**EF Core 8.0 Guided Hands-On Exercises:-**

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

## STEP 1: What is ORM?

## Imagine we have some ****toys**** (C# classes), and we want to ****store them in a toy box**** (SQL database). But we don’t want to manually write notes (SQL) every time. ****ORM**** (Object-Relational Mapping) helps us connect our toys and the toy box ****automatically!****

### Explanation:

C# classes = tables in the database

Object properties = columns in the table

EF Core does the work of:

Creating tables from classes

Saving and getting data using C# instead of SQL

**Benefits of ORM**:

You write **less SQL**

Code is **easy to understand**

Makes your project **neat and clean**

**Create a .NET Console App:-**

**dotnet new console -n RetailInventory**

**cd RetailInventory**

**Install EF Core Packages:-**

**dotnet add package Microsoft.EntityFrameworkCore.SqlServer**

**dotnet add package Microsoft.EntityFrameworkCore.Design**

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

**STEP 1: Create the Models (Product & Category)**

code .

Create a new file named:

Models.cs

using System.Collections.Generic;

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; } // Foreign Key

public Category Category { get; set; } // Navigation Property

}

· A **Category** (like “Fruits”) has a name and many **Products**.

· A **Product** (like “Apple”) has a name, price, and belongs to **one Category**.

**STEP 2: Create the AppDbContext:-**

In the same folder, create a file named:

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("Your\_Connection\_String\_Here");

}

}

* DbSet<Product> and DbSet<Category> mean:

EF, I want tables for Products and Categories.”

* UseSqlServer(...) is where we’ll **connect to our database.**

**STEP 3: Add a Connection String:-**

### Open SQL Server Management Studio (SSMS)

1. Create a Database

Add this in your AppDbContext.cs

optionsBuilder.UseSqlServer("Server=(localdb)\\MSSQLLocalDB;Database=RetailStoreDB;Trusted\_Connection=True;");

1. Save the File

Program.cs

using System;

class Program

{

static void Main()

{

using (var context = new AppDbContext())

{

Console.WriteLine("Connecting to database...");

var canConnect = context.Database.CanConnect();

Console.WriteLine(canConnect ? "✅ Connected!" : "❌ Failed to connect.");

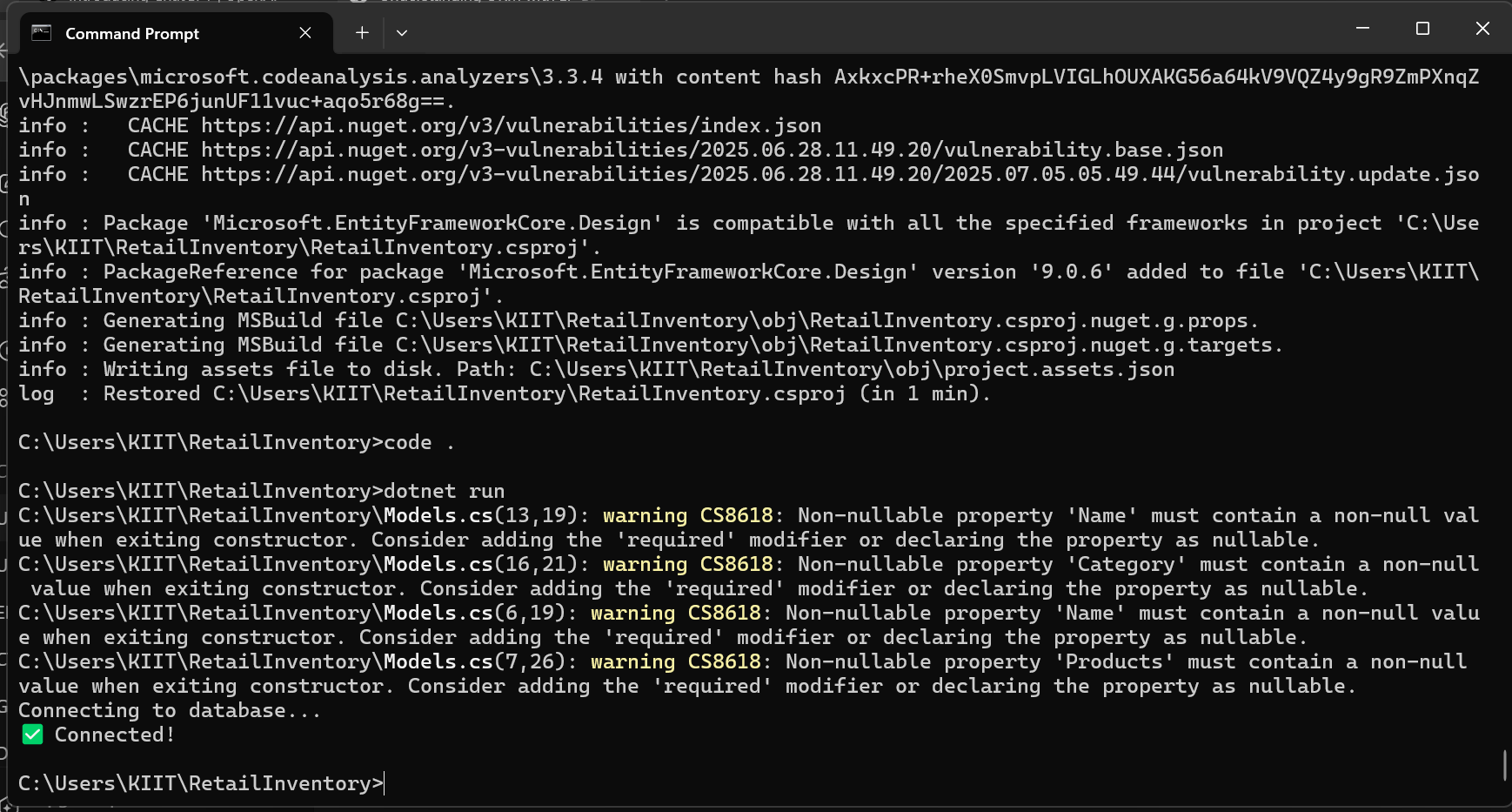
}

}

}

Run it:dotnet run

Output:-



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

**STEP 1: Install EF Core CLI**

dotnet tool install --global dotnet-ef

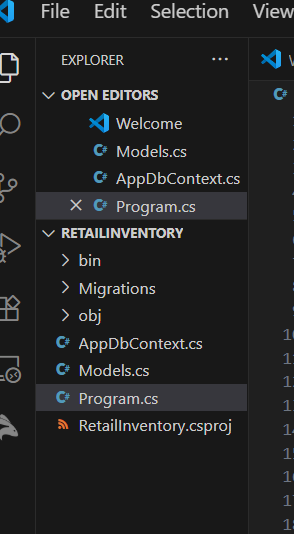
dotnet ef --version

**STEP 2: Create Your First Migration**

dotnet ef migrations add InitialCreate

If successful:

A new Migrations **folder** will appear in your project with .cs files inside



**STEP 3: Apply the Migration**

dotnet ef database update

**STEP 4: Verify in SSMS**

### Steps:

### Open **SSMS**

Connect to your local server:

Expand **Databases**

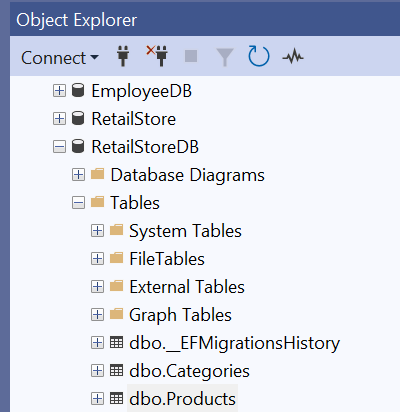
Find RetailStoreDB

Expand:

Tables → We should see:

dbo.Categories

dbo.Products



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

**STEP 1: Modify Program.cs to Add Data**

using System;

using System.Threading.Tasks;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

// Create categories

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

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

// Add categories to the context

await context.Categories.AddRangeAsync(electronics, groceries);

// Create products linked to categories

var product1 = new Product { Name = "Laptop", Price = 75000, Category = electronics };

var product2 = new Product { Name = "Rice Bag", Price = 1200, Category = groceries };

// Add products to the context

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

// Save all changes to the database

await context.SaveChangesAsync();

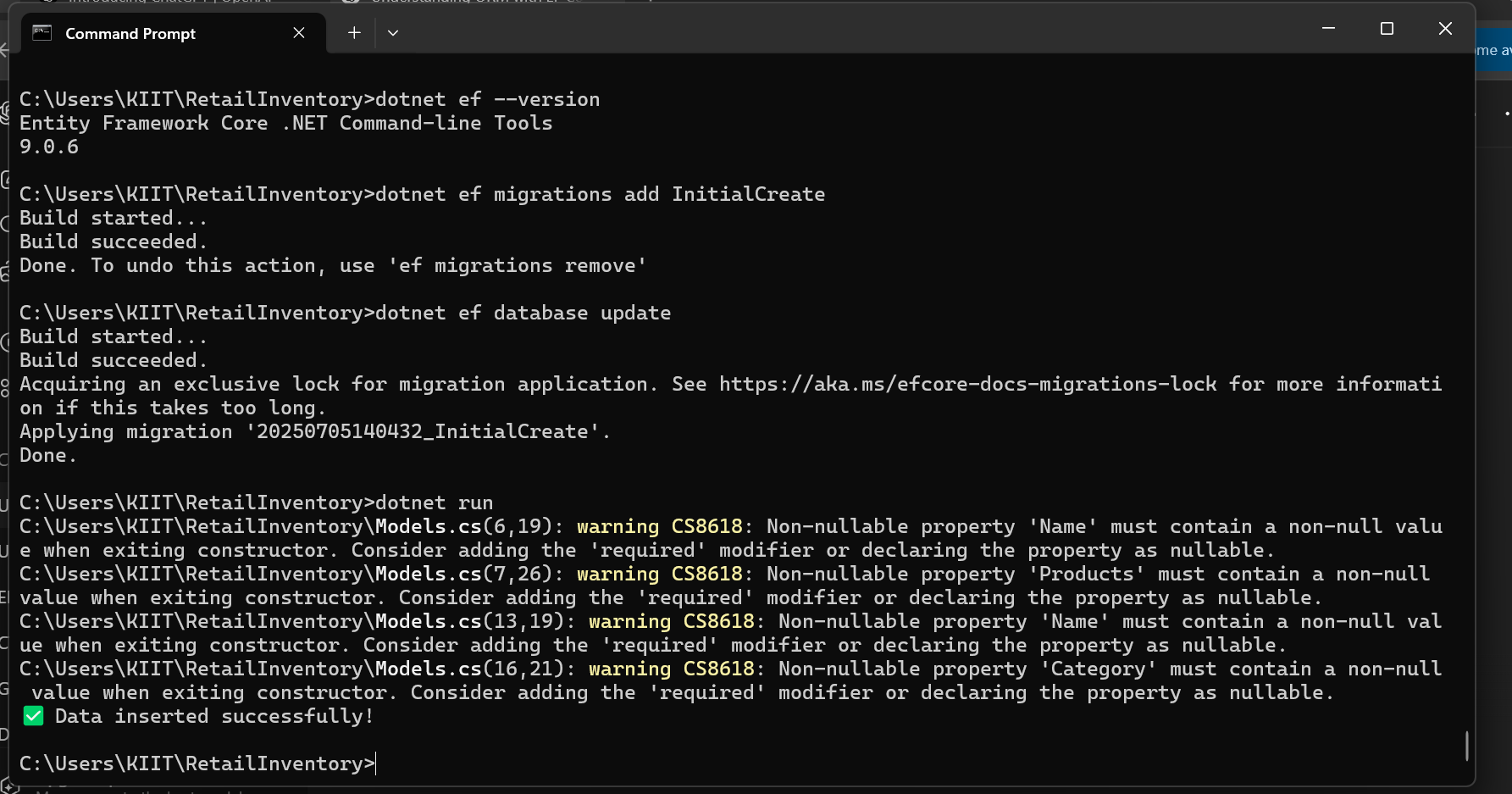
Console.WriteLine("✅ Data inserted successfully!");

}

}

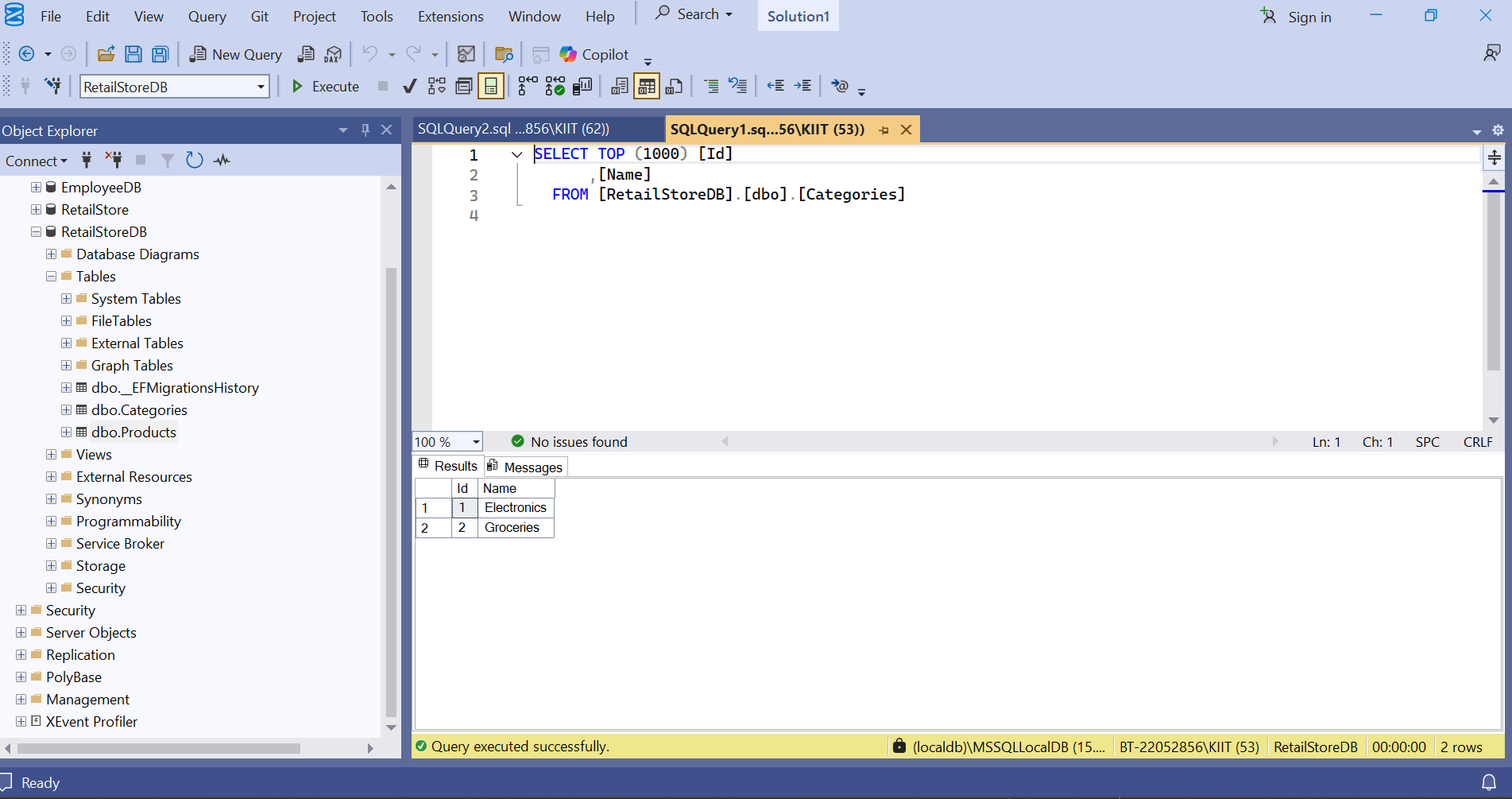
**STEP 2: Run the App**

Run it:-dotnet run

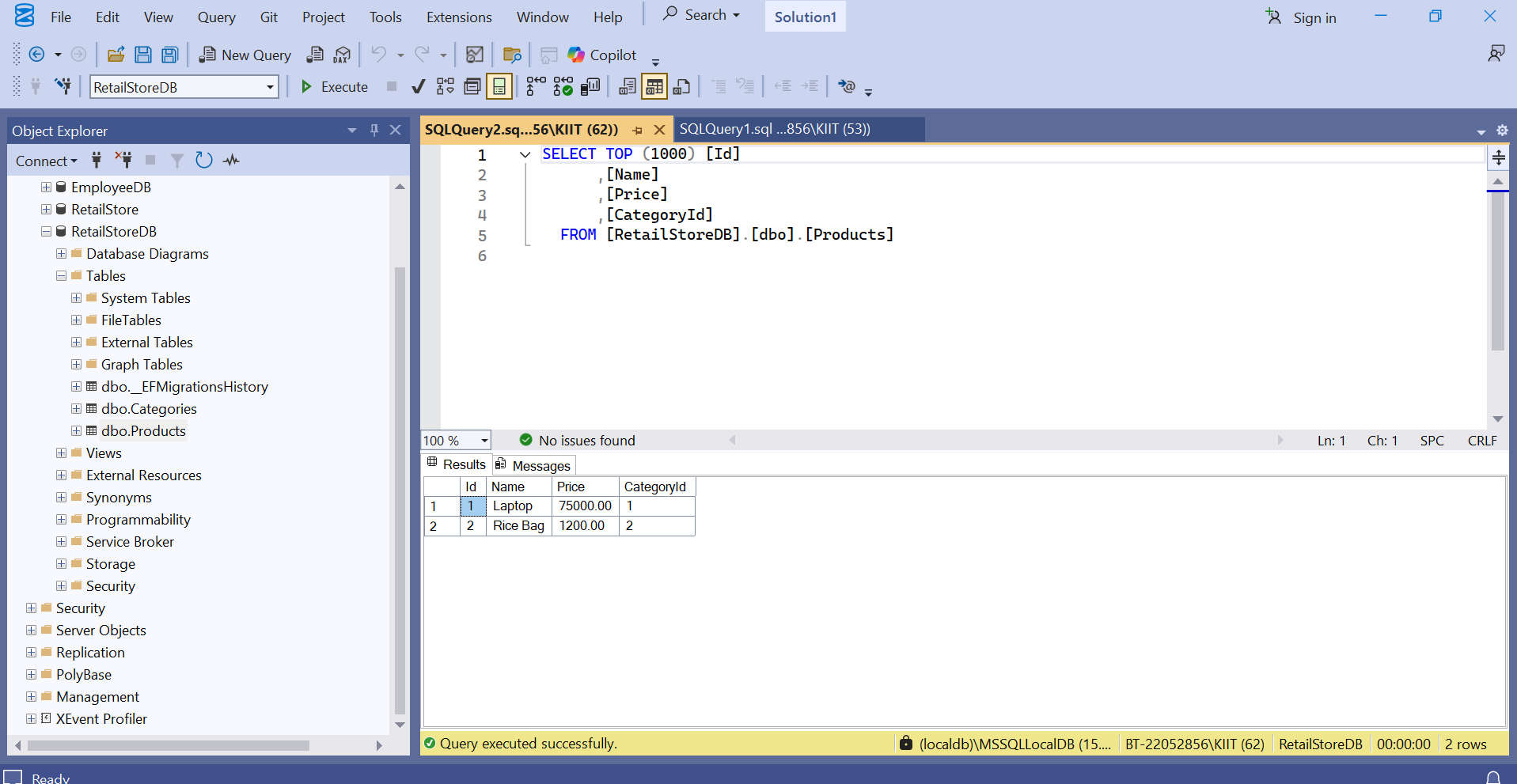
****

**STEP 3: Verify in SQL Server**

dbo.Categories



dbo.Products



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

**Program.cs:-**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main()

{

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 (ID = 1):");

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

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

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

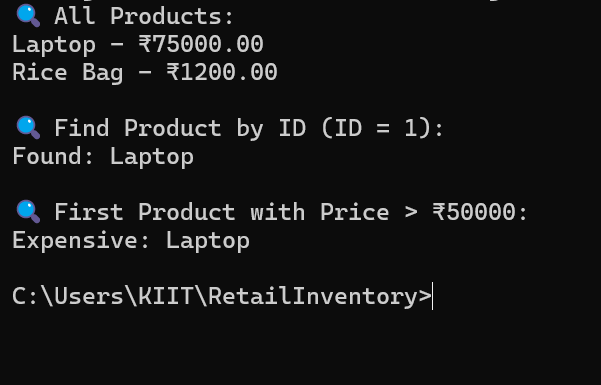
var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);

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

}

}

**Output:-**

****