# **DN4.0-WEEK3**

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

### **What is ORM?**

Object-Relational Mapping (ORM) is a technique that connects C# objects to database tables so that you don’t have to write raw SQL queries all the time.

**Working of ORM in EF Core**

* A C# class (e.g., Product) represents a table in the database.
* Each property in the class (e.g., ProductName, Price) maps to a column in that table.
* EF Core handles the conversion (mapping) between objects in your app and rows in the database.

**Example:**

public class Product

{

    public int Id { get; set; } //maps to Product.Id column

    public string Name { get; set; } //maps to Product.Name column

    public decimal Price { get; set; } //maps to Product.Price column

}

EF Core:

* Insert objects as rows
* Read database rows into objects
* Update and Delete rows using object manipulation

**Benefits of ORM**

* Productivity: Business logic using C# is focused more than SQL logics.
* Maintainability: Updating models in C# is easier without touching SQL code.
* Abstraction: Database level operations are taken care of EF Core and we only need to work at object level (C#).

### **EF Core vs EF Framework (EF6)**

|  |  |  |
| --- | --- | --- |
| **Feature** | **EF Core** | **EF Framework (EF6)** |
| Platform | Cross-platform (.NET Core / .NET 6/7/8) | Windows only |
| Modern Features | Supports async/await, LINQ, compiled queries | Lacks some newer features |
| Performance | Lightweight and faster with newer versions | Heavier, slower |
| Flexibility | Highly extensible (interceptors, migrations, etc.) | Less flexible |
| JSON Columns | Supported in EF Core 8 | Not supported |

### **EF Core 8.0 Features**

**JSON Column Mapping**

* Can map a property to a JSON column in SQL Server 2022.
* Useful for storing semi-structured data (e.g., product specifications or attributes).

public class Product

{

    public int Id { get; set; }

    public ProductDetails Details { get; set; } // stored as JSON

}

**Compiled Models**

* Improves startup performance for large models.
* EF pre-generates metadata so runtime execution is faster.

**Interceptors**

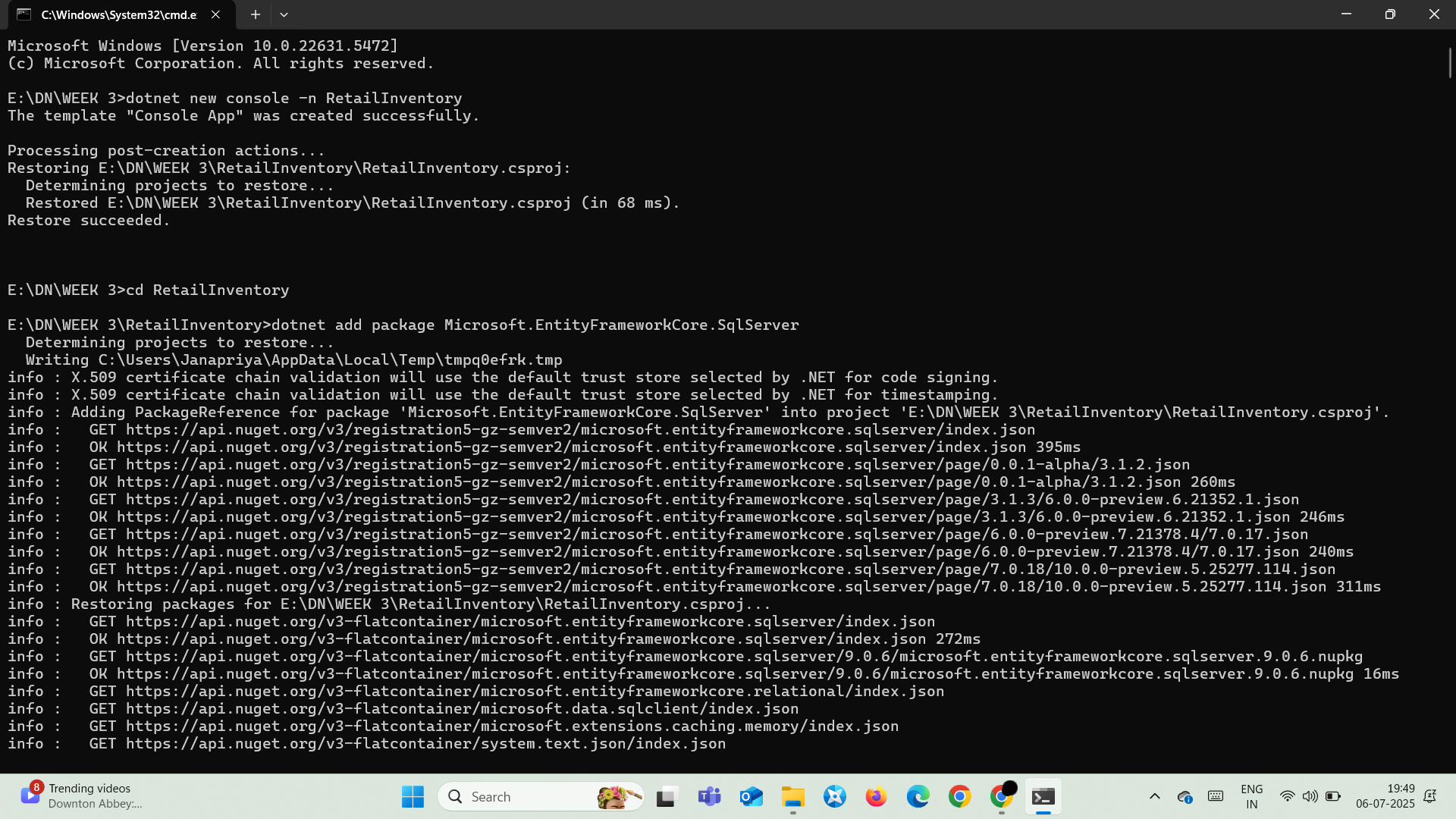
* Can intercept and log queries, commands, or changes which are useful for debugging or analytics.

**Bulk Updates & Deletes**

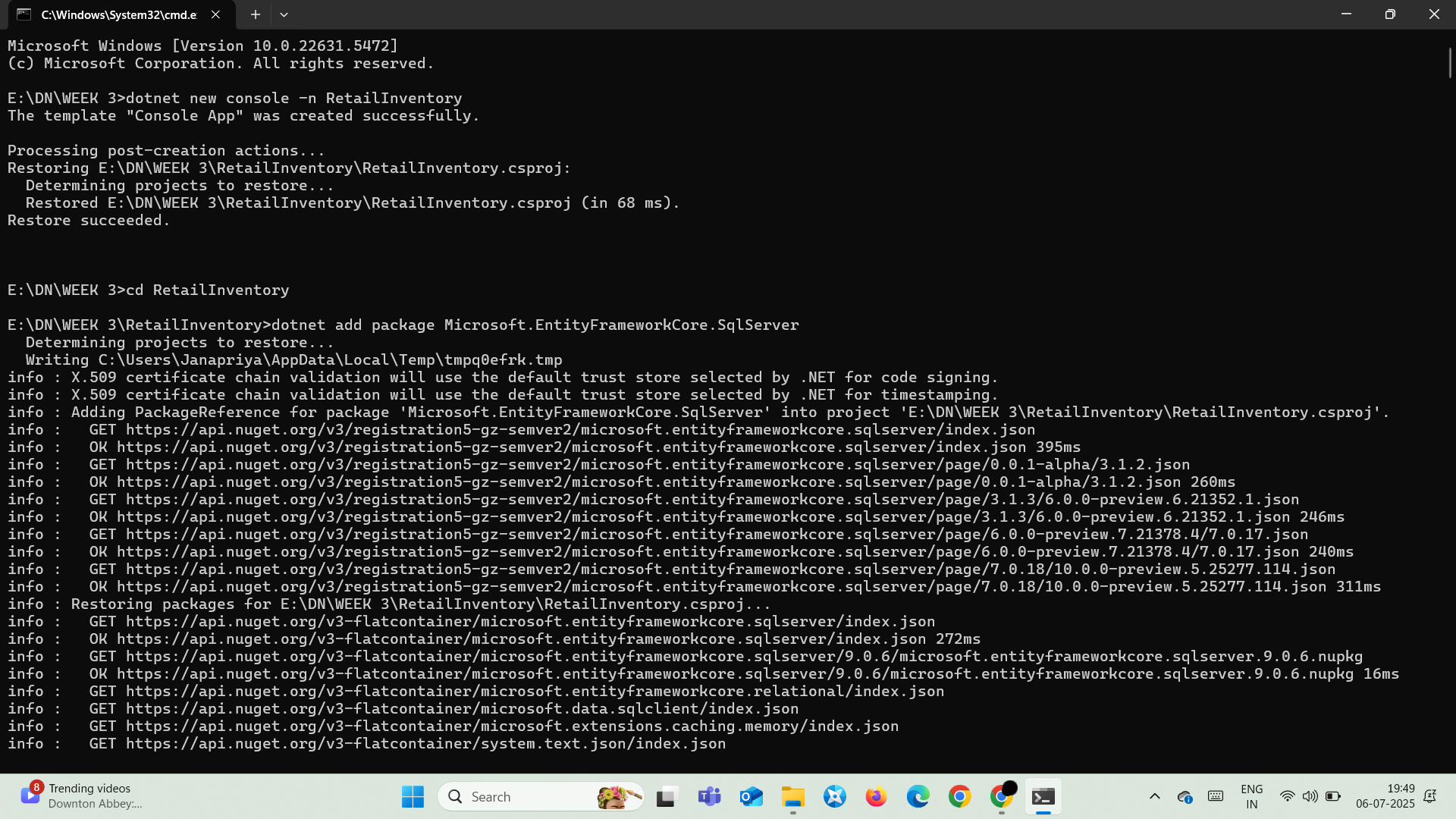
* EF Core 8 supports efficient bulk operations without loading entities into memory.

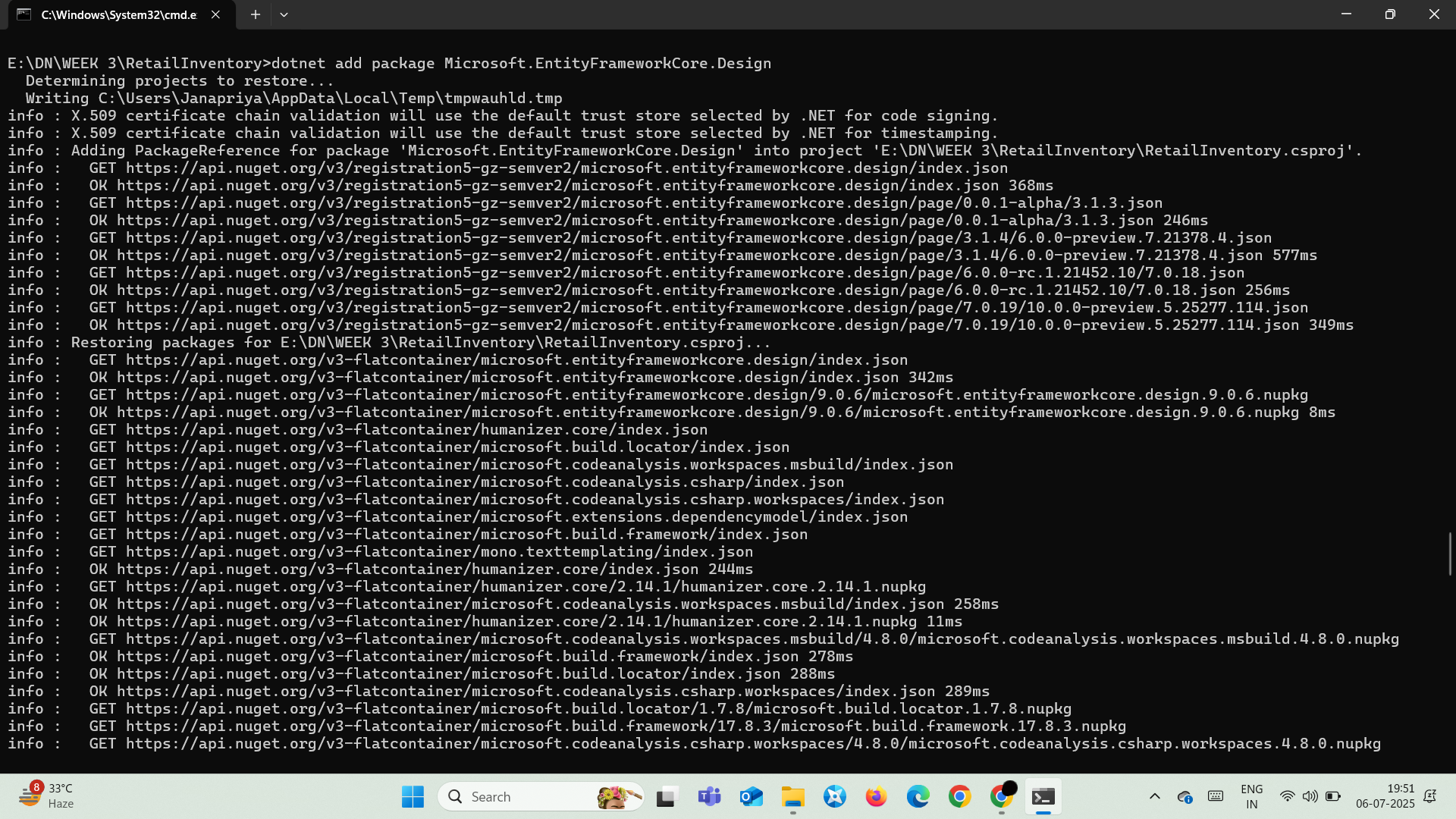
### **Create a .NET Console App**

To create a new Console Application in a folder called “RetailInventory”



### **Install EF Core Packages**





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

### **Model/**

#### **Category**

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; }

    }

}

#### **Product**

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; }

    }

}

### **AppDbContext.cs**

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

public class AppDbContext : DbContext

{

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

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

    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

    protected override void OnModelCreating(ModelBuilder modelBuilder)

    {

        base.OnModelCreating(modelBuilder);

    }

}

### **appsettings.json**

{

  "ConnectionStrings": {

    "DefaultConnection": "Server=localhost;Database=RetailInventoryDb;Trusted\_Connection=True;TrustServerCertificate=True;"

  }

}

### **Program.cs**

using Microsoft.Extensions.Configuration;

using Microsoft.EntityFrameworkCore;

using System.IO;

class Program

{

    static void Main(string[] args)

    {

        var config = new ConfigurationBuilder()

            .SetBasePath(Directory.GetCurrentDirectory())

            .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)

            .Build();

        var connectionString = config.GetConnectionString("DefaultConnection");

        var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();

        optionsBuilder.UseSqlServer(connectionString);

        using (var context = new AppDbContext(optionsBuilder.Options))

        {

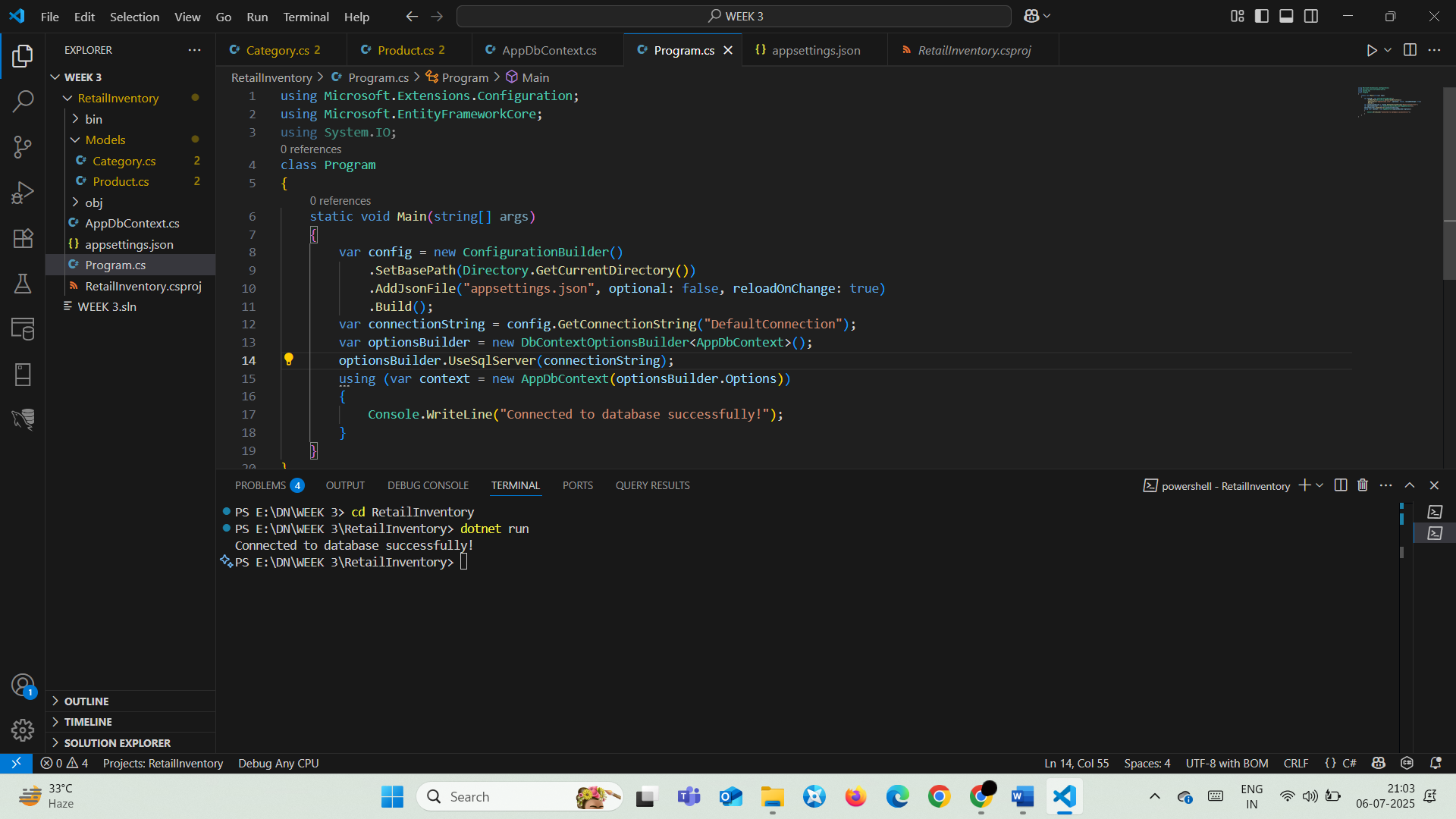
            Console.WriteLine("Connected to database successfully!");

        }

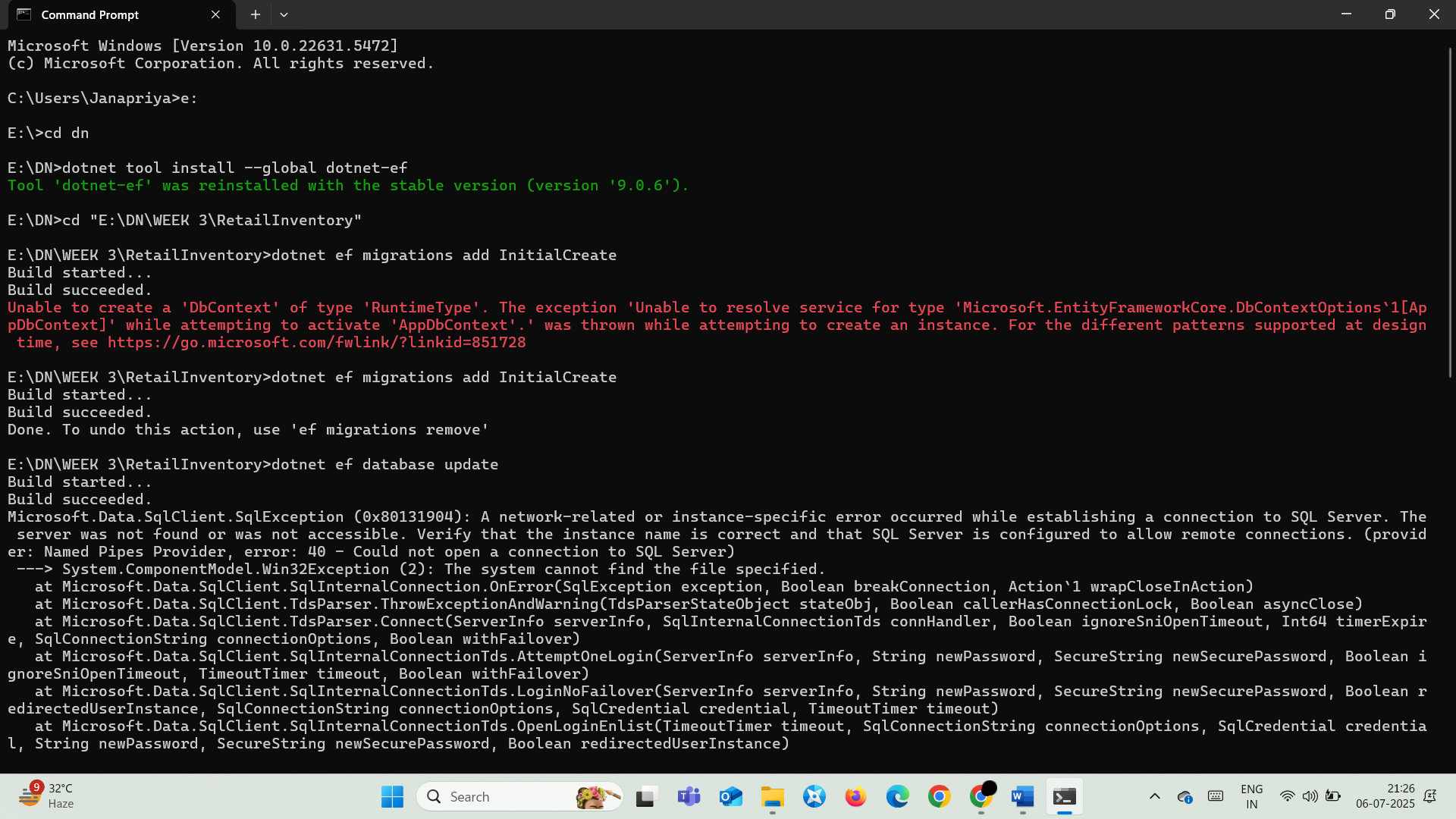
    }

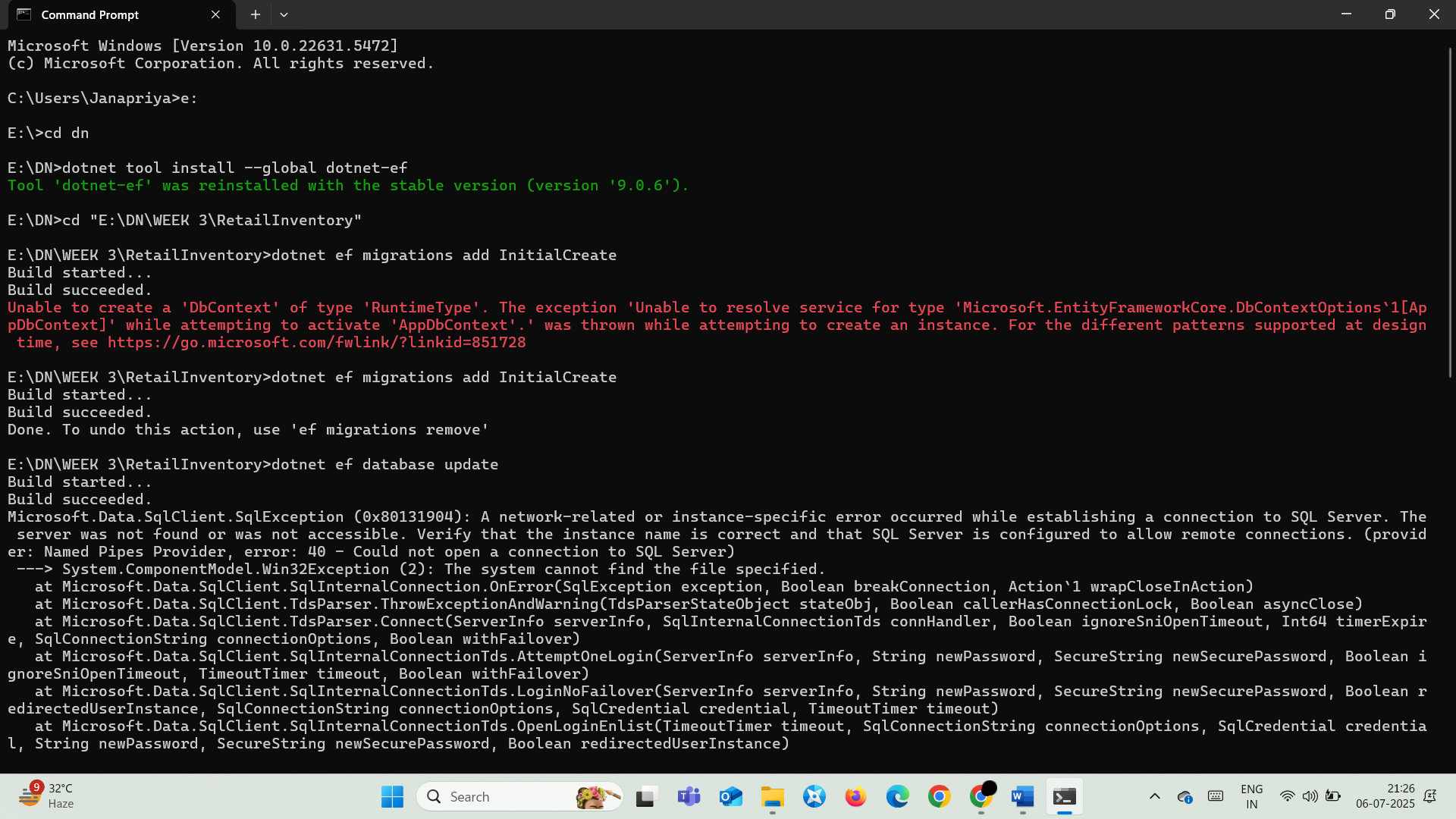
}

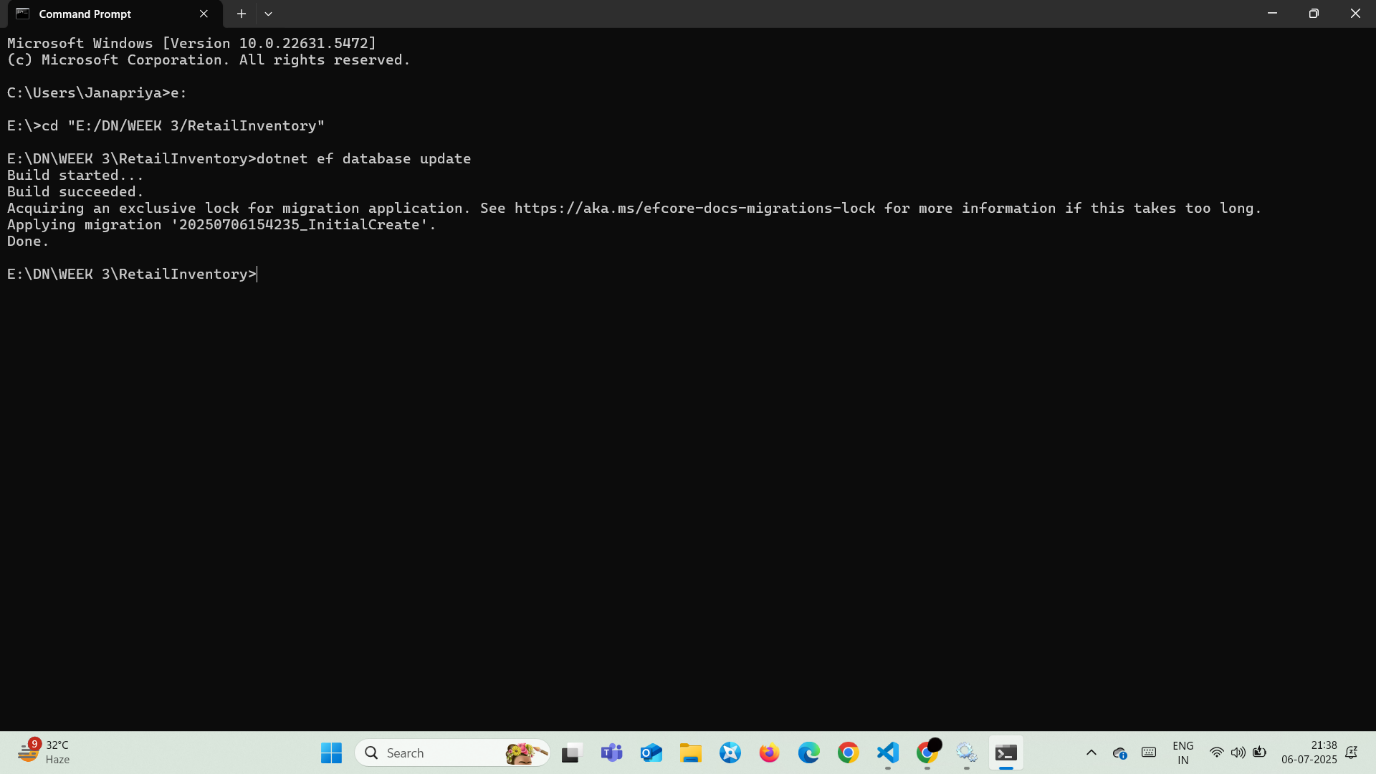
### **Output:**

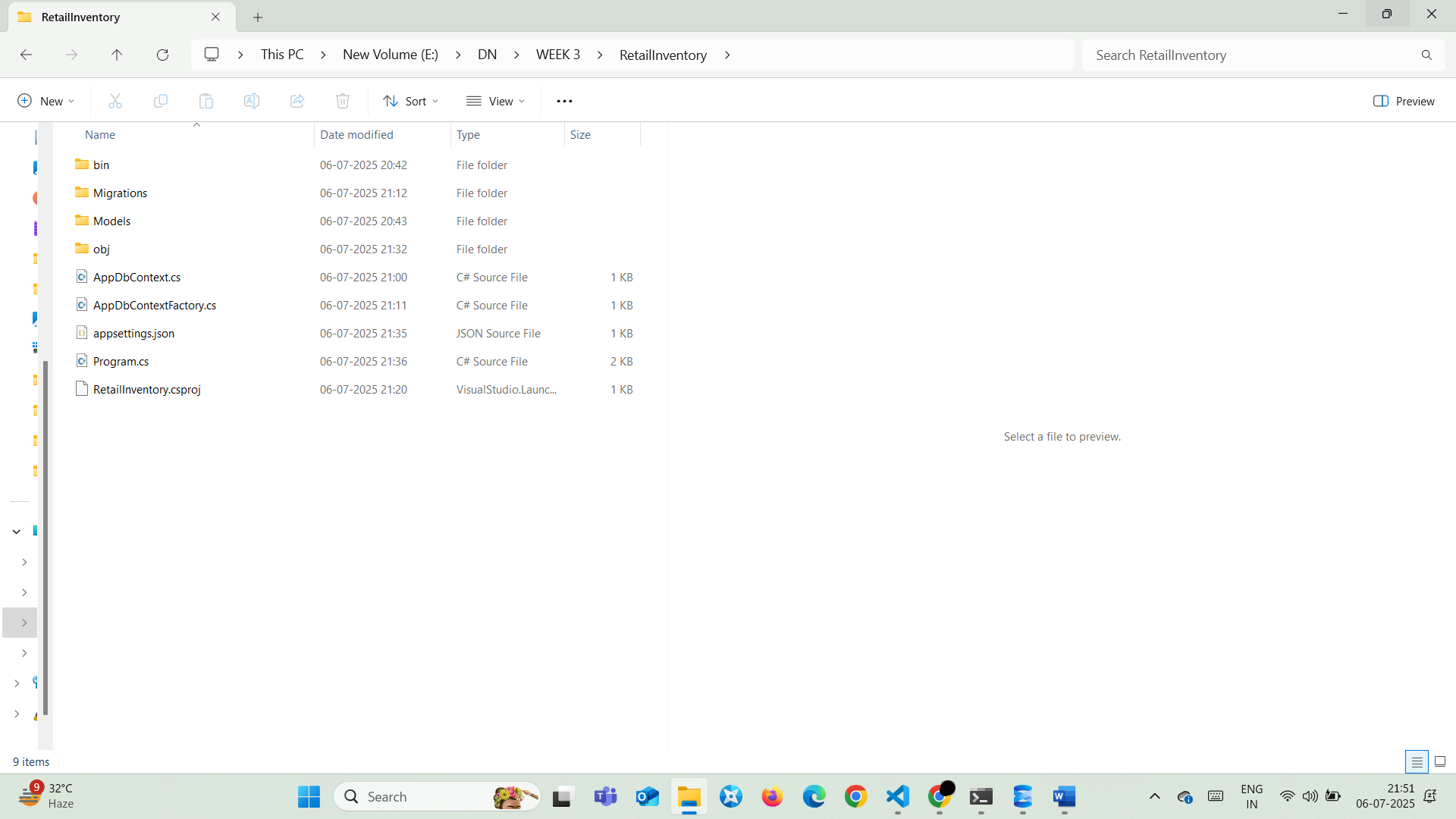


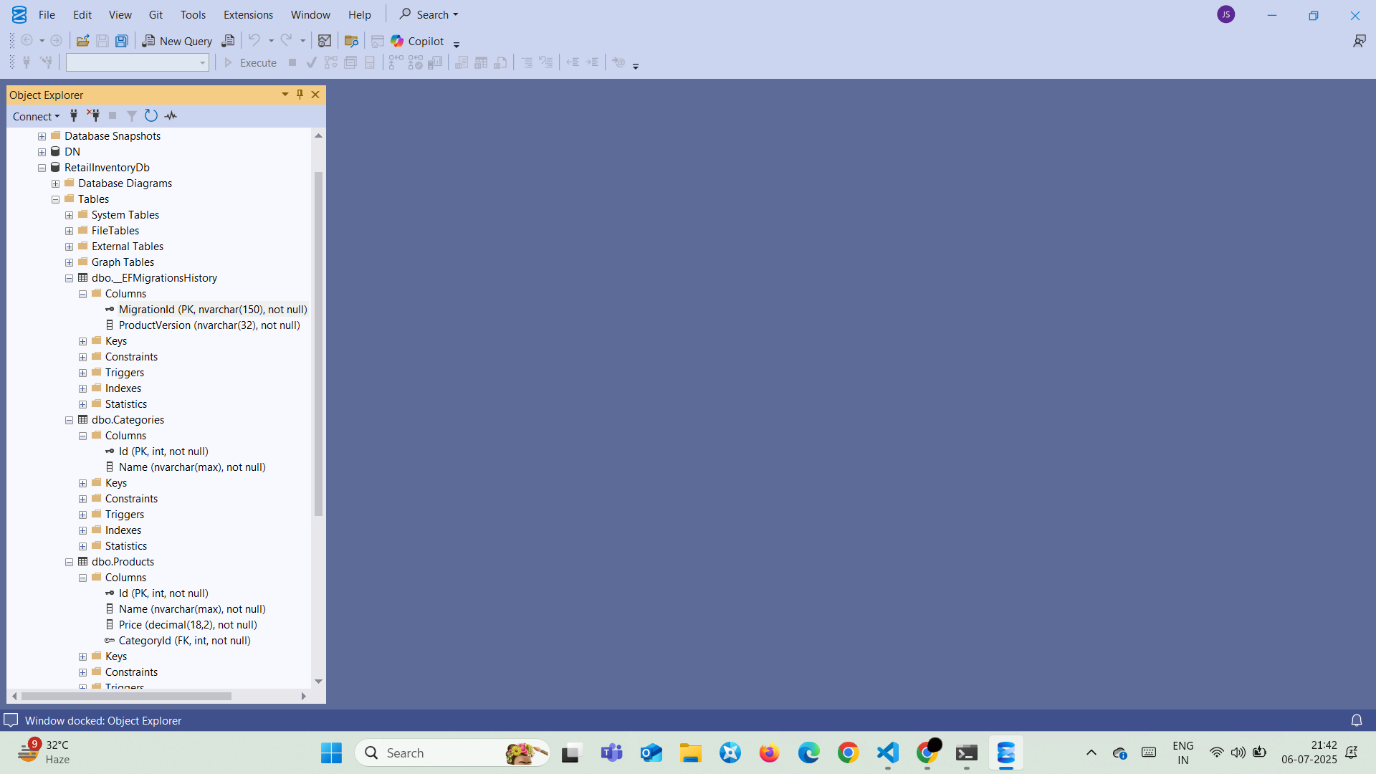
## **Lab 3:** **Using EF Core CLI to Create and Apply Migrations Scenario**











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

### **Program.cs**

using System;

using System.IO;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using RetailInventory.Models;

class Program

{

    static async Task Main(string[] args)

    {

        var config = new ConfigurationBuilder()

            .SetBasePath(Directory.GetCurrentDirectory())

            .AddJsonFile("appsettings.json")

            .Build();

        var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();

        var connectionString = config.GetConnectionString("DefaultConnection");

        optionsBuilder.UseSqlServer(connectionString);

        using var context = new AppDbContext(optionsBuilder.Options);

        if (!await context.Categories.AnyAsync())

        {

            var stationery=new Category{Name="Stationery", Products=new List<Product>()};

            var accessories = new Category { Name = "Accessories", Products = new List<Product>() };

            await context.Categories.AddRangeAsync(stationery, accessories);

            var product1 = new Product { Name = "Pens", Price = 200, Category = stationery };

            var product2 = new Product { Name = "Bracelet", Price = 120, Category = accessories };

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

            await context.SaveChangesAsync();

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

        }

        else

        {

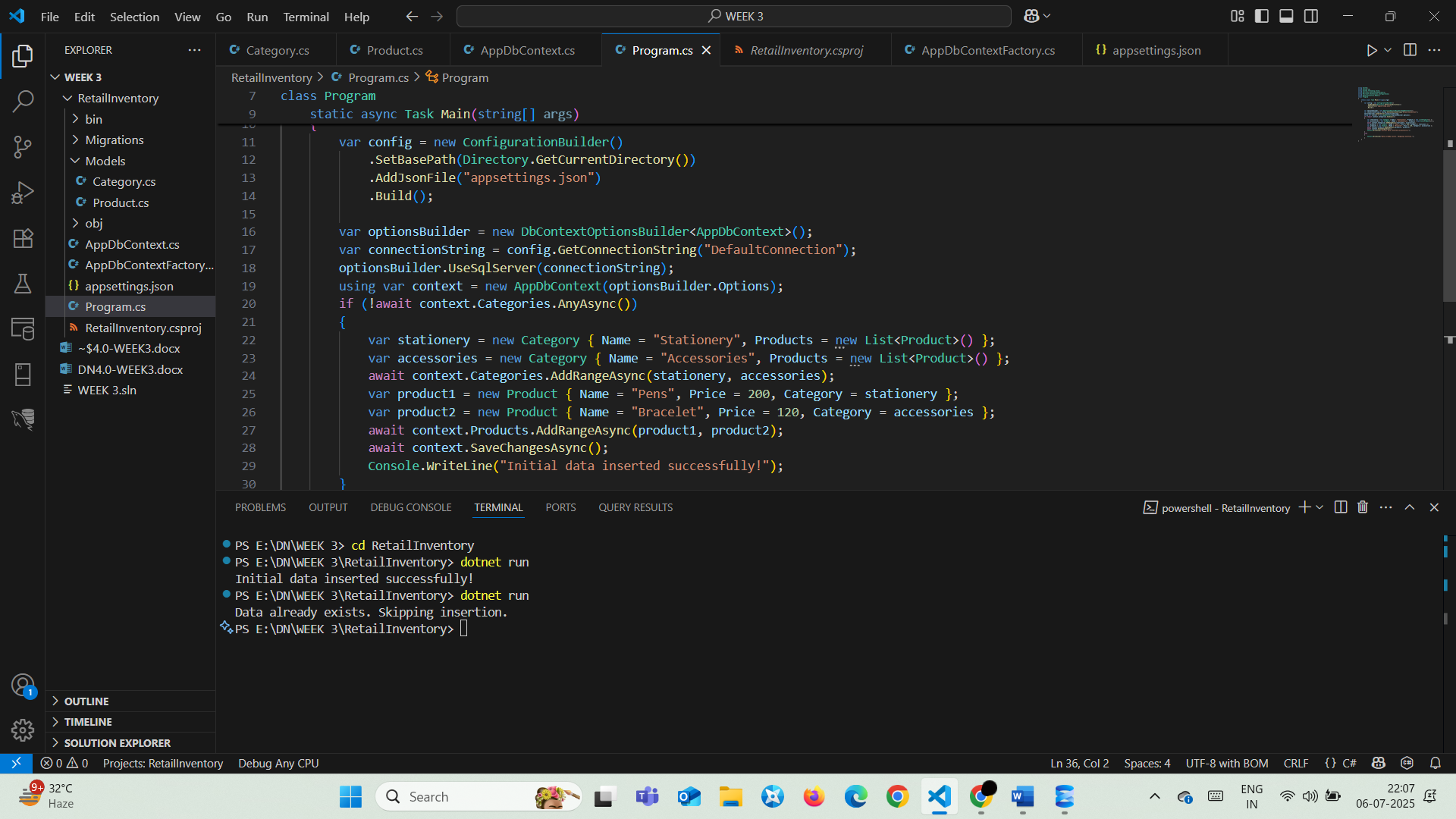
            Console.WriteLine("Data already exists. Skipping insertion.");

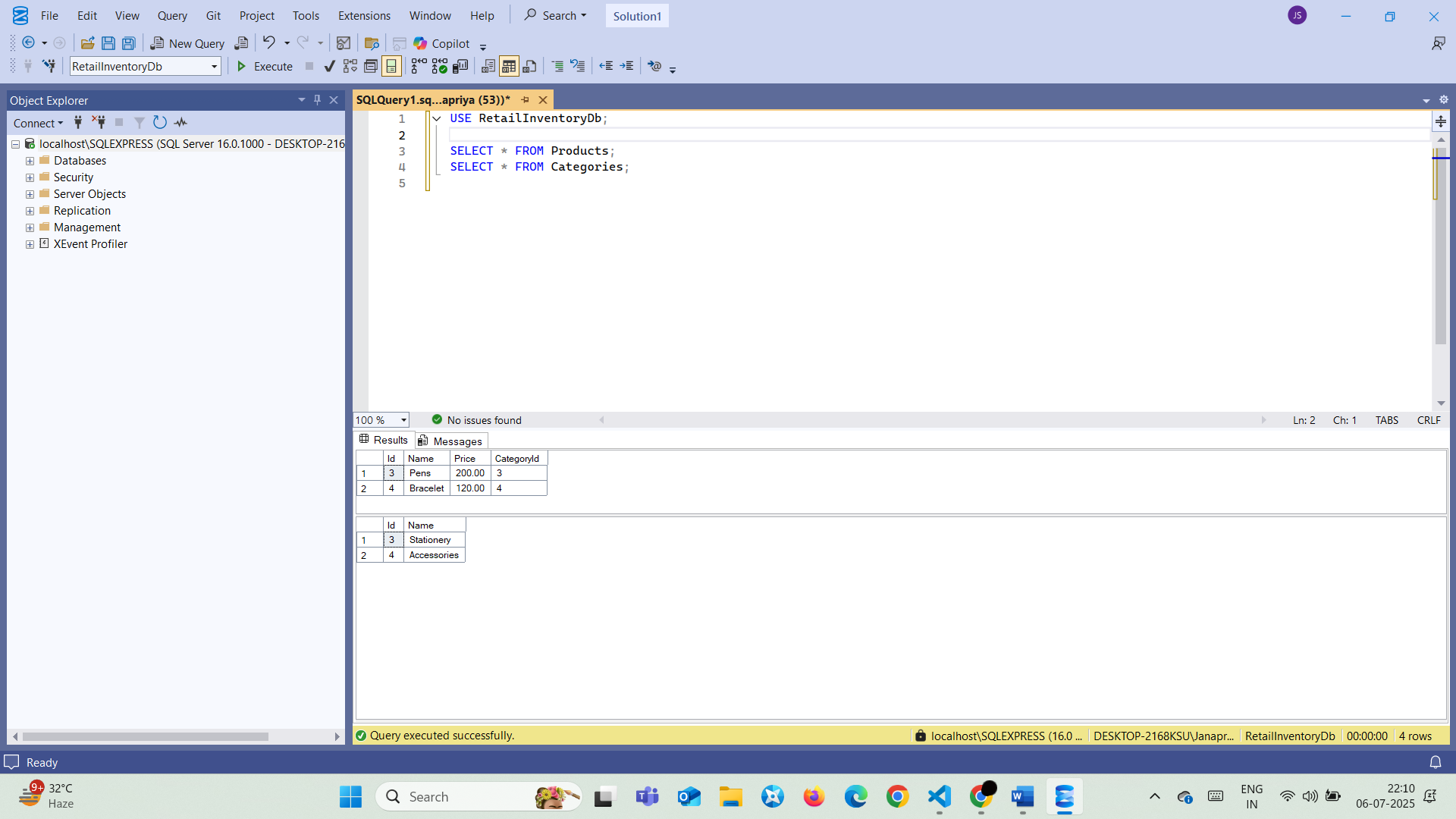
        }

    }

}

### **Output:**





## **Lab 5:** **Using EF Core CLI to Create and Apply Migrations Scenario**

### **Program.cs:**

using System;

using System.IO;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using RetailInventory.Models;

class Program

{

    static async Task Main(string[] args)

    {

        var config = new ConfigurationBuilder()

            .SetBasePath(Directory.GetCurrentDirectory())

            .AddJsonFile("appsettings.json")

            .Build();

        var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();

        var connectionString = config.GetConnectionString("DefaultConnection");

        optionsBuilder.UseSqlServer(connectionString);

        using var context = new AppDbContext(optionsBuilder.Options);

        if (!await context.Categories.AnyAsync())

    {

        var stationery = new Category { Name = "Stationery", Products = new List<Product>() };

        var accessories = new Category { Name = "Accessories", Products = new List<Product>() };

        var electronics = new Category { Name = "Electronics", Products = new List<Product>() };

        await context.Categories.AddRangeAsync(stationery, accessories, electronics);

        var product1=new Product{Name="Pens", Price=200, Category=stationery};

        var product2=new Product{Name="Bracelet", Price=120, Category=accessories};

        var product3=new Product{Name="iPhone", Price=75000, Category=electronics};

        await context.Products.AddRangeAsync(product1, product2, product3);

        await context.SaveChangesAsync();

        Console.WriteLine("Data inserted successfully!");

    }

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

        Console.WriteLine("All Products:");

        foreach (var p in products)

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

        var product = await context.Products.FindAsync(5);

        Console.WriteLine($"\nFound by ID: {product?.Name ?? "Not Found"}");

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

        Console.WriteLine($"\nExpensive Product: {expensive?.Name ?? "None found"}");

    }

}

### **Output:**

