Lab 1: Understanding ORM with a Retail Inventory System

1. **What is ORM?**

Object Relational Mapping (ORM) is a technique that maps C# classes (objects) to relational database tables. This allows you to use LINQ instead of SQL and eliminates boilerplate ADO.NET code.

**Key Benefits:**

* Productivity: Less SQL, faster development
* Maintainability: Centralized models, easy to update
* Abstraction: Code-first approach hides DB details

**2. EF Core vs EF Framework**

Feature EF Core (Latest) EF Framework (EF6)

Platform Cross-platform (.NET 6/7/8) Windows-only (.NET Framework)

Performance Faster, optimized queries Slower

Features LINQ, async, compiled models, interceptors Mature but rigid

Use Case New development Legacy apps

**3. EF Core 8.0 New Features:**

* JSON Column Mapping
* Compiled Models for Better Performance
* Interceptors and Logging Improvements
* Enhanced Bulk Updates
* Improved LINQ translation

**PROJECT STRUCTURE**

RetailInventory is a .NET Console App built using EF Core 9.0 with a simple inventory management system. The project contains:

Models/: Contains entity classes Category.cs and Product.cs.

AppDbContext.cs: Configures EF Core and defines the DbSet properties.

Program.cs: The main entry point of the app that seeds and displays data.

Migrations/: Contains auto-generated EF migration files.

RetailInventory.csproj: Project file with EF Core dependencies.

**CODE:**

*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; }

}

*AppDbContext .cs*

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer(@"Server=localhost;Database=RetailDb;Trusted\_Connection=True;Encrypt=False;");

}

}

*Program .cs*

using System;

using System.Linq;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

class Program

{

static void Main(string[] args)

{

using (var context = new AppDbContext()

)

{

if (context.Categories.Any())

{

context.Categories.RemoveRange(context.Categories);

context.Products.RemoveRange(context.Products);

context.SaveChanges();

}

var electronics = new Category

{

CategoryName = "Electronics"

};

context.Categories.Add(electronics);

context.SaveChanges();

var product = new Product

{

Name = "Tablets",

Stock = 100,

CategoryId = electronics.CategoryId

};

context.Products.Add(product);

context.SaveChanges();

var products = context.Products.ToList();

Console.WriteLine("\n--- Product List ---");

foreach (var p in products)

{

Console.WriteLine($"Product: {p.Name}, Stock: {p.Stock}, Category ID: {p.CategoryId}");

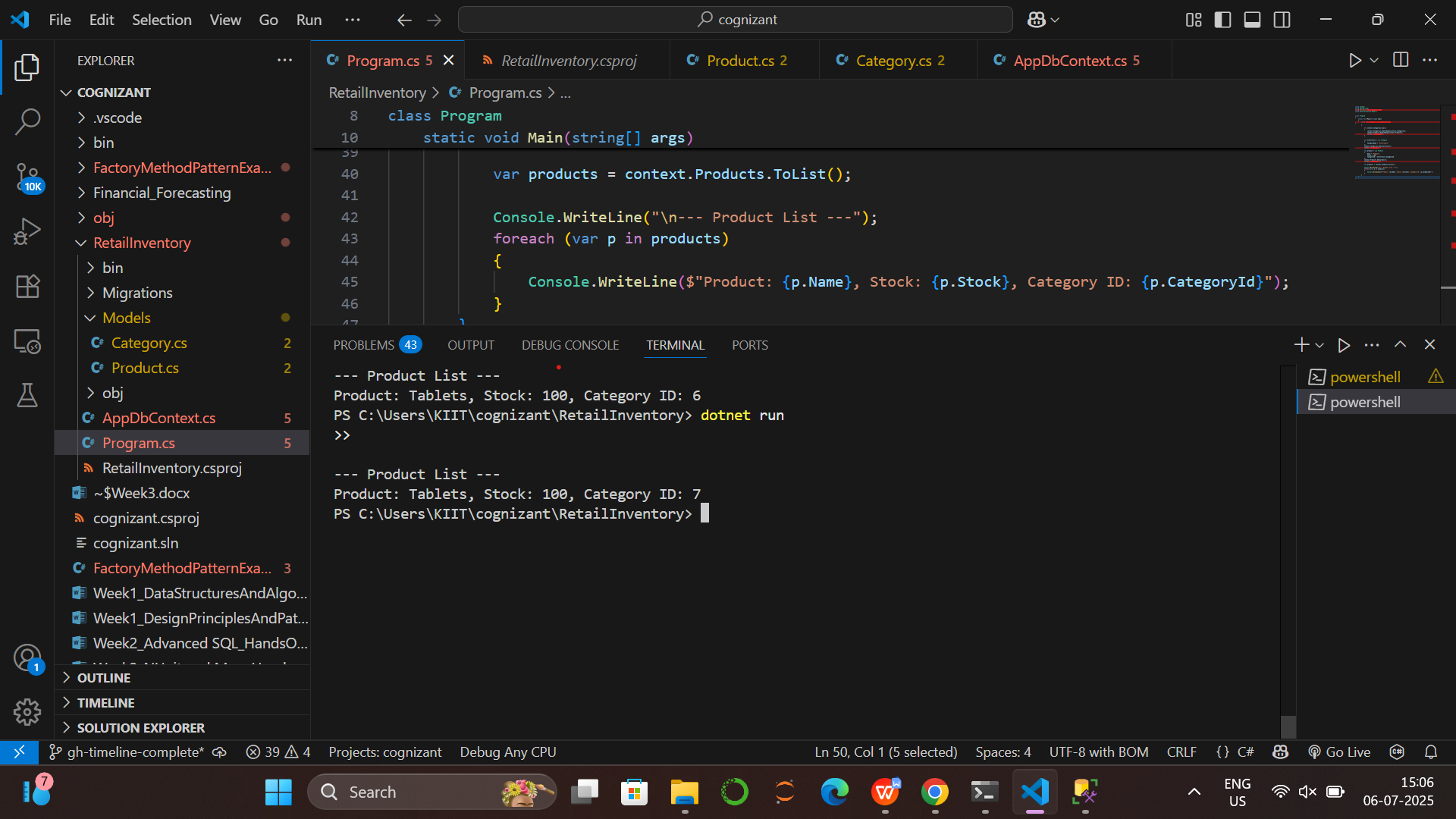
}

}

}

}

*OUTPUT:*



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

**PROJECT STRUCTURE**

RetailInventoryLab2 is a .NET Console App built using EF Core 9.0 with a simple inventory management system. The project contains:

Models/: Contains entity classes Category.cs and Product.cs.

AppDbContext.cs: Configures EF Core and defines the DbSet properties.

Program.cs: The main entry point of the app that seeds and displays data.

Migrations/: Contains auto-generated EF migration files.

RetailInventoryLab2.csproj: Project file with EF Core dependencies.

**CODE:**

*Models/Product.cs*

namespace RetailInventory.Models;

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; } = null!;

}

*Models/Category .cs*

namespace RetailInventory.Models;

public class Category

{

public int Id { get; set; }

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

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

}

*AppDbContext .cs*

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;Encrypt=False;");

}

}

*Program .cs*

using RetailInventory.Models;

using System;

using System.Linq;

class Program

{

static void Main(string[] args)

{

using (var context = new AppDbContext())

{

// Add sample data if not exists

if (!context.Categories.Any())

{

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

var product = new Product

{

Name = "Wheat",

Price = 75.5M,

Category = category

};

context.Categories.Add(category);

context.Products.Add(product);

context.SaveChanges();

}

var products = context.Products.ToList();

Console.WriteLine("--- Product List ---");

foreach (var p in products)

{

Console.WriteLine($"Name: {p.Name}, Price: ₹{p.Price}, CategoryId: {p.CategoryId}");

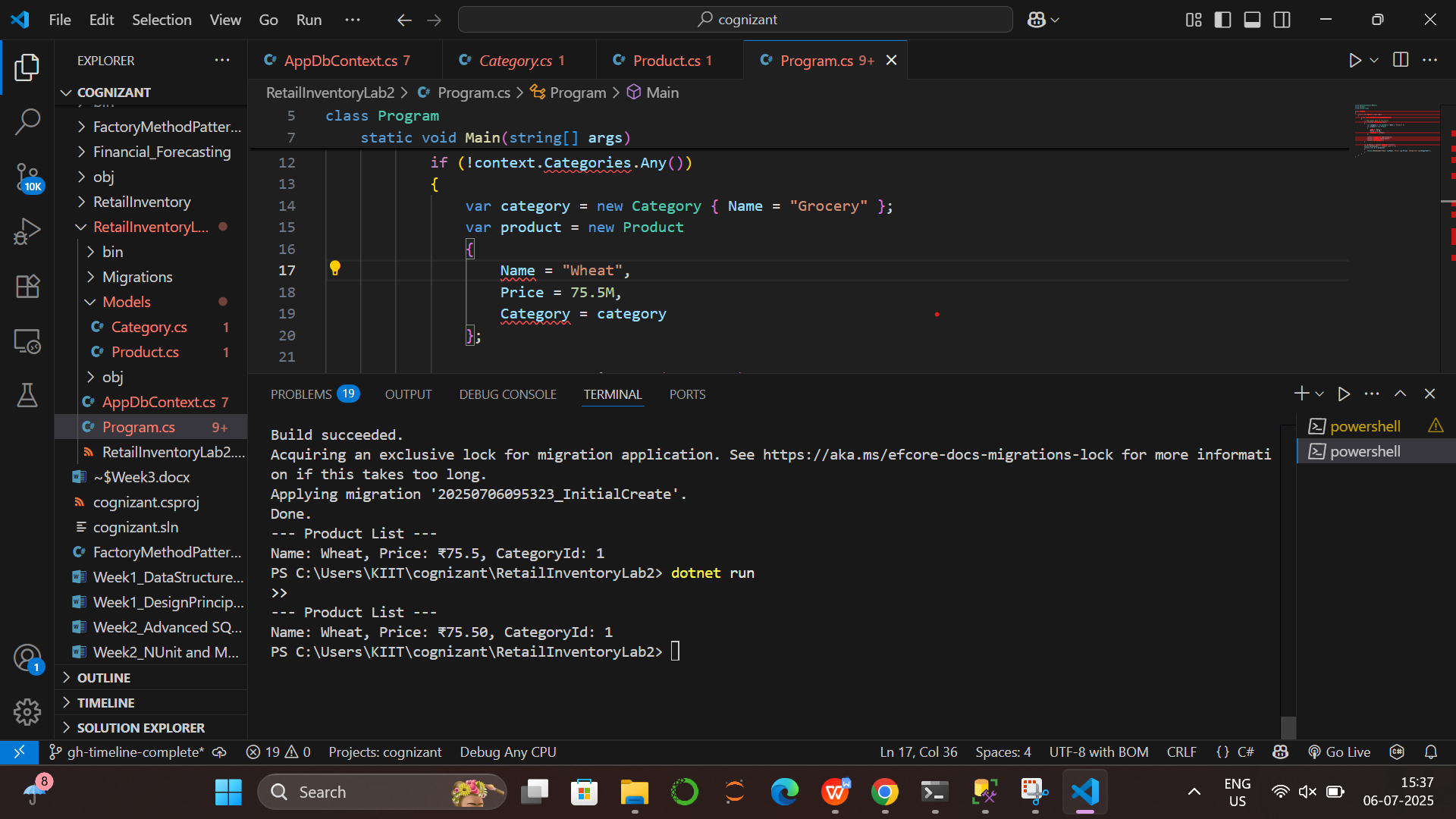
}

}

}

}

*OUTPUT:*

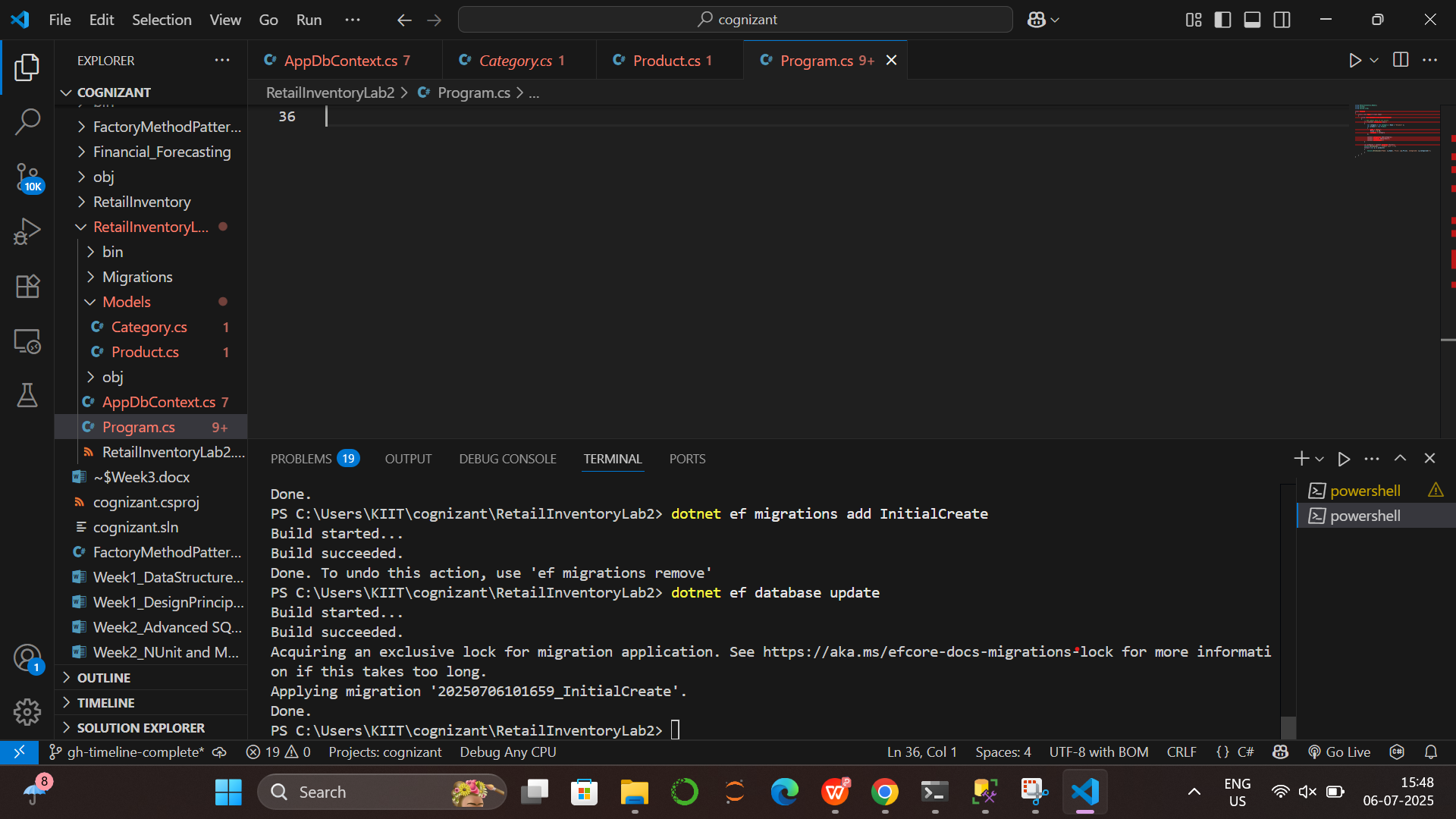


**Lab 3: Using EF Core CLI to Create and Apply Migrations**

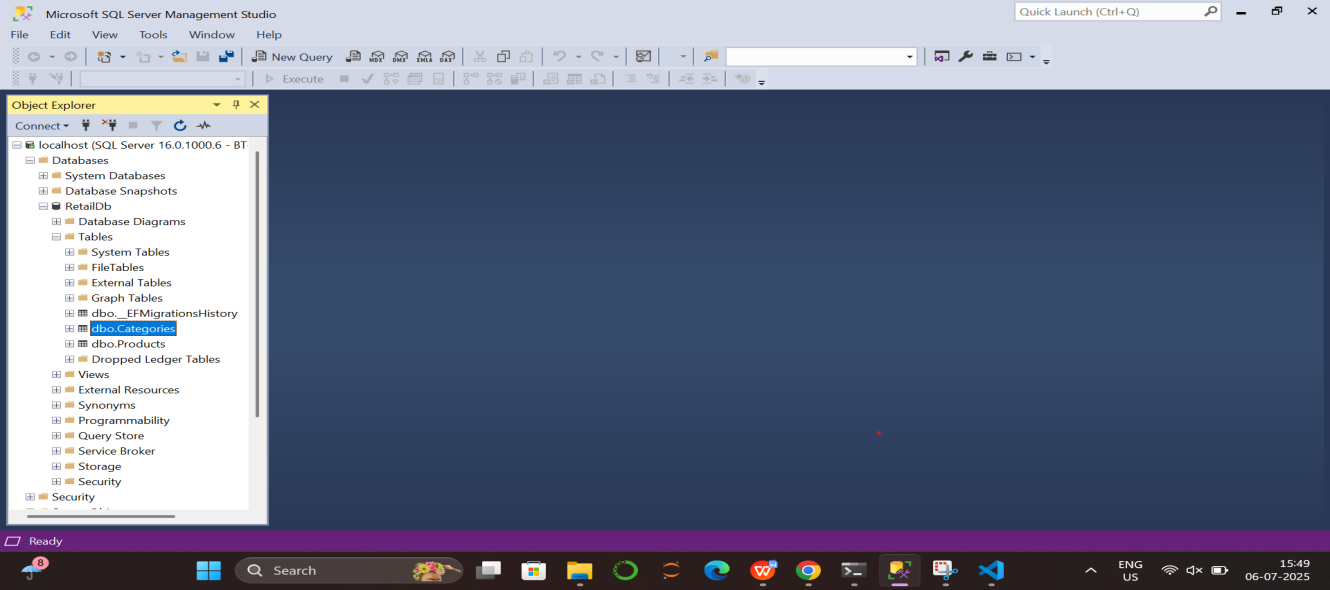
dotnet tool install --global dotnet-ef

dotnet ef migrations add InitialCreate

dotnet ef database update

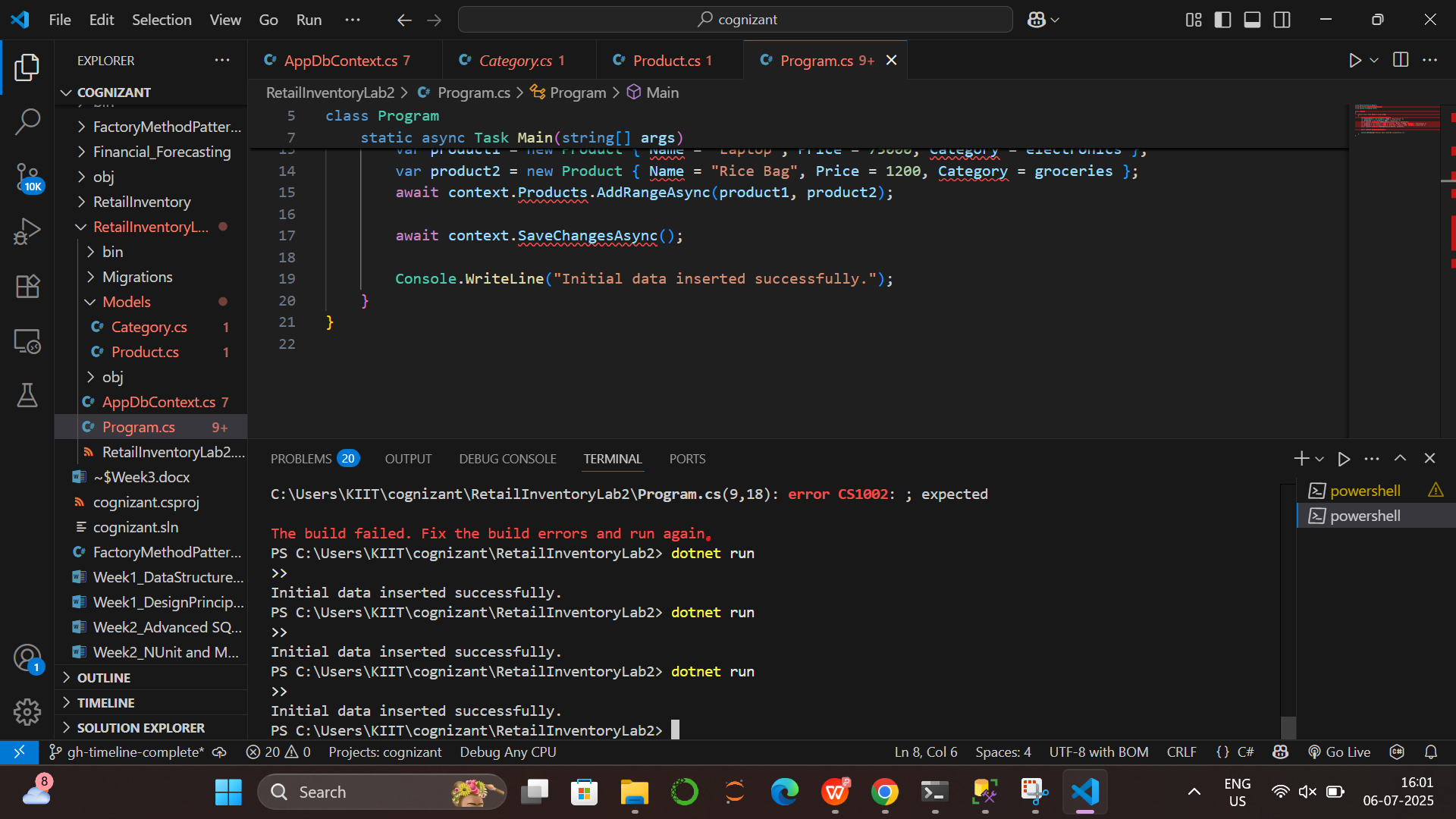


**Verify in SQL Server:**

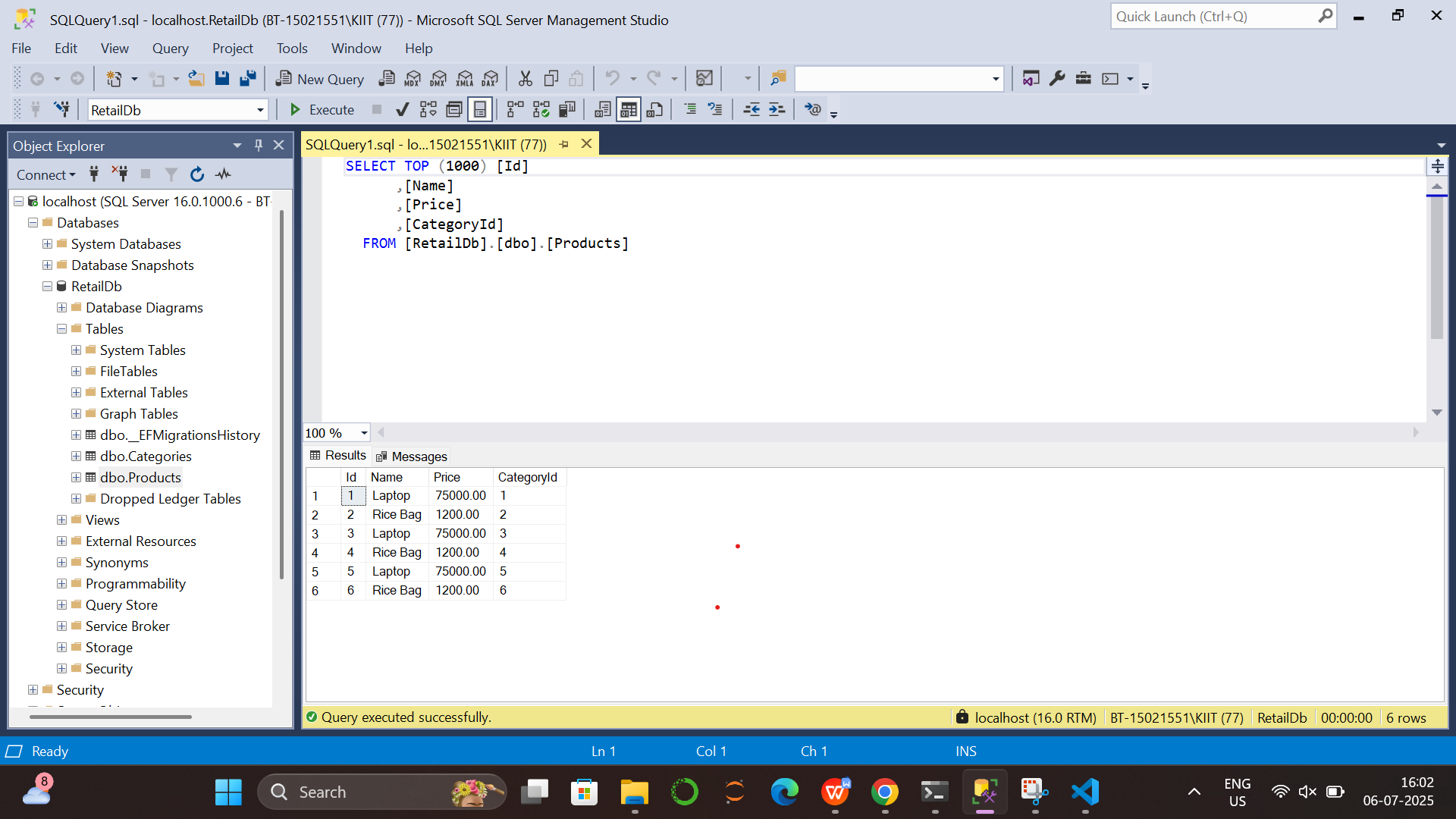


**Lab 4: Inserting Initial Data into the Database**

dotnet run



**Verify in SQL Server:**



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

**CODE**

*Program.cs*

using RetailInventory.Models;

using Microsoft.EntityFrameworkCore;

using System;

using System.Linq;

using System.Threading.Tasks;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

Console.WriteLine("\n All Products ");

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

foreach (var p in products)

{

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

}

Console.WriteLine("\n Find by ID (ID = 1) ");

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

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

Console.WriteLine("\n First Expensive Product (Price > 50000) ");

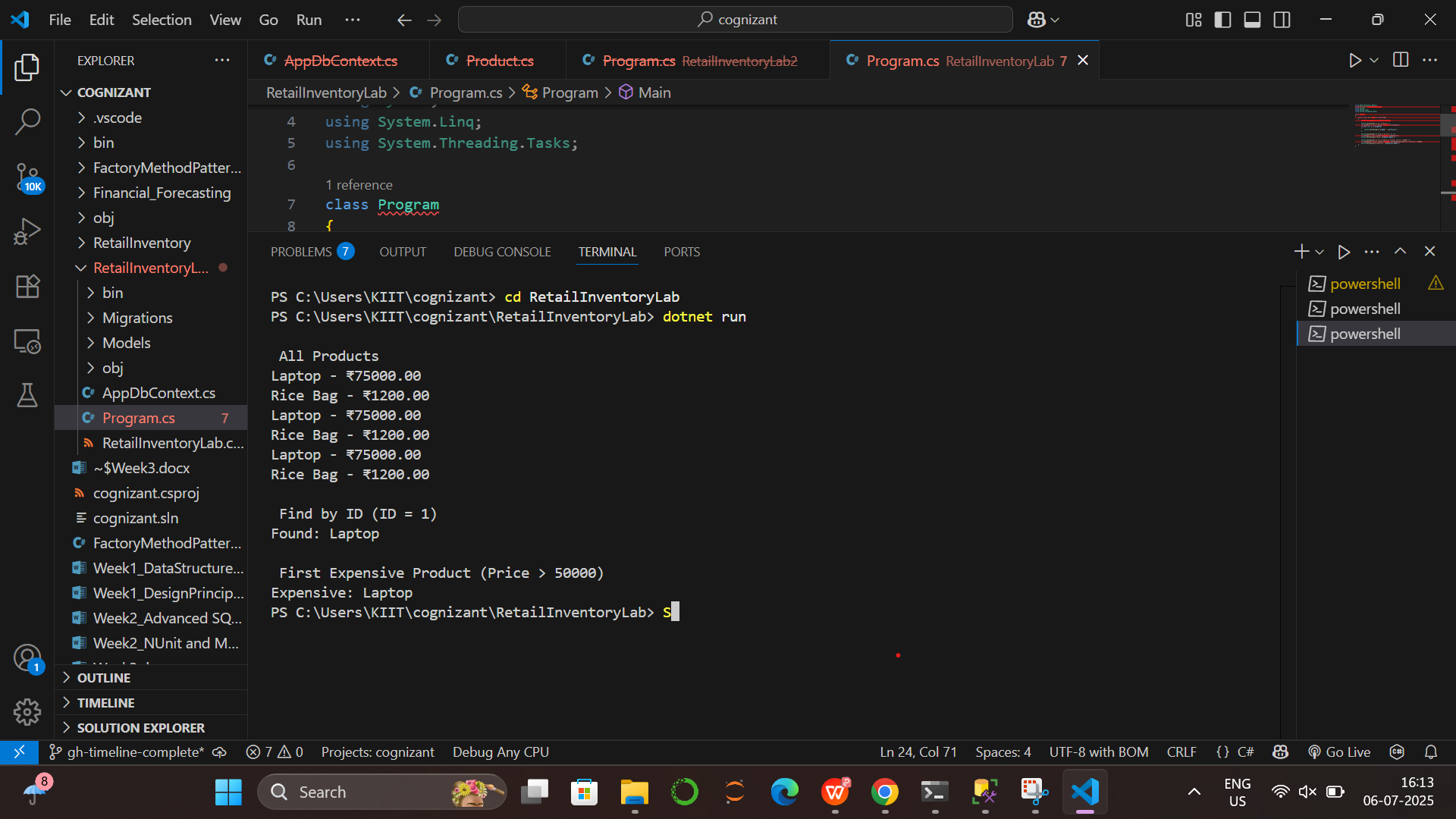
var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);

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

}

}

*OUTPUT:*



**Lab 6: Updating and Deleting Records**

**CODE**

*Program.cs*

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

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 price of '{product.Name}' 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("\n Current Product List ");

foreach (var p in products)

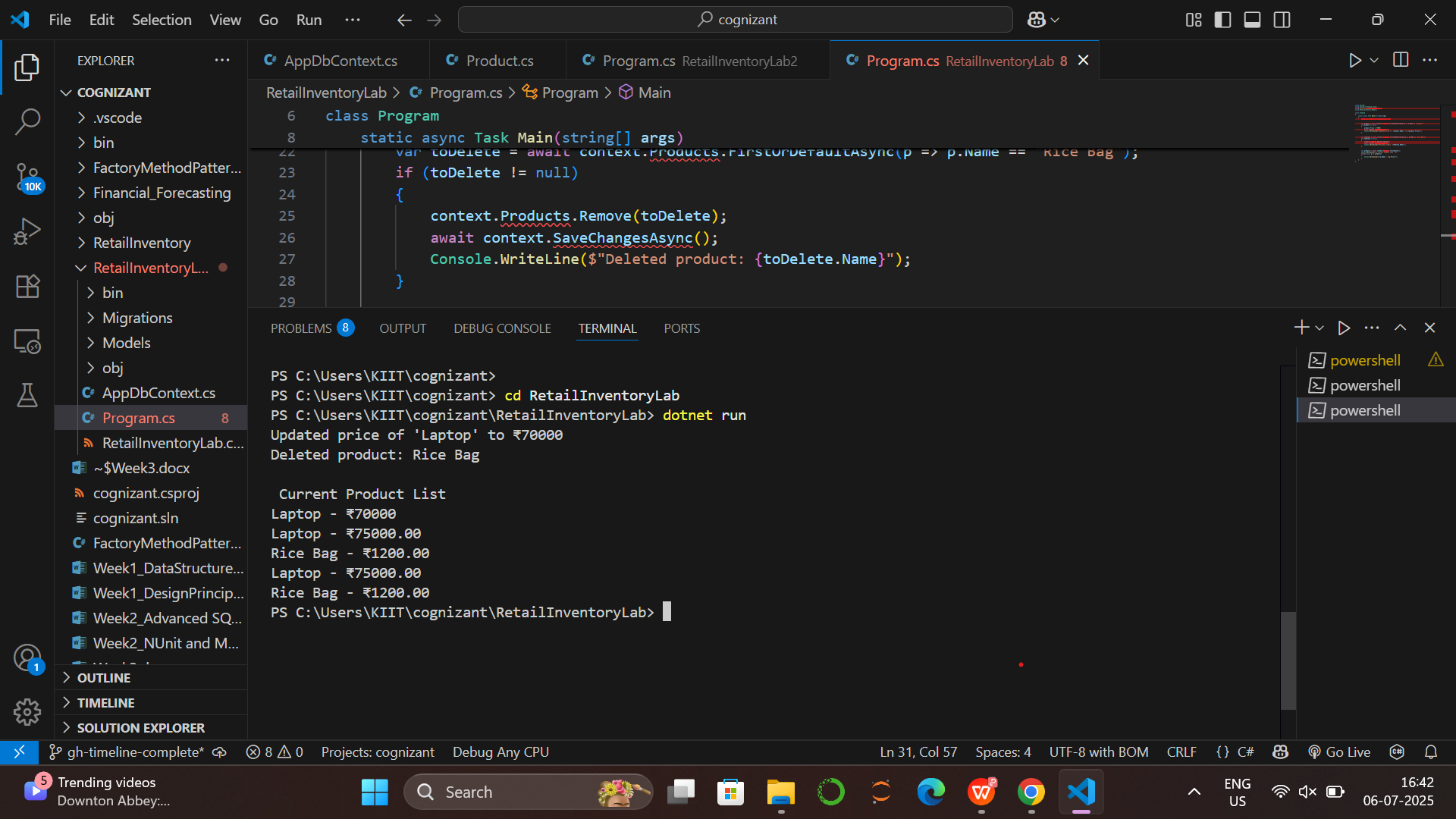
{

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

}

}

}

*OUTPUT:*

**Lab 7: Writing Queries with LINQ**

**CODE**

*Program.cs*

using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

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 & 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("\n Product DTOs (Name & Price only) ");

foreach (var dto in productDTOs)

{

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

}

}

}

*OUTPUT:*

