**SuperSet Id: 6365365**

**Lab1-**

#### ****Step 1: What is ORM?****

#### ****ORM (Object-Relational Mapping)**** maps C# classes to database tables.

Benefits:

No need to write SQL queries for common operations.

Cleaner code (deals with objects instead of rows/columns)

Easier maintenance and refactoring.

#### ****Step 2: EF Core vs EF Framework****

EF Core is:

Lightweight

Cross-platform

Modern (async/await, LINQ support)

EF Framework (EF6):

Windows-only

Mature but tightly coupled to the older .NET Framework.

#### ****Step 3: EF Core 8.0 Features****

JSON column mapping (store structured data in a single column)

Performance improvements.

Support for interceptors and bulk operations

**Step4 and Step5:**

**MODELS Folder:**

**Category.cs**

namespace RetailInventory.Models

{

    public class Category

    {

        public int CategoryId { get; set; }

        public string Name { get; set; }

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

    }

}

**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; }

    }

}

**DATA Folder:**

**AppDbContext.cs**

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory.Data

{

    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=(localdb)\MSSQLLocalDB;Database=RetailInventoryDb;Trusted\_Connection=True;");

        }

    }

}

**Program.cs**

using RetailInventory.Data;

using RetailInventory.Models;

using Microsoft.EntityFrameworkCore;

using (var context = new AppDbContext())

{

    context.Database.EnsureCreated();

    if (!context.Categories.Any())

    {

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

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

        context.Categories.AddRange(electronics, groceries);

        context.Products.AddRange(

            new Product { Name = "TV", Stock = 10, Category = electronics },

            new Product { Name = "Laptop", Stock = 5, Category = electronics },

            new Product { Name = "Apple", Stock = 50, Category = groceries }

        );

        context.SaveChanges();

        Console.WriteLine("Data Seeded!");

    }

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

    foreach (var p in products)

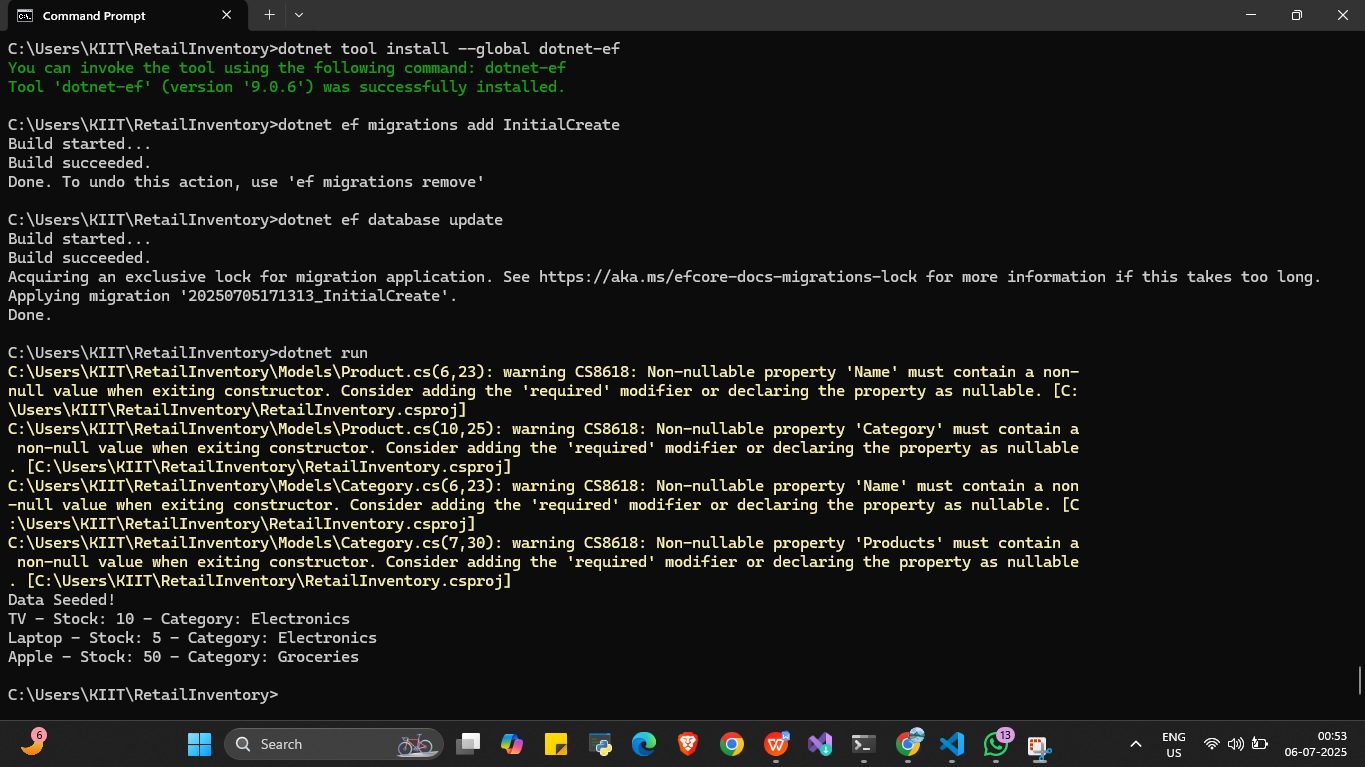
    {

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

    }

}

**Output:**



**LAB2:**

**RetailStoreDb**

**Models**

**Category.cs**

using System.Collections.Generic;

namespace RetailStoreDb.Models

{

    public class Category

    {

        public int Id { get; set; }

        public string? Name { get; set; }

        public List<Product>? Products { get; set; }

    }

}

**Product.cs**

namespace RetailStoreDb.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 RetailStoreDb.Models;

namespace RetailStoreDb

{

    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=(localdb)\\MSSQLLocalDB;Database=RetailStoreDb;Trusted\_Connection=True;");

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            modelBuilder.Entity<Product>()

                .Property(p => p.Price)

                .HasPrecision(18, 2); // 👈 Set precision for decimal

            base.OnModelCreating(modelBuilder);

        }

    }

}

**Program.cs**

using Microsoft.EntityFrameworkCore;

using RetailStoreDb;

using RetailStoreDb.Models;

var context = new AppDbContext();

if (!context.Categories.Any())

{

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

    var clothing = new Category { Name = "Clothing" };

    context.Categories.AddRange(electronics, clothing);

    context.Products.AddRange(

        new Product { Name = "Laptop", Price = 45000, Category = electronics },

        new Product { Name = "T-shirt", Price = 500, Category = clothing }

    );

    context.SaveChanges();

}

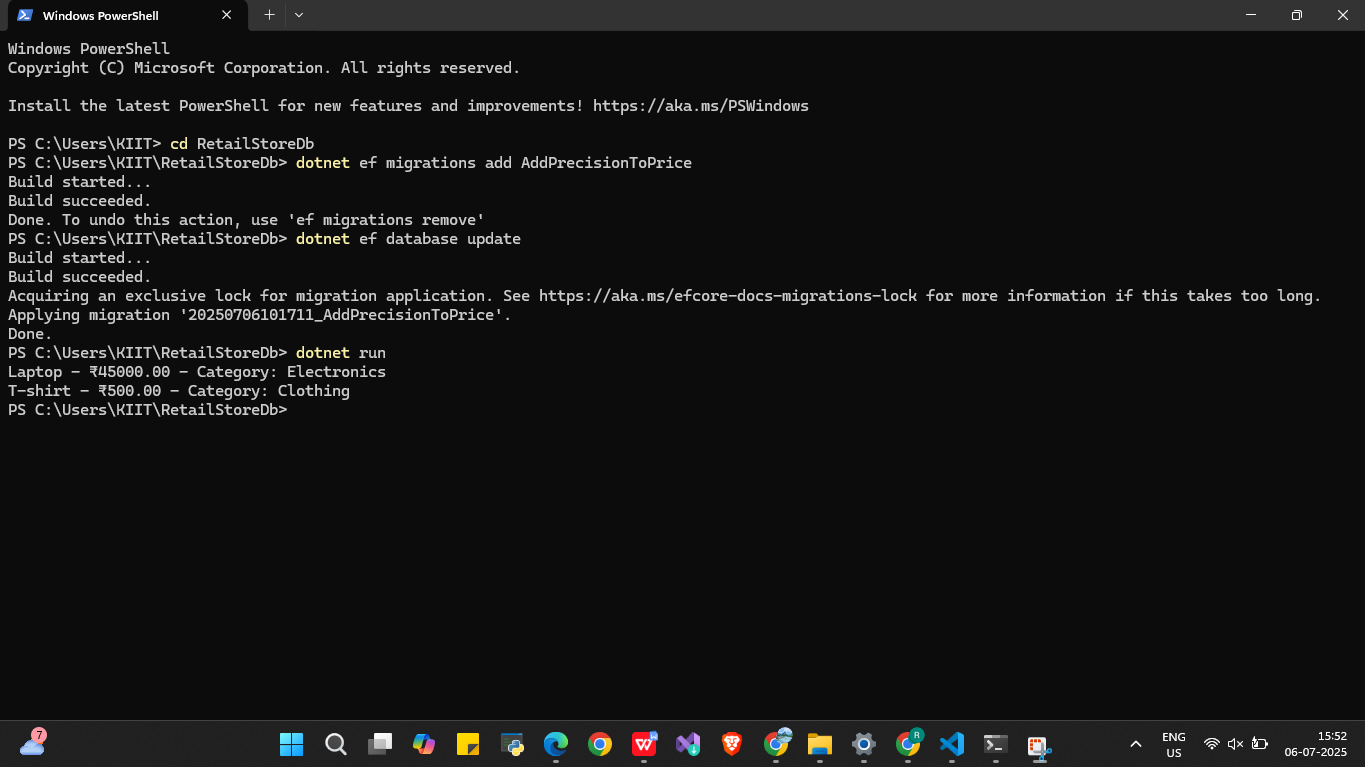
foreach (var product in context.Products.Include(p => p.Category))

{

    Console.WriteLine($"{product.Name} - ₹{product.Price} - Category: {product.Category.Name}");

}

**Output**



**LAB 3:**

**RetailStore**

**Models**

**Category.cs**

namespace RetailStore.Models;

public class Category

{

    public int CategoryId { get; set; }

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

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

}

**Product.cs**

namespace RetailStore.Models;

public class Product

{

    public int ProductId { 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!;

}

**Data**

**RetailStoreDbContext.cs**

using Microsoft.EntityFrameworkCore;

using RetailStore.Models;

namespace RetailStore.Data;

public class RetailStoreDbContext : DbContext

{

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

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

    public RetailStoreDbContext(DbContextOptions<RetailStoreDbContext> options)

        : base(options) { }

}

**appsettings.json**

{

  "ConnectionStrings": {

    "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=RetailStoreDb;Trusted\_Connection=True;"

  }

}

**Program.cs**

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Configuration;

using Microsoft.EntityFrameworkCore;

using RetailStore.Data;

var builder = Host.CreateDefaultBuilder(args)

    .ConfigureAppConfiguration((hostingContext, config) =>

    {

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

    })

    .ConfigureServices((context, services) =>

    {

        var configuration = context.Configuration;

        var connectionString = configuration.GetSection("ConnectionStrings")["DefaultConnection"];

        services.AddDbContext<RetailStoreDbContext>(options =>

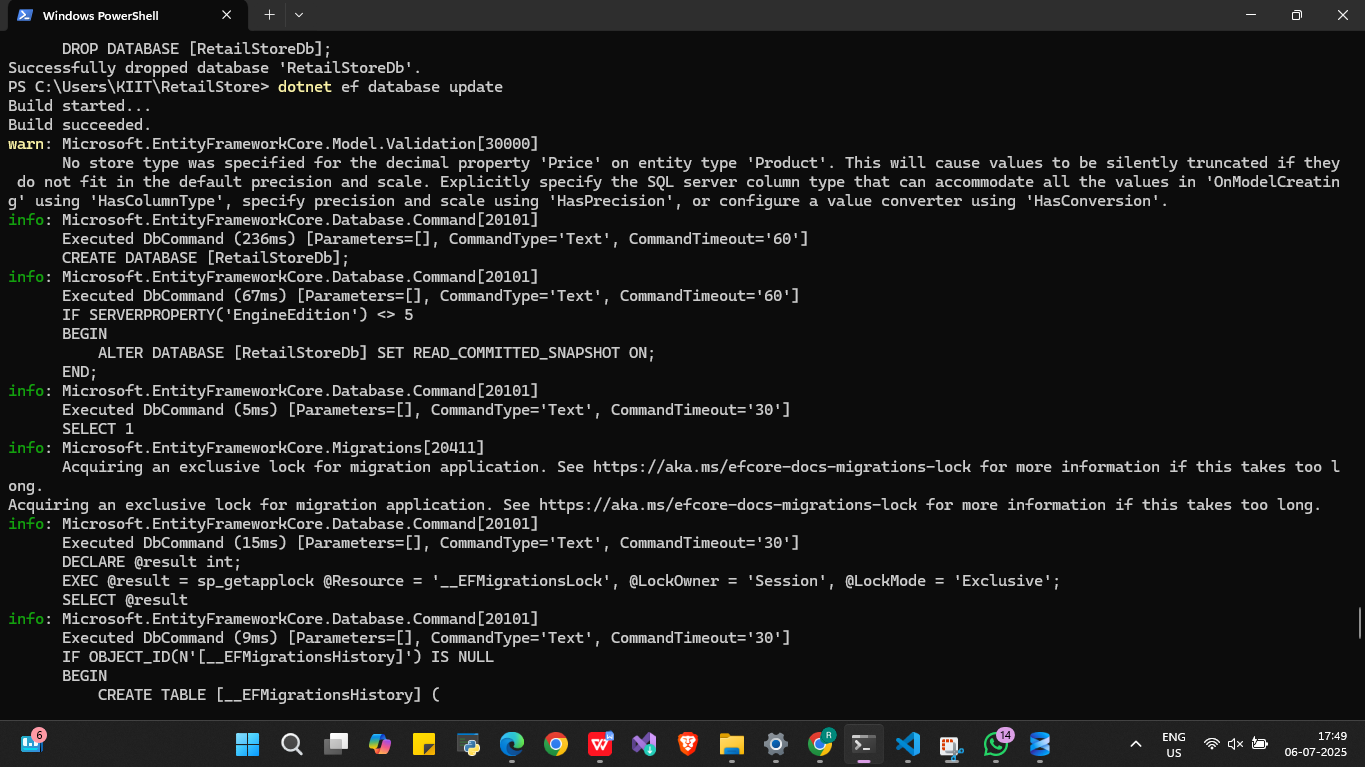
            options.UseSqlServer(connectionString));

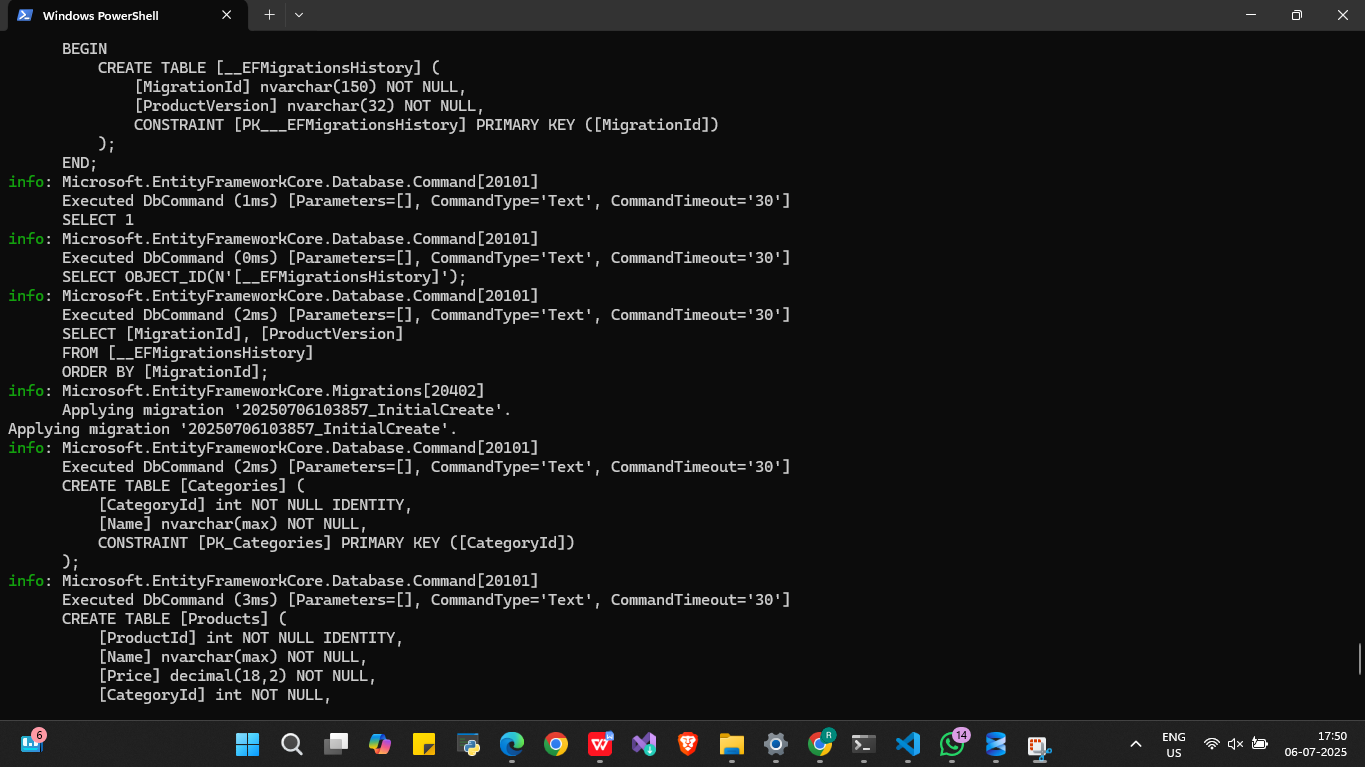
    });

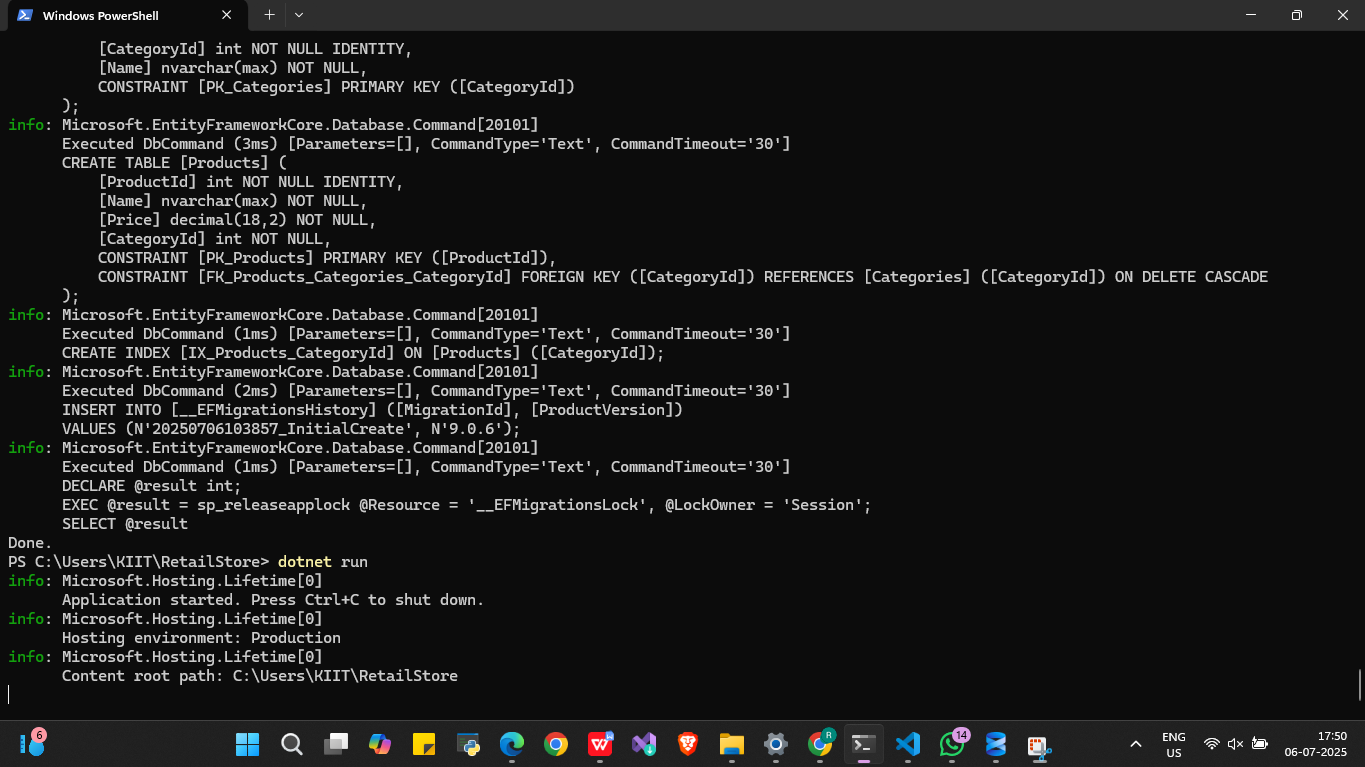
var host = builder.Build();

host.Run();

**Output**

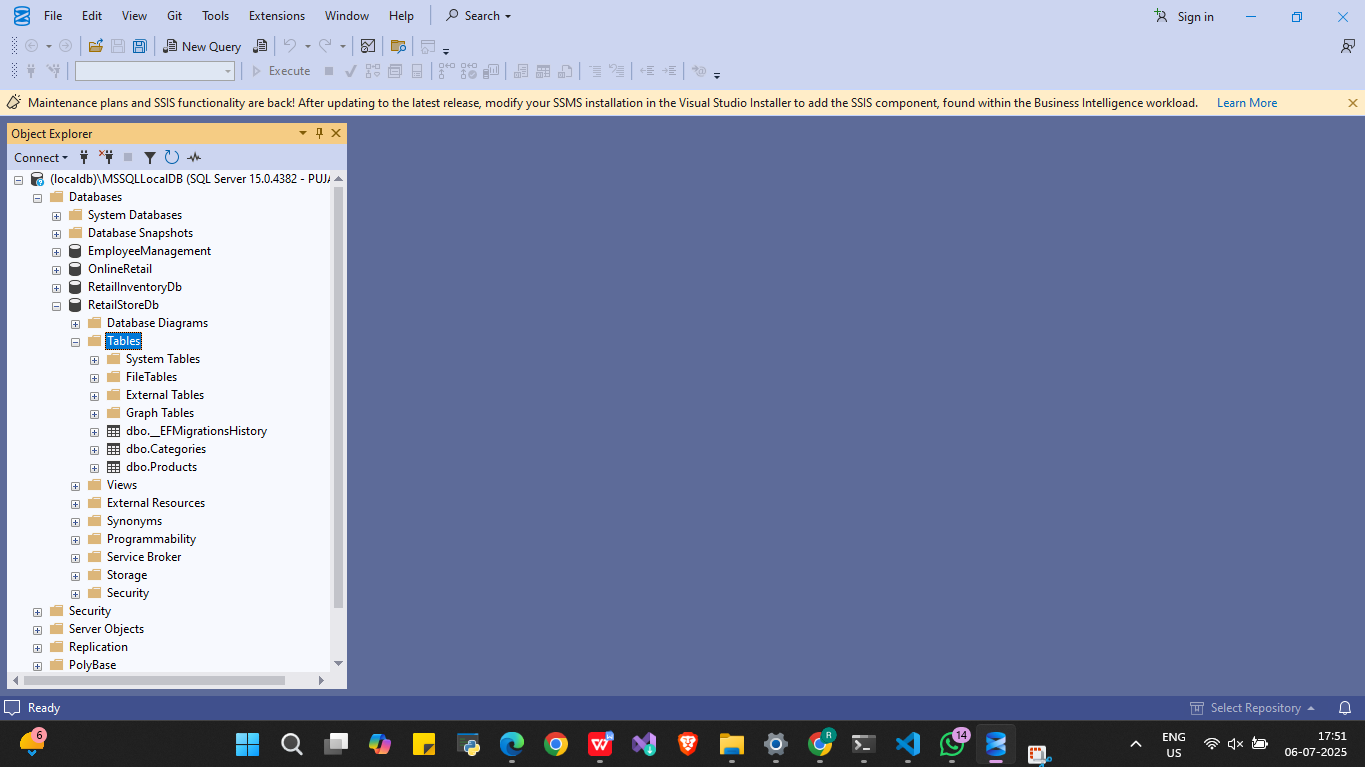






**Step 4:**

**Verified in SQL Server-:**



**LAB 4:**

**RetailStore**

**Models**

**Category.cs**

namespace RetailStore.Models;

public class Category

{

    public int CategoryId { get; set; }

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

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

}

**Product.cs**

namespace RetailStore.Models;

public class Product

{

    public int ProductId { 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!;

}

**Data**

**RetailStoreDbContext.cs**

using Microsoft.EntityFrameworkCore;

using RetailStore.Models;

namespace RetailStore.Data;

public class RetailStoreDbContext : DbContext

{

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

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

    public RetailStoreDbContext(DbContextOptions<RetailStoreDbContext> options)

        : base(options) { }

}

**appsettings.json**

{

  "ConnectionStrings": {

    "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=RetailStoreDb;Trusted\_Connection=True;"

  }

}

**Program.cs**

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Configuration;

using Microsoft.EntityFrameworkCore;

using RetailStore.Data;

using RetailStore.Models;

var builder = Host.CreateDefaultBuilder(args)

    .ConfigureAppConfiguration((hostingContext, config) =>

    {

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

    })

    .ConfigureServices((context, services) =>

    {

        var configuration = context.Configuration;

        var connectionString = configuration.GetSection("ConnectionStrings")["DefaultConnection"];

        services.AddDbContext<RetailStoreDbContext>(options =>

            options.UseSqlServer(connectionString));

    });

var host = builder.Build();

// Call your async method for seeding

await SeedInitialDataAsync(host);

await host.RunAsync();

static async Task SeedInitialDataAsync(IHost host)

{

    using var scope = host.Services.CreateScope();

    var context = scope.ServiceProvider.GetRequiredService<RetailStoreDbContext>();

    // Don't insert if already seeded

    if (!context.Categories.Any())

    {

        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("✅ Seed data inserted successfully.");

    }

    else

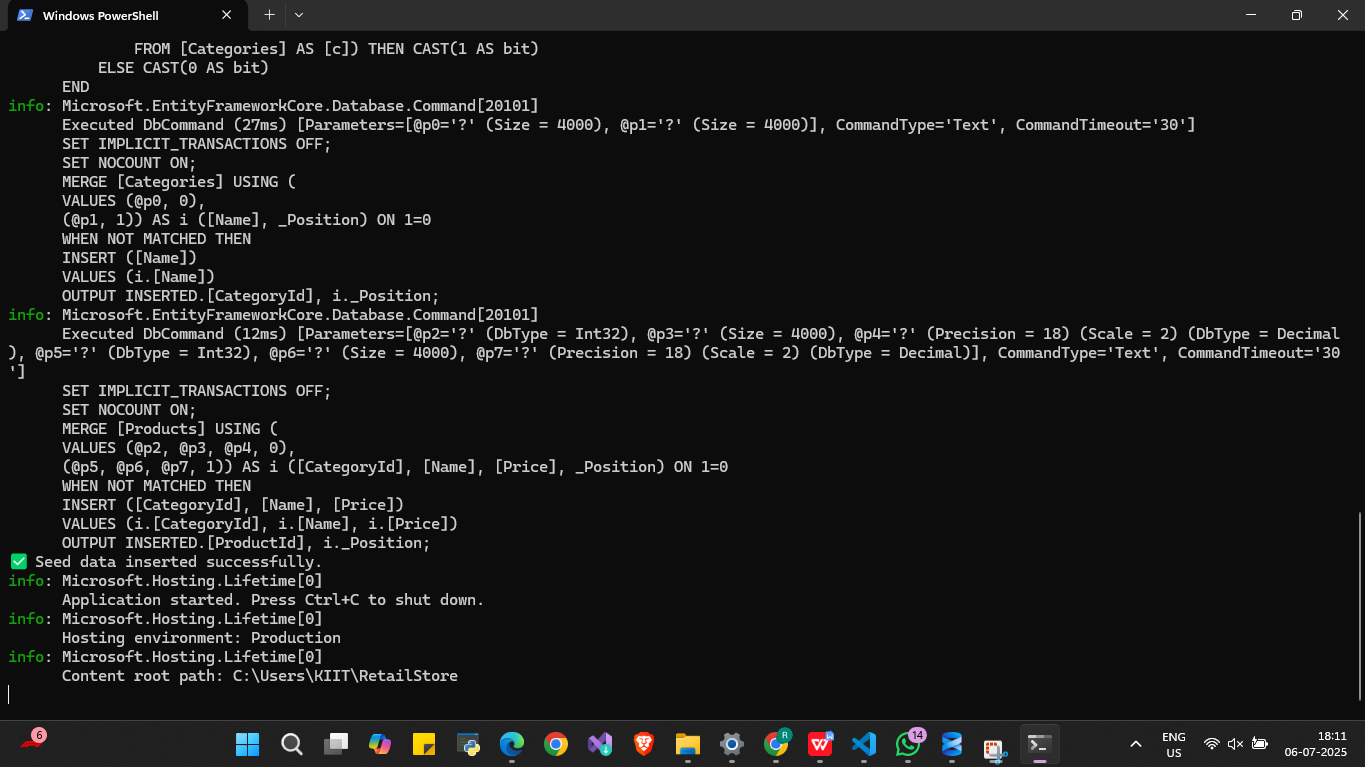
    {

        Console.WriteLine("ℹ️ Data already exists. Skipping seeding.");

    }

}

**Output**



**LAB5:**

**Program.cs**

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Configuration;

using Microsoft.EntityFrameworkCore;

using RetailStore.Data;

using RetailStore.Models;

var builder = Host.CreateDefaultBuilder(args)

    .ConfigureAppConfiguration((hostingContext, config) =>

    {

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

    })

    .ConfigureServices((context, services) =>

    {

        var configuration = context.Configuration;

        var connectionString = configuration.GetSection("ConnectionStrings")["DefaultConnection"];

        services.AddDbContext<RetailStoreDbContext>(options =>

            options.UseSqlServer(connectionString));

    });

var host = builder.Build();

await RetrieveProductDataAsync(host);  // ✅ Add this line to call retrieval method

await host.RunAsync();

static async Task RetrieveProductDataAsync(IHost host)

{

    using var scope = host.Services.CreateScope();

    var context = scope.ServiceProvider.GetRequiredService<RetailStoreDbContext>();

    // ✅ 1. Get all products

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

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

    foreach (var p in products)

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

    // ✅ 2. Find by ID

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

    Console.WriteLine($"\n🔍 Found (ID 1): {product?.Name}");

    // ✅ 3. FirstOrDefault with condition

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

    Console.WriteLine($"\n💰 Expensive Product (>₹50,000): {expensive?.Name}");

}

**Output**