**DN 4.0 Dotnet FSE**

**Name: Sparsh Guha**

**Superset ID:6361106**

**Week 3**

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

Answers:

1. **What is ORM?**  
   ORM (Object-Relational Mapping) is a technique that lets you interact with a database using C# classes instead of SQL queries.

**How it works:**  
Each C# class maps to a table in the database.  
Each property maps to a column.  
Each object instance maps to a row.

**Benefits:**  
Productivity: Write less SQL, focus on C# code.  
Maintainability: Changes in code reflect in the database model.  
Abstraction: You don’t need to write raw SQL for most operations.

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

• EF Core is cross-platform, lightweight, and supports modern features like

LINQ, async queries, and compiled queries.

• EF Framework (EF6) is Windows-only and more mature but less flexible.

**3. EF Core 8.0 Features**  
JSON column mapping: Store and query JSON data directly in columns.  
Compiled models: Faster startup and query execution.  
Interceptors: Customize and monitor database operations.  
Bulk operations: Improved performance for batch updates/inserts.

1. **Create a .NET Console App**  
    Open VS Code terminal and run:  
   dotnet new console -n RetailInventory  
   cd RetailInventory
2. **Install EF Core Packages**  
   dotnet add package Microsoft.EntityFrameworkCore.SqlServer  
   dotnet add package Microsoft.EntityFrameworkCore.Design

**Models.cs**

**Code:**

using Microsoft.EntityFrameworkCore;

public class Category

{

public int CategoryId { get; set; }

public string Name { get; set; } = "";

public List<Product> Products { get; set; } = new();

}

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; }

}

public class InventoryContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder options)

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

}

**Create the Database (Migration)**  
In terminal, run:  
  
dotnet tool install --global dotnet-ef  
dotnet ef migrations add InitialCreate  
dotnet ef database update

**Program.cs**

**Code**:

using System;

using System.Linq;

class Program

{

static void Main()

{

using var db = new InventoryContext();

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

db.Categories.Add(category);

db.Products.Add(new Product { Name = "Laptop", Stock = 10, Category = category });

db.SaveChanges();

var products = db.Products

.Select(p => new { p.Name, p.Stock, Category = p.Category!.Name })

.ToList();

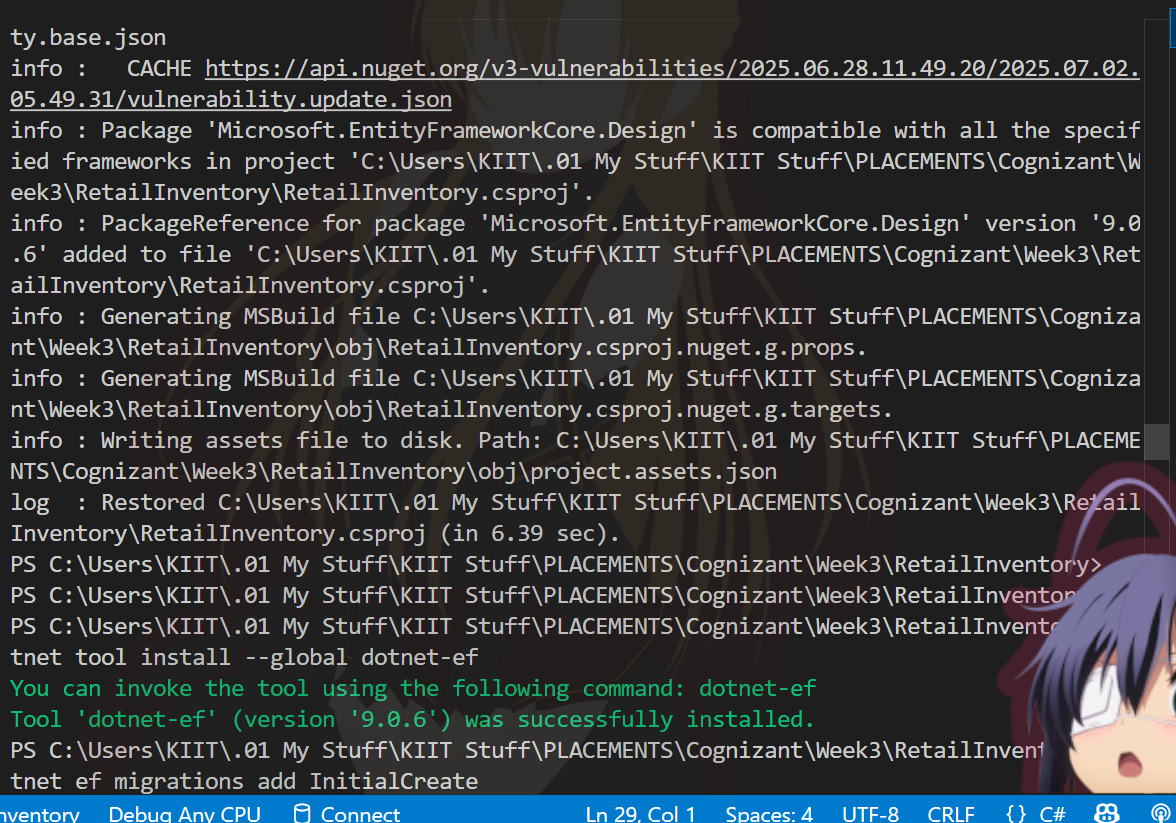
foreach (var p in products)

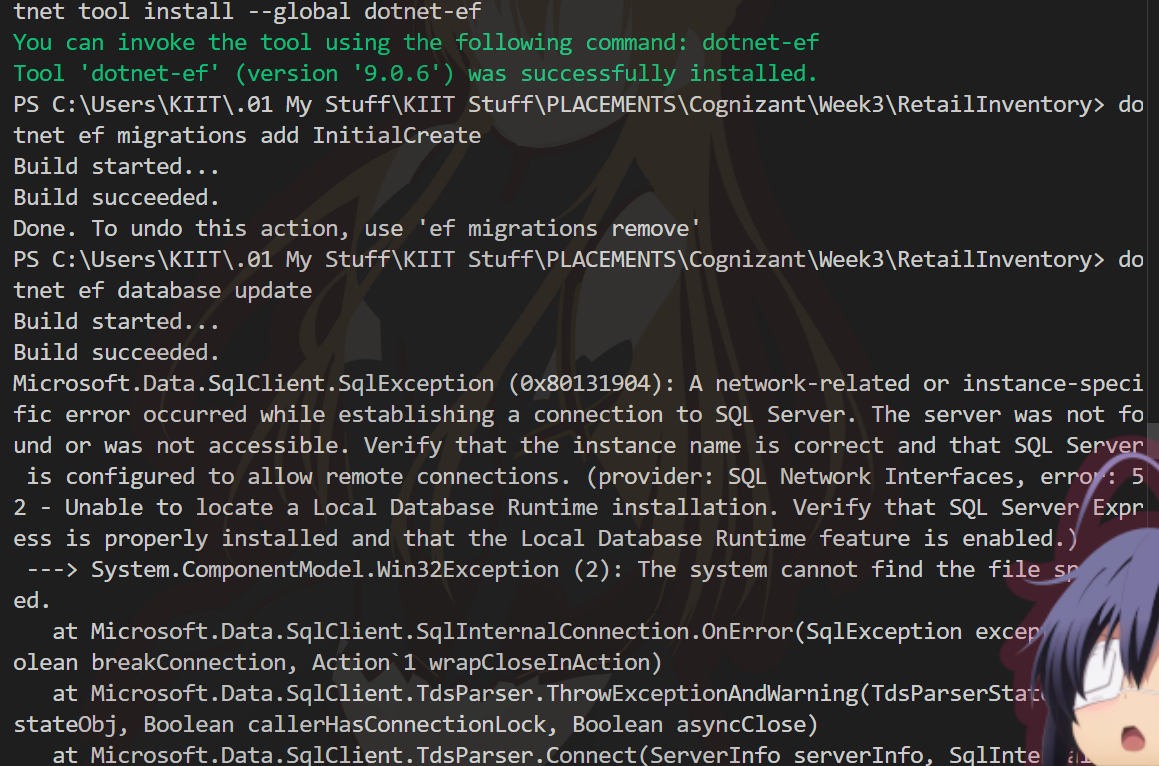
Console.WriteLine($"{p.Name} ({p.Category}): Stock={p.Stock}");

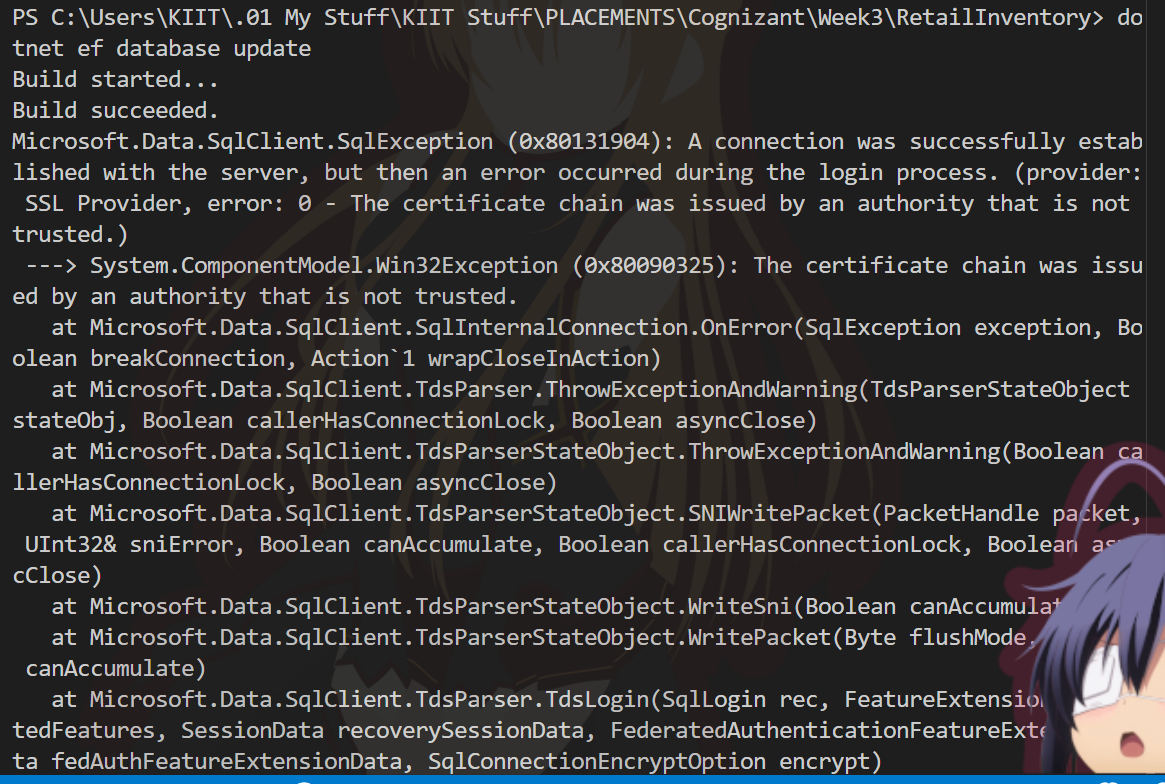
}

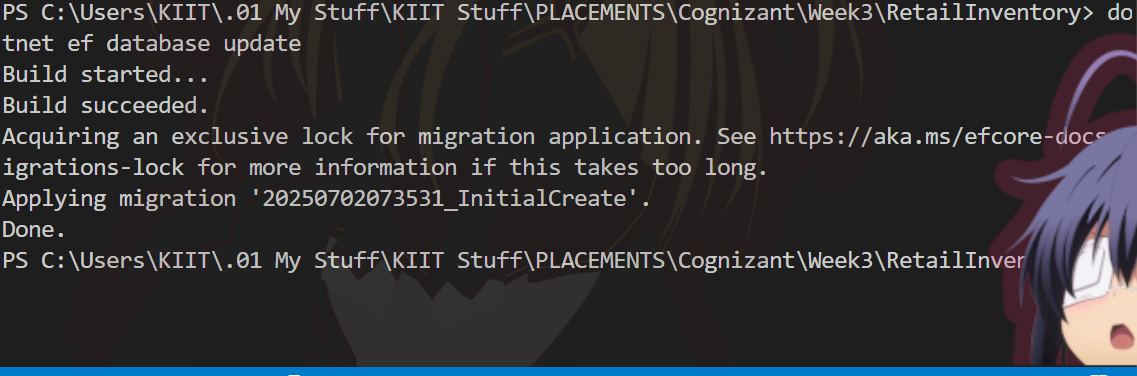
}

**Output:**

****

****

****

****

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

**Code:**

**Models.cs**

public class Category

{

public int Id { get; set; }

public string Name { get; set; } = string.Empty;

public List<Product> Products { get; set; } = new();

}

public class Product

{

public int Id { get; set; }

public string Name { get; set; } = string.Empty;

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category? Category { 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\\SQLEXPRESS;Database=RetailInventoryDb;Trusted\_Connection=True;TrustServerCertificate=True;");

}

}

**Program.cs**

using System;

using System.Linq;

class Program

{

static void Main()

{

using var db = new AppDbContext();

if (!db.Categories.Any())

{

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

db.Categories.Add(category);

db.Products.Add(new Product { Name = "Rice", Price = 50.0m, Category = category });

db.SaveChanges();

}

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

foreach (var p in products)

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

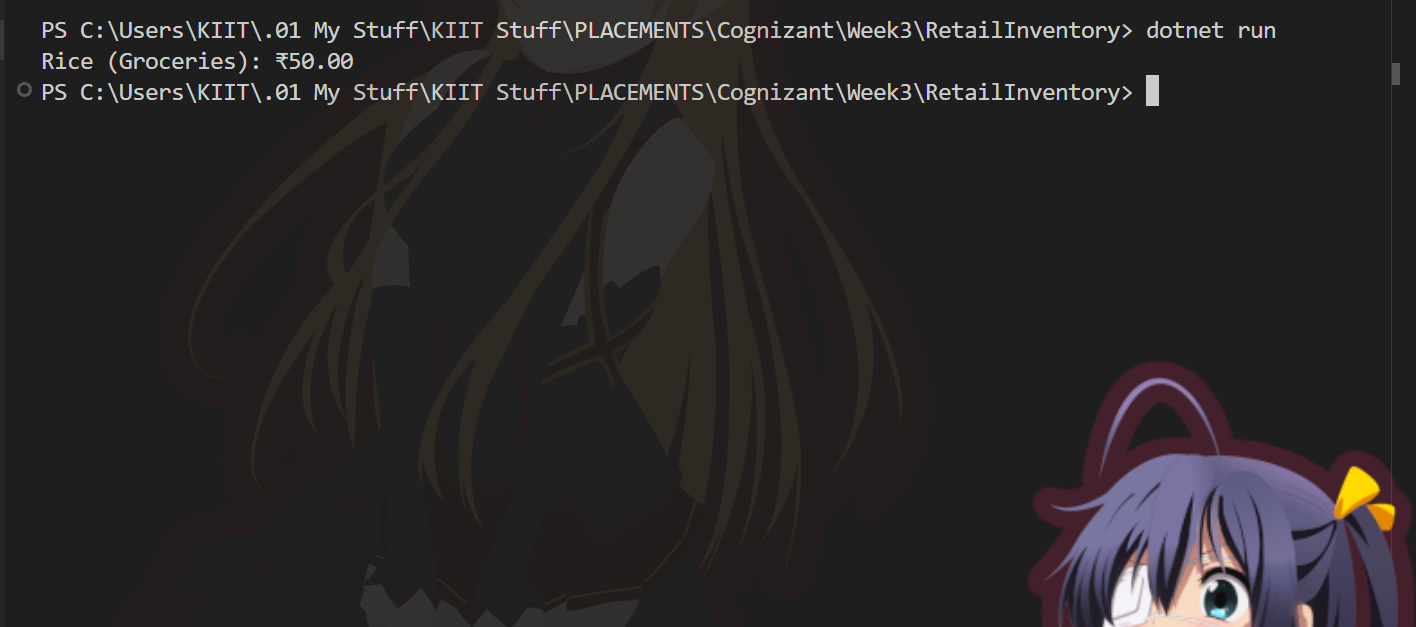
}

}

**In terminal:**  
  
dotnet ef migrations add UpdateModels  
dotnet ef database update

**To run the Application**  
dotnet run

**OUTPUT:**

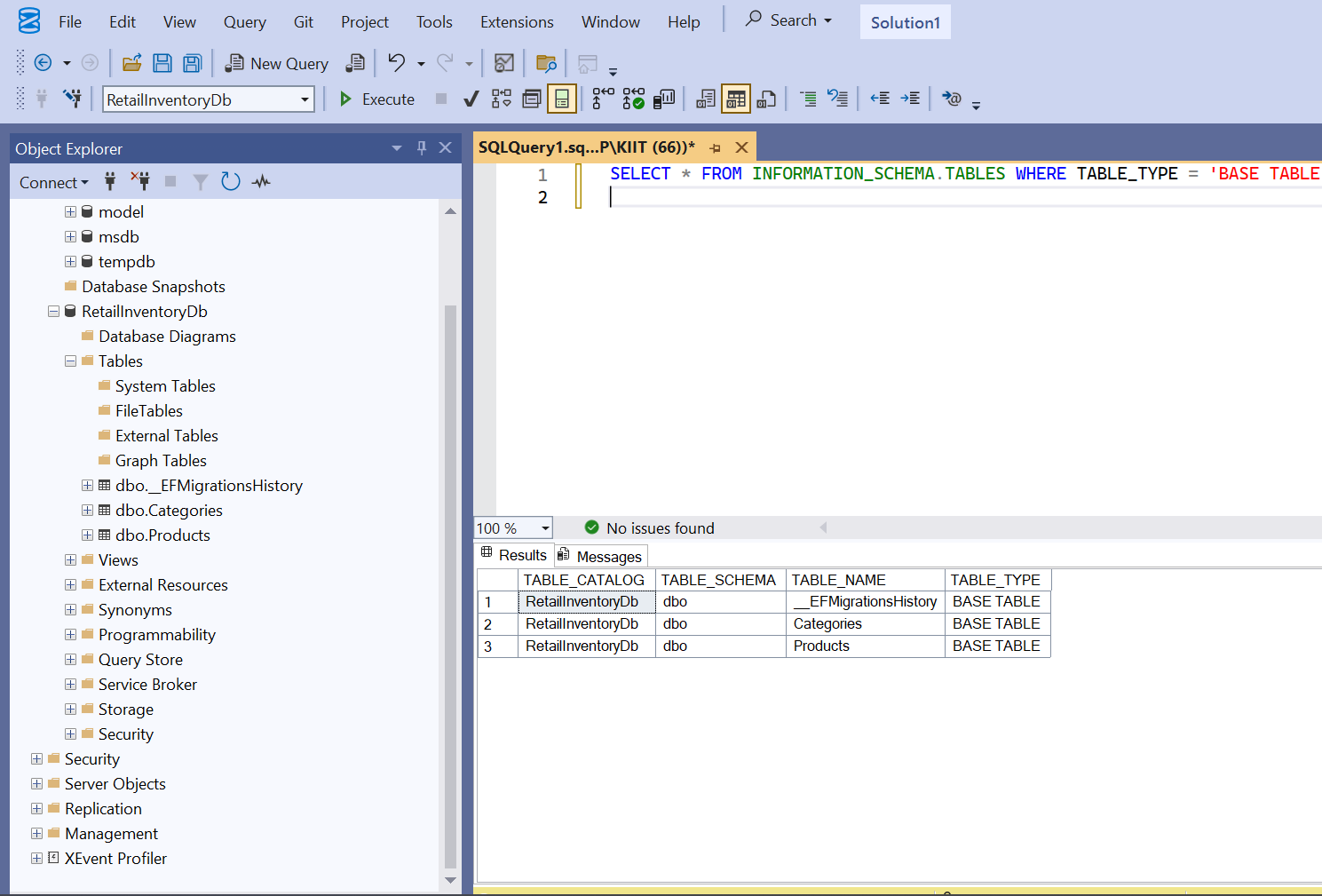


**Lab 3: Using EF Core CLI to Create and Apply Migrations**

**Input:**  
**Code:**  
1**. Create Initial Migration (From RetailInventory project directory:)**  
dotnet ef migrations add InitialCreate  
  
2**. Apply the Migration to Create/Update the Database**  
dotnet ef database update  
  
3**. Verify in SQL Server**  
Open SQL Server Management Studio (SSMS)  
Connect to SQLEXPRESS  
Expand the RetailInventoryDb database.  
Expand the Tables node to confirm that Products and Categories tables exist.  
4**. List Tables via SQL**  
Running this SQL in a query window to list all tables:

SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_TYPE = 'BASE TABLE';

**Output:**

****

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

**Code:**

**Program.cs**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main()

{

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);

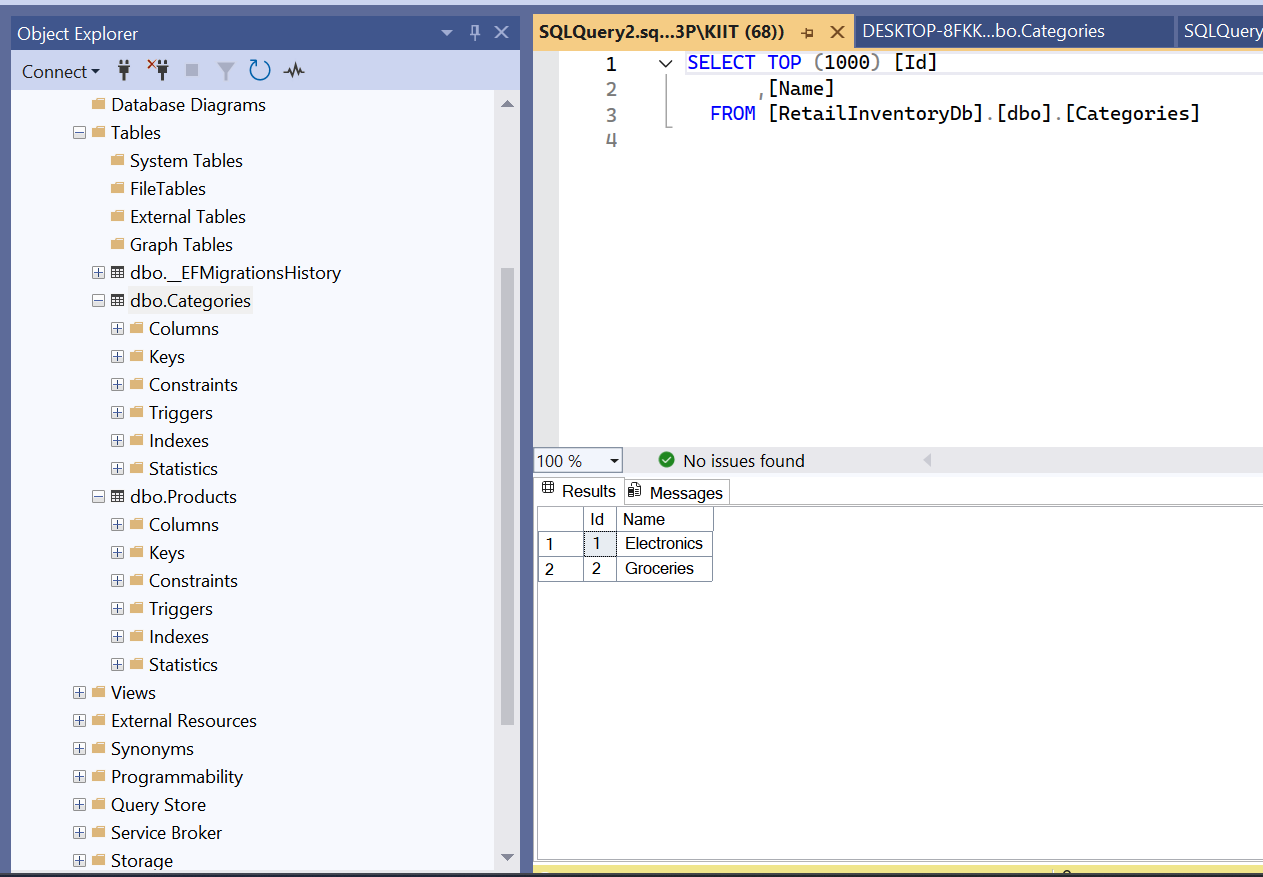
await context.SaveChangesAsync();

}

}

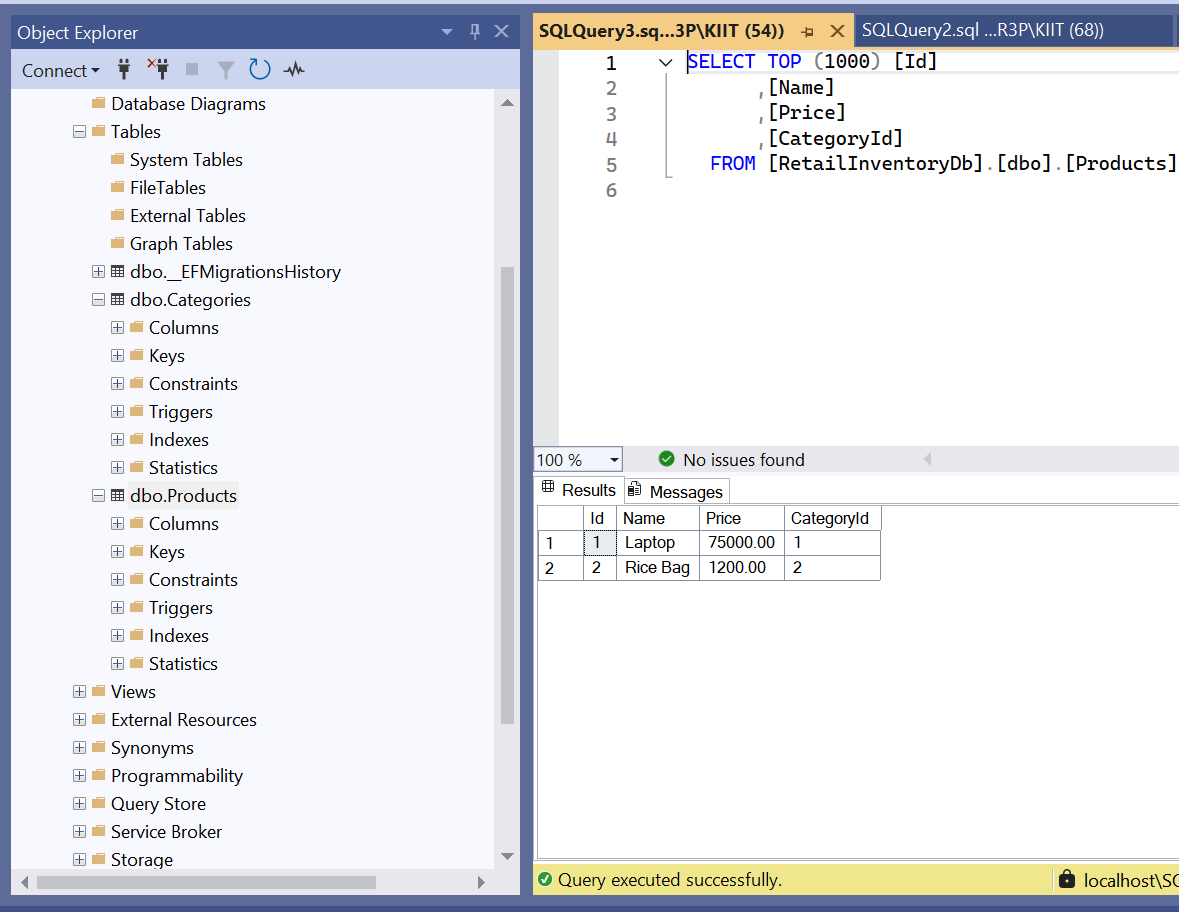
**2. Running Application**  
In terminal:  
dotnet run  
  
**3. Verify in SQL Server (OUTPUT INCLUDED)**  
Open SSMS  
Connect to localhost\SQLEXPRESS, open RetailInventoryDb database.  
Right-click the Categories table > Select Top 1000 Rows.

Categories: Electronics, Groceries



**4.**  
Right-click the Products table > Select Top 1000 Rows.

Products: Laptop (Electronics), Rice Bag (Groceries)



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

**Code:**

**Program.cs**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

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

foreach (var p in products)

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

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

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

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

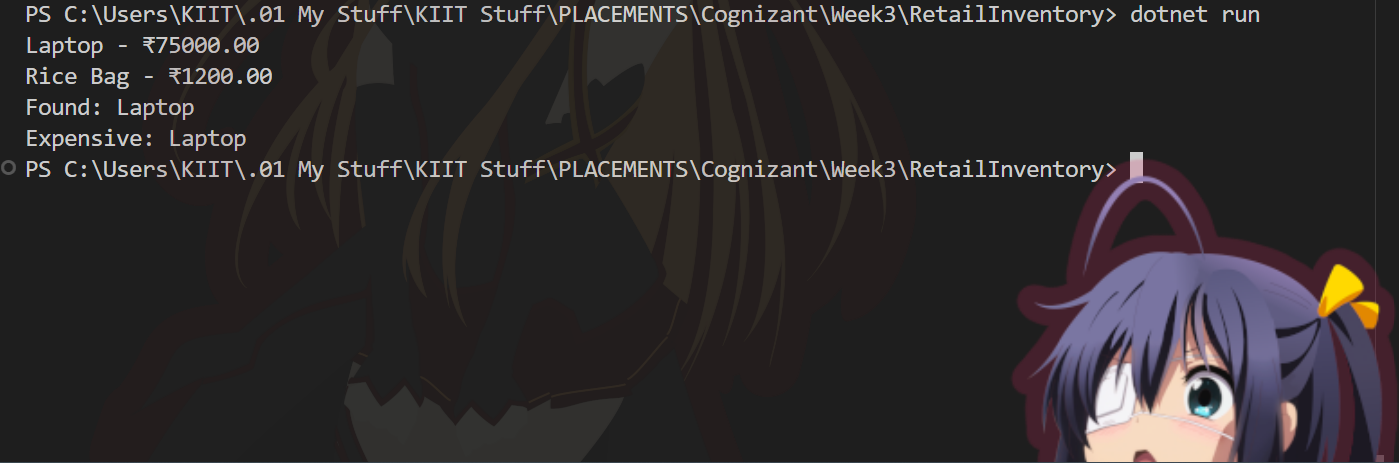
Console.WriteLine($"Expensive: {expensive?.Name}");

}

}

1. **Running Application**  
   In terminal:  
   dotnet run

**Output:**

****

**ADDITIONAL QUESTIONS:**

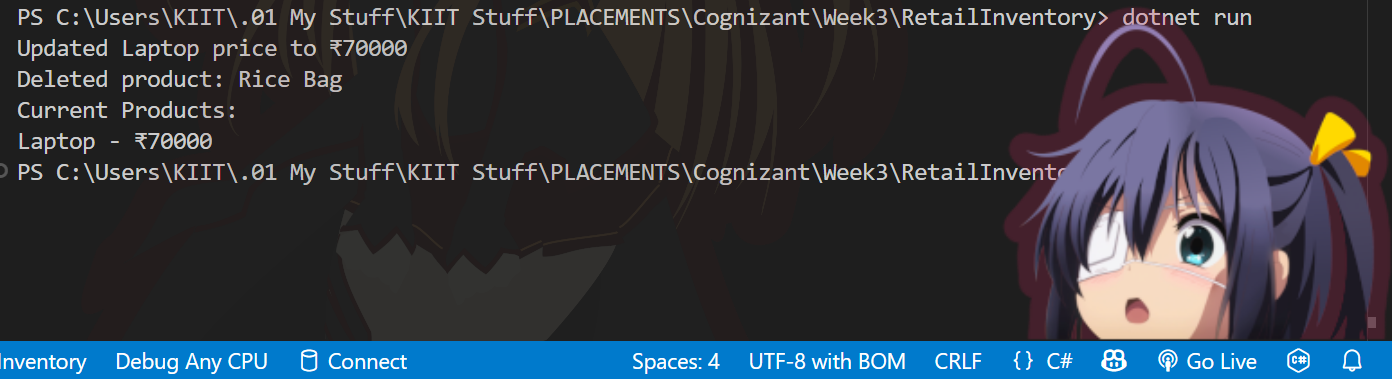
**Lab 6: Updating and Deleting Records**

**Code:**

**Program.cs**using System;  
using System.Threading.Tasks;  
using Microsoft.EntityFrameworkCore;  
  
class Program  
{  
static async Task Main()  
{  
using var context = new AppDbContext();  
  
var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");  
if (product != null)  
{  
product.Price = 70000;  
await context.SaveChangesAsync();  
Console.WriteLine($"Updated {product.Name} price to ₹{product.Price}");  
}  
  
var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Rice Bag");  
if (toDelete != null)  
{  
context.Products.Remove(toDelete);  
await context.SaveChangesAsync();  
Console.WriteLine($"Deleted product: {toDelete.Name}");  
}  
  
var products = await context.Products.ToListAsync();  
Console.WriteLine("Current Products:");  
foreach (var p in products)  
Console.WriteLine($"{p.Name} - ₹{p.Price}");  
}  
}

**Running Application**  
In terminal:  
dotnet run

**Output:**

****

**Lab 7: Writing Queries with LINQ**

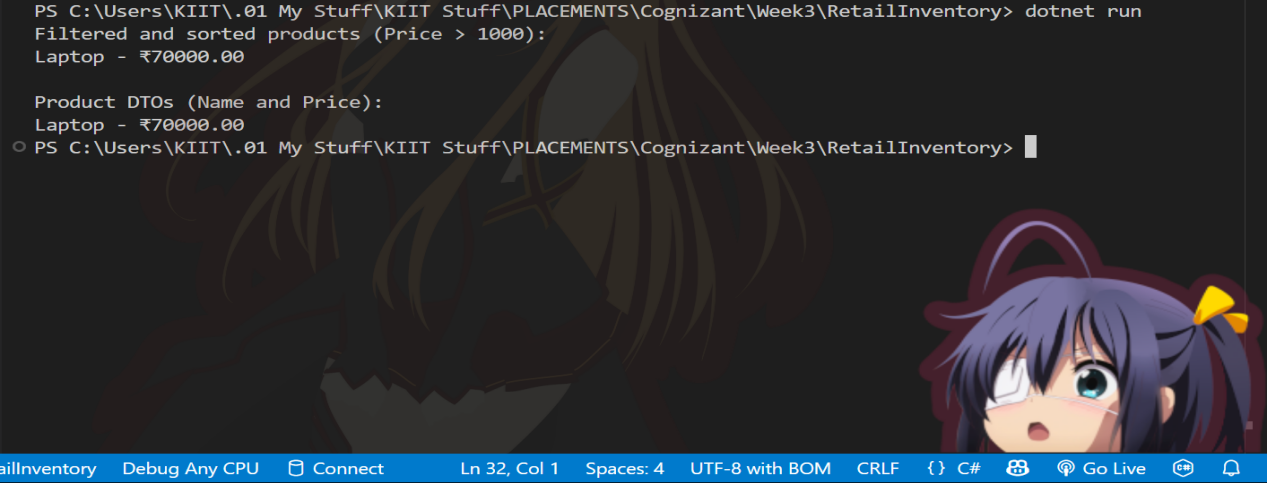
**Code:**

**Program.cs**

using System;  
using System.Threading.Tasks;  
using System.Linq;  
using Microsoft.EntityFrameworkCore;  
  
class Program  
{  
static async Task Main()  
{  
using var context = new AppDbContext();  
  
var filtered = await context.Products  
.Where(p => p.Price > 1000)  
.OrderByDescending(p => p.Price)  
.ToListAsync();  
  
Console.WriteLine("Filtered and sorted products (Price > 1000):");  
foreach (var p in filtered)  
Console.WriteLine($"{p.Name} - ₹{p.Price}");  
  
var productDTOs = await context.Products  
.Select(p => new { p.Name, p.Price })  
.ToListAsync();  
  
Console.WriteLine("\nProduct DTOs (Name and Price):");  
foreach (var dto in productDTOs)  
Console.WriteLine($"{dto.Name} - ₹{dto.Price}");  
}  
}

**Running Application**  
In terminal:  
dotnet run

**OUTPUT:**

****