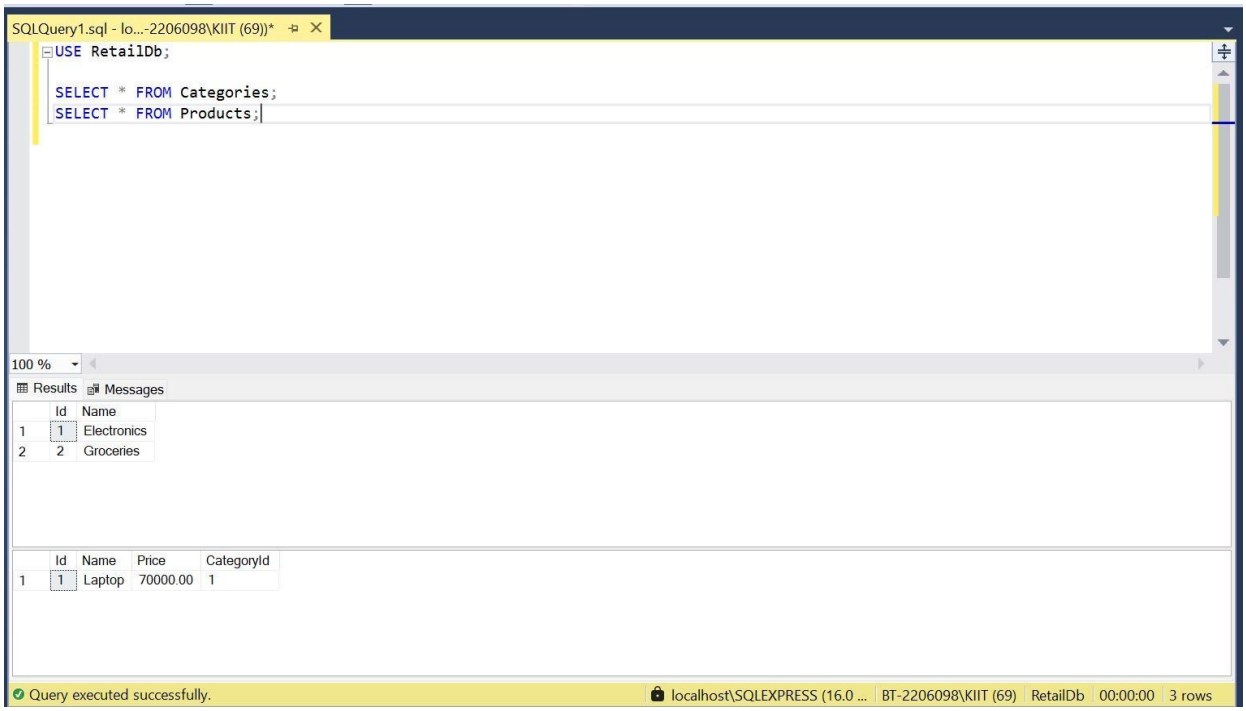
product.Price = 70000; await context.SaveChangesAsync();

var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Rice Bag");

context.Products.Remove(toDelete); await context.SaveChangesAsync();

Output:





Lab 7: Writing Queries with LINQ

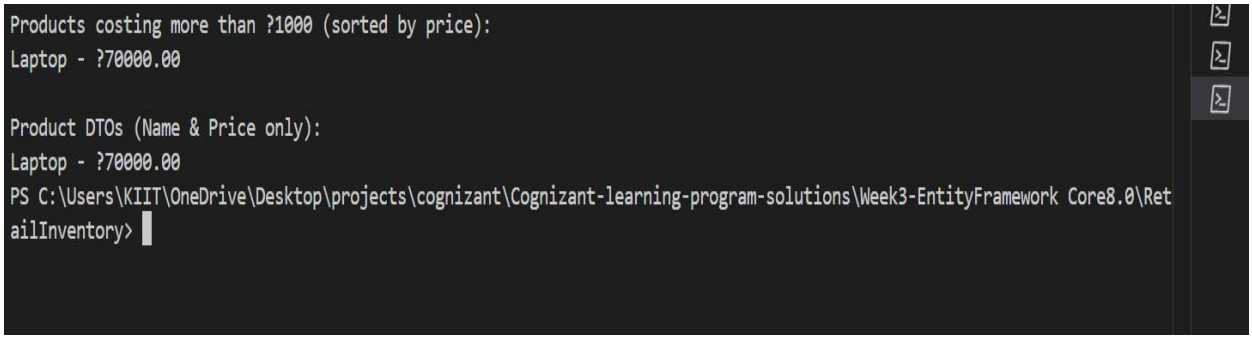
Objective: Use LINQ to filter, sort, and project product data into lightweight DTOs.

Steps:

1. Filter products with Price > ₹1000 and sort by Price descending.
2. Use Select to return only Name and Price as anonymous objects. **Code Sample:**

var filtered = await context.Products.Where(p => p.Price > 1000).OrderByDescending(p => p.Price).ToListAsync(); var productDTOs = await context.Products.Select(p => new { p.Name, p.Price }).ToListAsync();

Output:


Products costing more than ?1000 (sorted by price):
Laptop - ?70000.00
Product DTOs (Name & Price only):
Laptop - ?70000.00
PS C:\Users\KIIT\OneDrive\Desktop\projects\cognizant\Cognizant-learning-program-solutions\Week3-EntityFramework Core8.0\RetailInventory>