**-----------------------------------------------------------------------------------------**

**Lab 1: Understanding ORM with a Retail Inventory System  
  
1. What is ORM?**

**• Explain how ORM maps C# classes to database tables.**

**ORM**  is a technique that connects database tables with C# objects. In EF Core,

A class like Product maps to a Products table.

Properties like Name, Price map to table columns.

### **• Benefits: Productivity, maintainability and abstraction from SQL. **Benefits of ORM:****

**Productivity**: Write less code. Focus on business logic.

**Maintainability**: Centralized model classes. Easier updates.

**Abstraction**: No need to write raw SQL queries.

### **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.**

| **Feature** | **EF Core** | **EF Framework (EF6)** |
| --- | --- | --- |
| Platform | Cross-platform (.NET Core) | Windows only |
| Performance | Lightweight, fast, modern | Slower but stable |
| Features | LINQ, async, compiled queries | Mature but older tech |
| JSON Support | JSON columns in EF Core 8.0 | Not supported |

### **EF Core 8.0 Features: • JSON column mapping. • Improved performance with compiled models. • Interceptors and better bulk operations.**

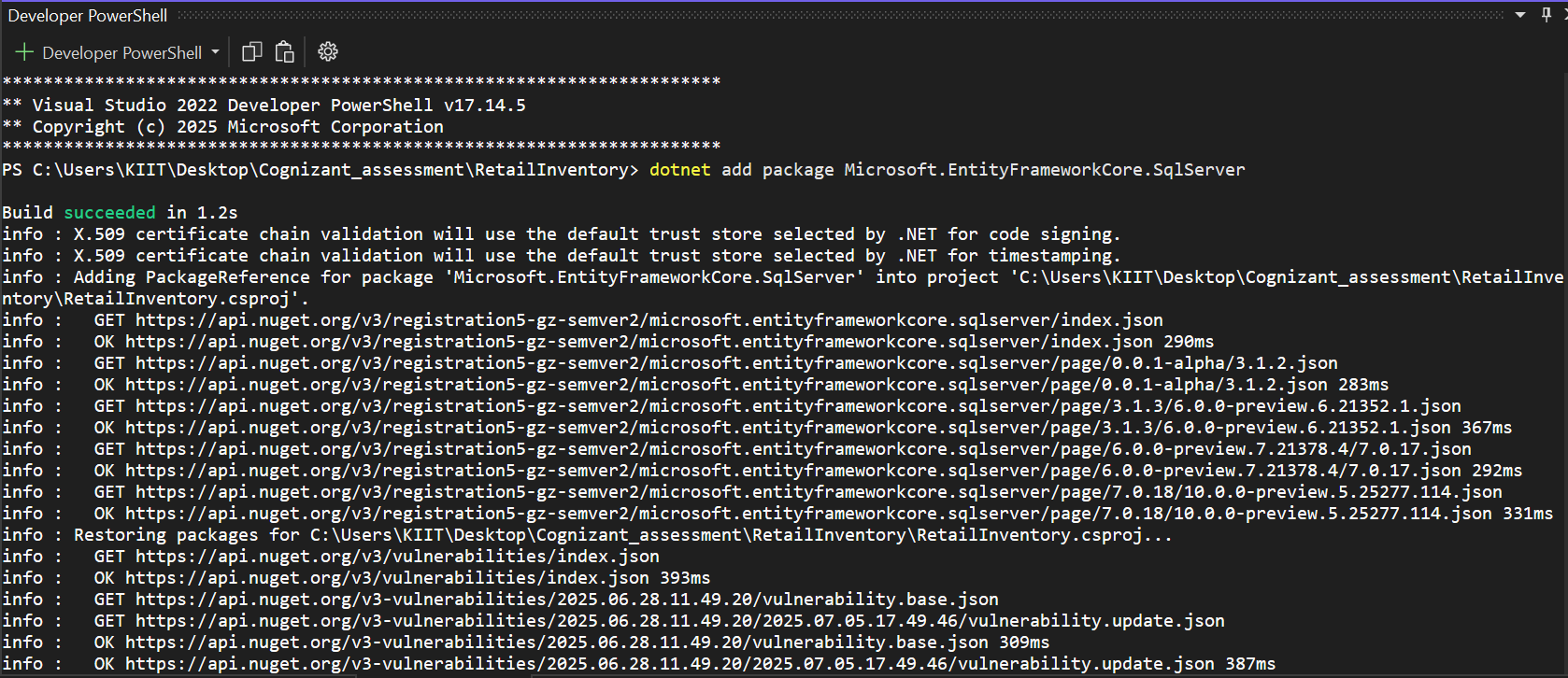
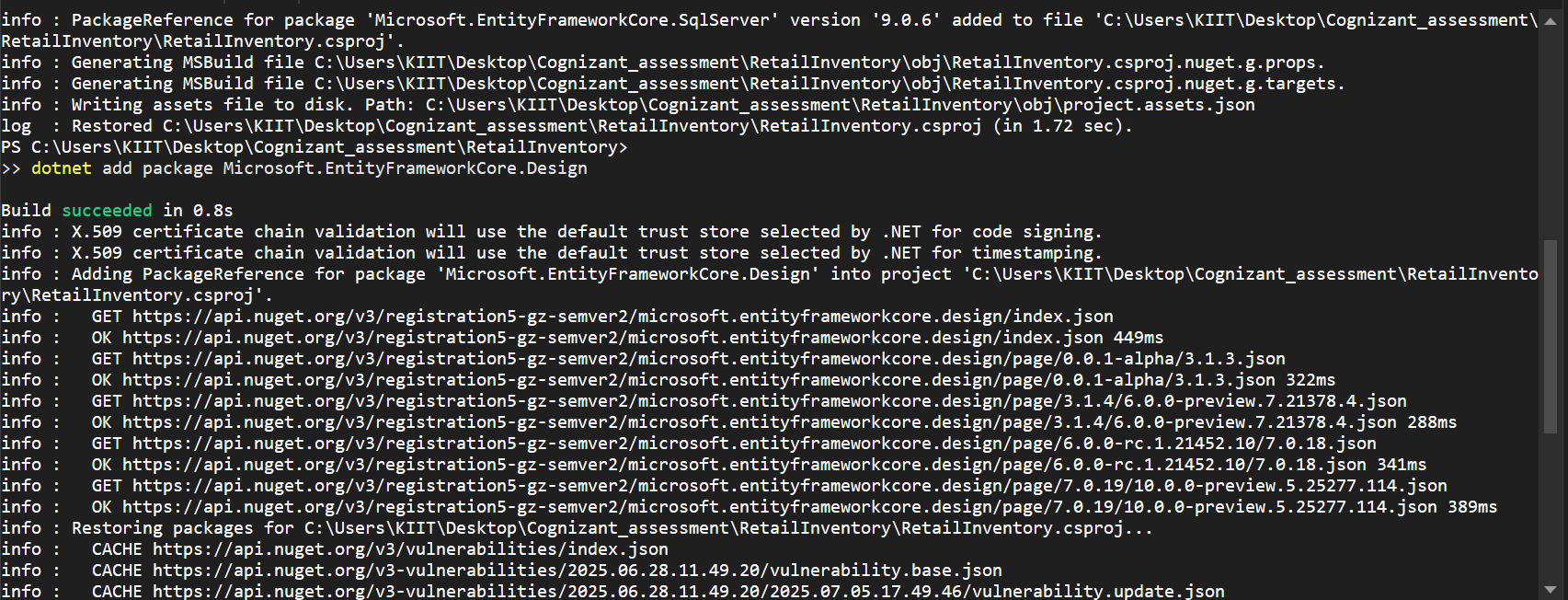
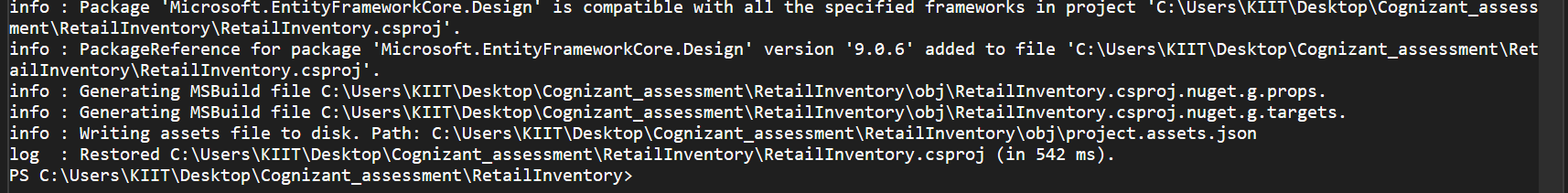
**JSON Column Mapping** – You can store JSON in SQL columns.

**Compiled Models** – Greatly improves startup and runtime performance.

**Bulk Updates/Deletes** – Much faster operations.

**Interceptors** – Hook into EF Core operations for logging, caching, etc.

**Installation:-** dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Design  
  
  
**Lab 2: Setting Up the Database Context for a Retail Store  
  
Model.cs**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; }

}

**AppDbContext.cs**

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using System.IO;

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

var config = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json")

.Build();

var connStr = config.GetConnectionString("RetailInventoryConnection");

optionsBuilder.UseSqlServer(connStr);

}

}

|  |
| --- |
| **NOTE:- *Installed “dotnet add package Microsoft.Extensions.Configuration.Json” for Configuration.*** |

**Lab 3: Using EF Core CLI to Create and Apply Migrations  
  
1.Install EF Core  
2.Create Initial Migration  
3.Apply Migration to Create Database  
  
PS C:\Users\KIIT\Desktop\Cognizant\_assessment\RetailInventory> dotnet tool install --global dotnet-ef**

**>>**

**Tool 'dotnet-ef' is already installed.**

**PS C:\Users\KIIT\Desktop\Cognizant\_assessment\RetailInventory> dotnet ef migrations add InitialCreate**

**>>**

**Build started...**

**Build succeeded.**

**Done. To undo this action, use 'ef migrations remove'**

**PS C:\Users\KIIT\Desktop\Cognizant\_assessment\RetailInventory> dotnet ef database update**

**>>**

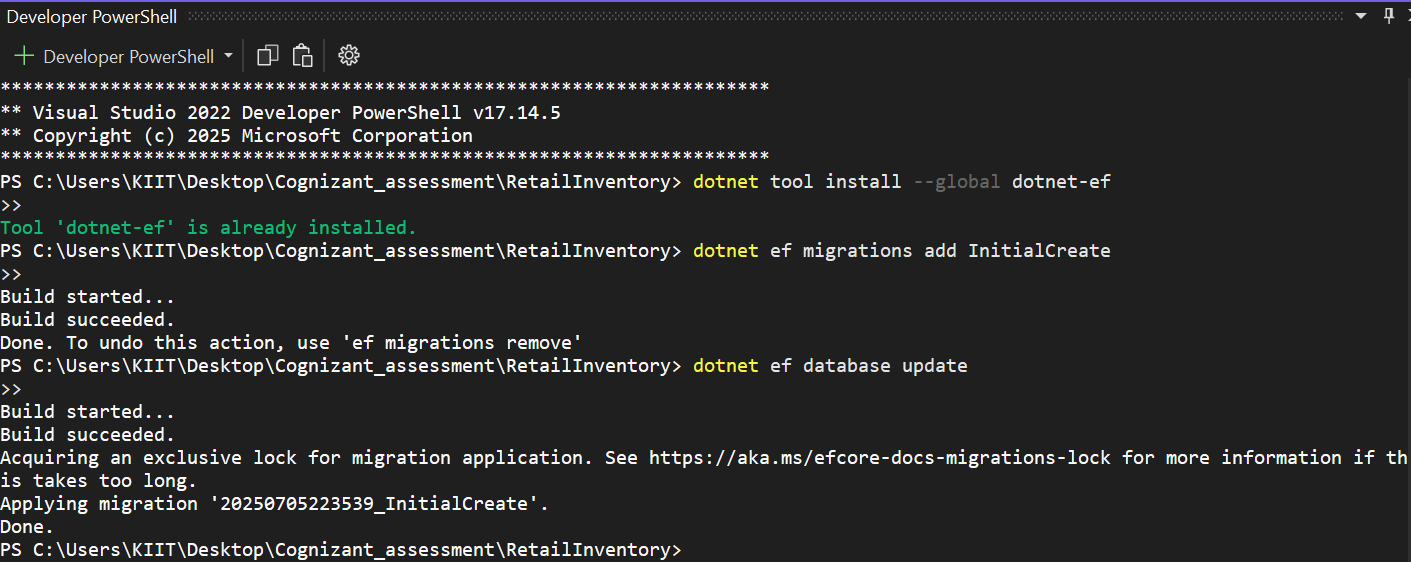
**Build started...**

**Build succeeded.**

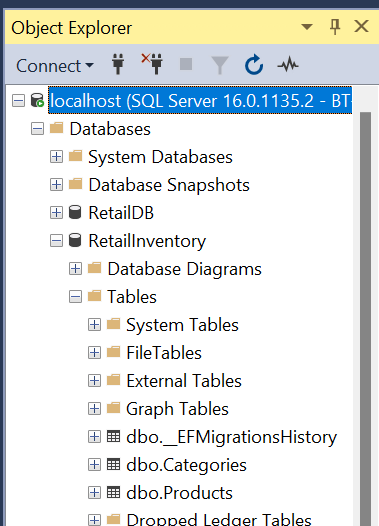
**Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.**

**Applying migration '20250705223539\_InitialCreate'.**

**Done.**



**Verify in SQL Server:**

**Databases --> RetailInventory -->Tables**

**-----------------------------------------------------------------------------------------**

**Lab 4: Inserting Initial Data into the Database   
  
Program.cs**  
  
using System;

using System.Threading.Tasks;

class Program

{

static async Task Main(string[] args)

{

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 };

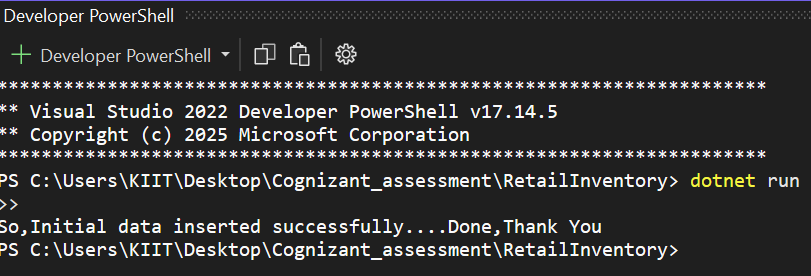
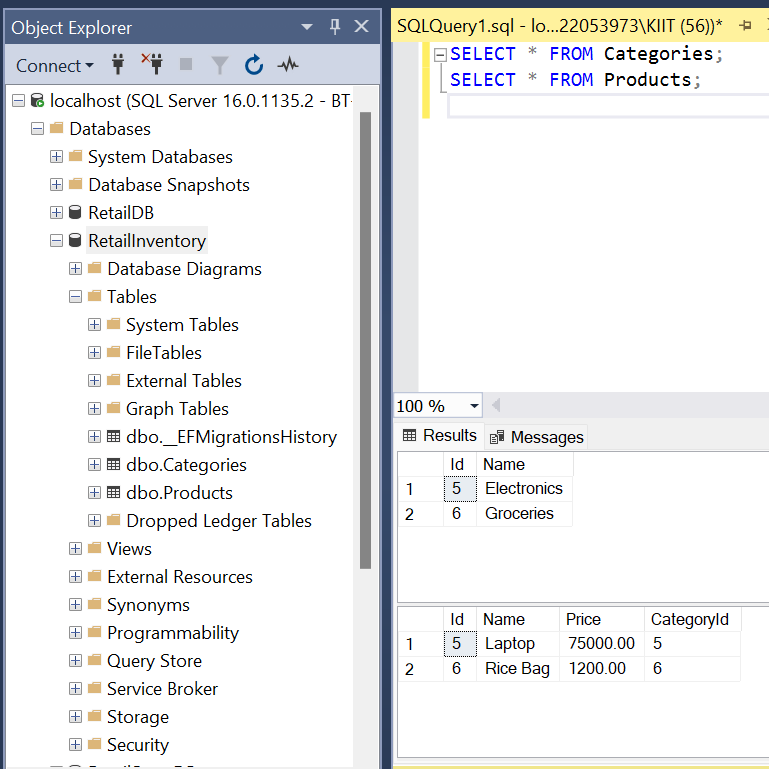
await context.Products.AddRangeAsync(product1, product2);

await context.SaveChangesAsync();

Console.WriteLine("So,Initial data inserted successfully....Done,Thank You");

}

}

**Lab 5: Retrieving Data from the Database  
  
Program.cs**using System;

using System.Threading.Tasks;

using System.Linq;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

Console.WriteLine("🔹 All Products:");

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

foreach (var p in products)

{

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

}

Console.WriteLine("\n🔹 Find Product By ID:");

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

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

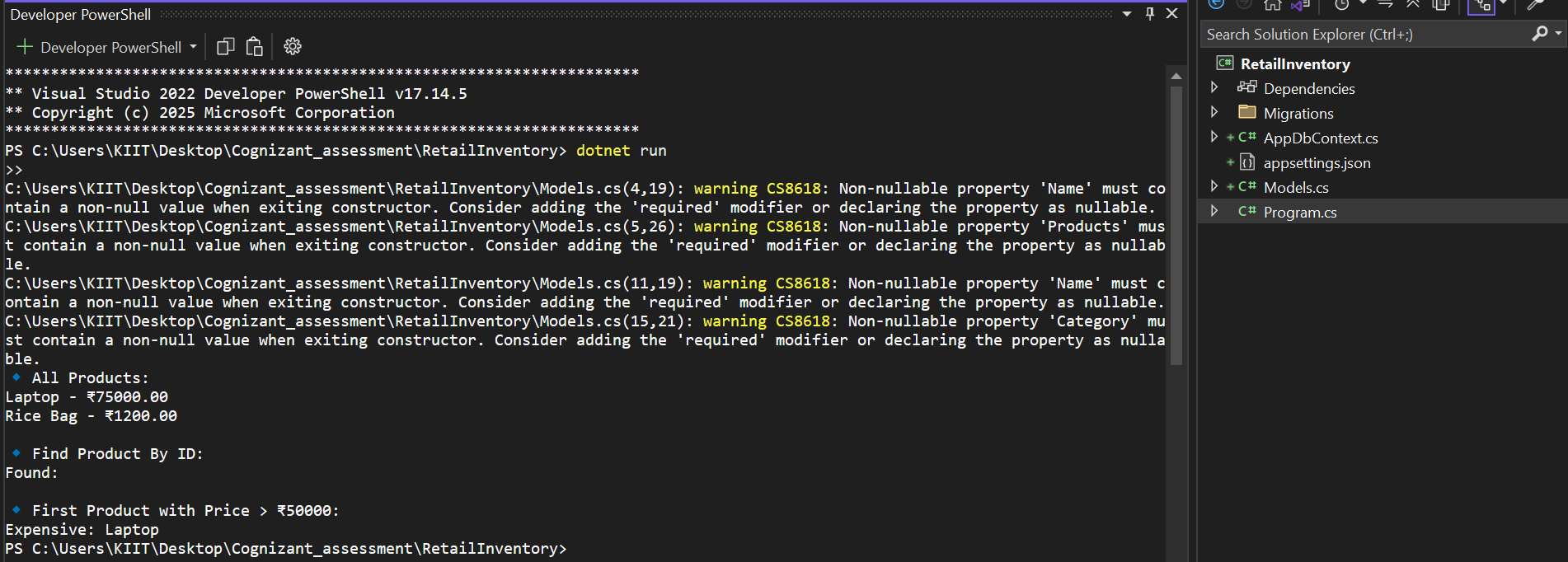
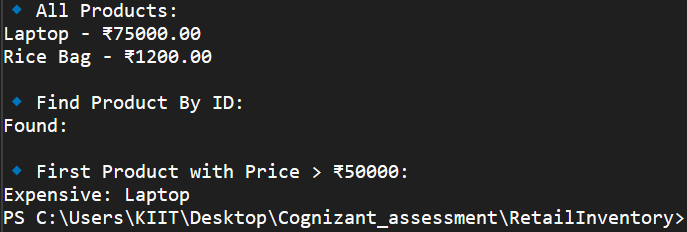
Console.WriteLine("\n🔹 First Product with Price > ₹50000:");

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

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

}

}

  
  
  
  
  
  
 ***Additional Hands On:-*Lab 6: Updating and Deleting Records   
  
Program.cs**  
using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

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}'s price to ₹{product.Price}");

}

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

if (toDelete != null)

{

context.Products.Remove(toDelete);

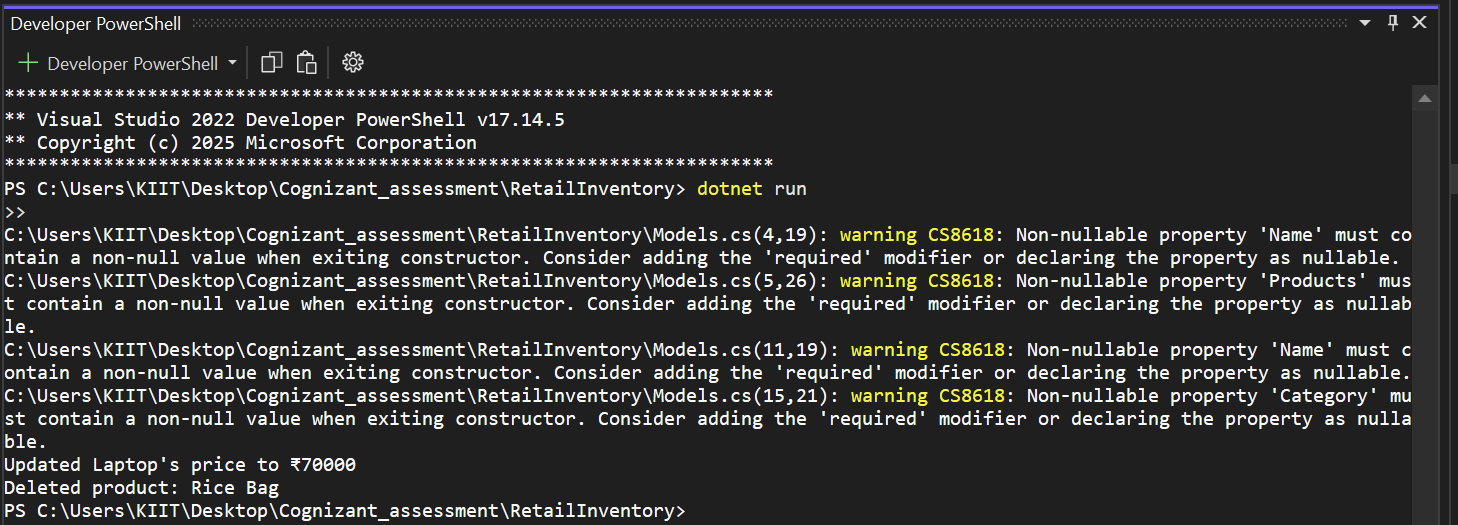
await context.SaveChangesAsync();

Console.WriteLine($"Deleted product: {toDelete.Name}");

}

}

}



**Lab 7: Writing Queries with LINQ**

**Program.cs**  
using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

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 & Price Only):");

foreach (var dto in productDTOs)

{

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

}

}

}

|  |
| --- |
| ***Output--> Next Page*** |

