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

Models

AppDbContext.cs

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models; // Required for recognizing Product and Category

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlite("Data Source=retail.db"); // SQLite connection

}

}

Category.cs

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace RetailInventory.Models

{

public class Category

{

public int CategoryId { get; set; }

[MaxLength(100)] // Fixes the SQLite issue

public string Name { get; set; }

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

}

}

Product.cs

using System.ComponentModel.DataAnnotations;

namespace RetailInventory.Models

{

public class Product

{

public int ProductId { get; set; }

[MaxLength(100)] // Fixes the SQLite issue

public string Name { get; set; }

public int StockLevel { get; set; }

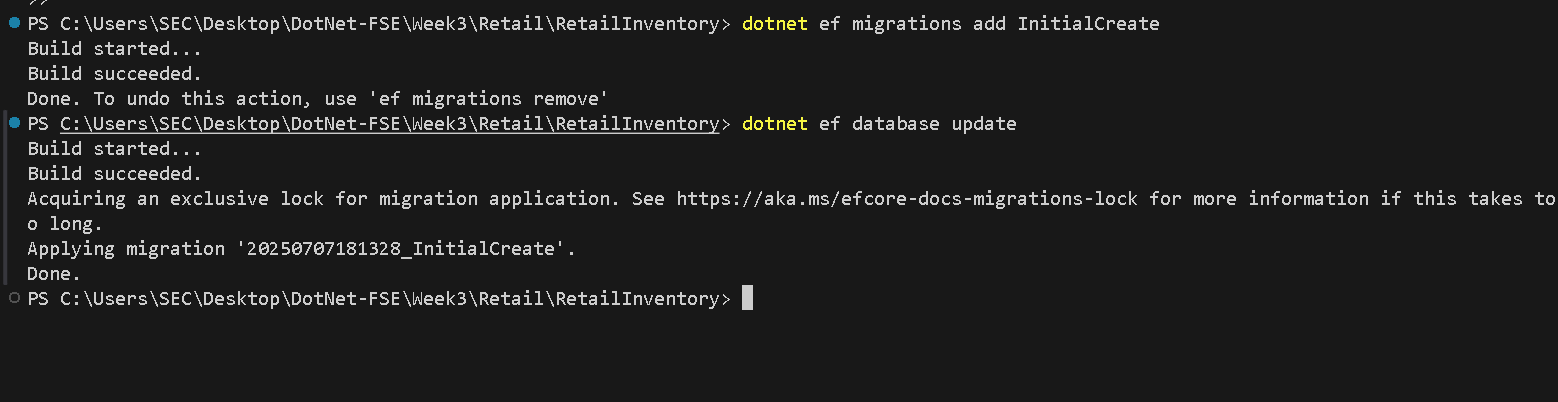
public int CategoryId { get; set; }

public Category Category { get; set; }

}

}

OUTPUT:

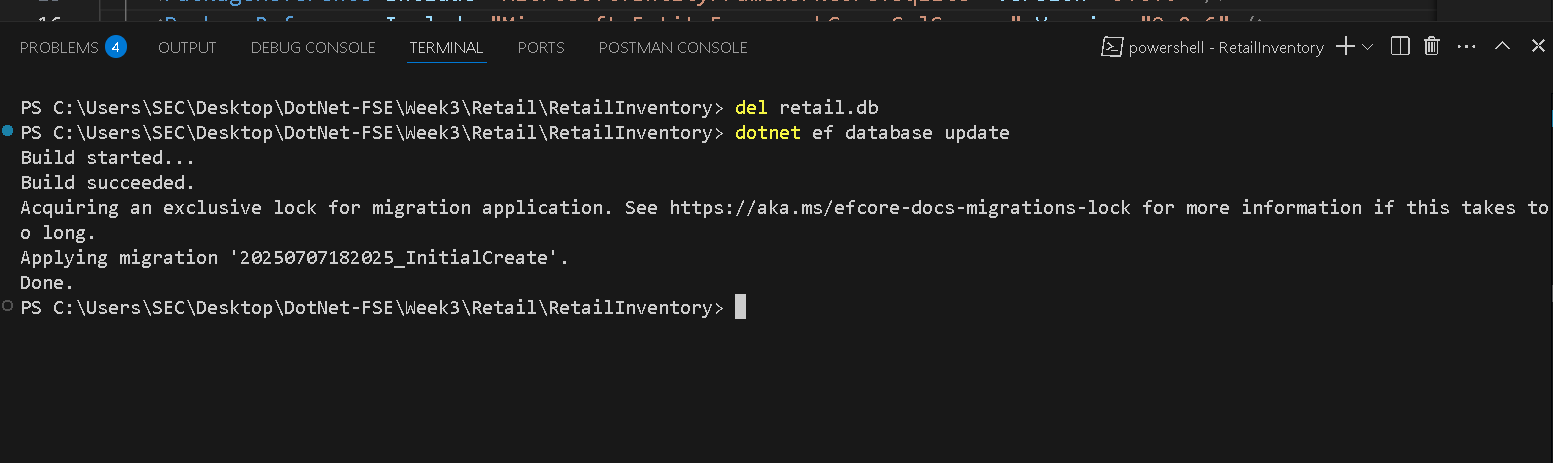


**Lab 2: Setting Up the Database Context for a Retail Store**

**Scenario:**

The retail store wants to store product and category data in SQL Server.

OUTPUT:



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

**Scenario:**

The retail store's database needs to be created based on the models you've defined.

You’ll use EF Core CLI to generate and apply migrations

Models

Product.cs

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; }

}

Category.cs

using System.Collections.Generic;

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

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)

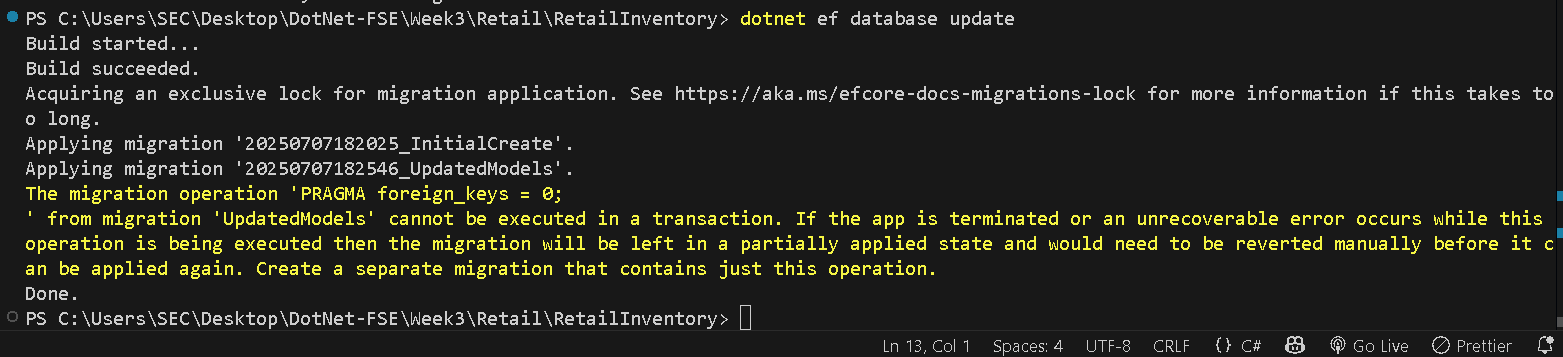
{

optionsBuilder.UseSqlite("Data Source=retail.db");

}

}

OUTPUT:



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

**Scenario:**

The store manager wants to add initial product categories and products to the system

Category.cs

public class Category

{

public int CategoryId { get; set; }

public string Name { get; set; }

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

}

Product.cs

public class Product

{

public int ProductId { get; set; }

public string Name { get; set; }

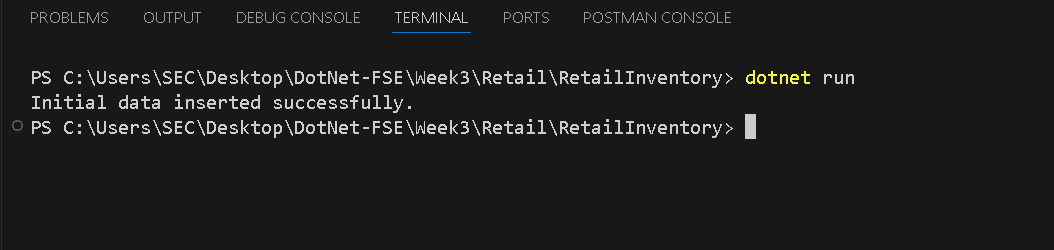
public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

}

OUTPUT:



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

**Scenario:**

The store wants to display product details on the dashboard.

**Objective:**

Use Find, FirstOrDefault, and ToListAsync to retrieve data.

**Steps:**

**1. Retrieve All Products:**

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

foreach (var p in products)

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

**2. Find by ID:**

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

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

**3. FirstOrDefault with Condition:**var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 5000

0);

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

Program.cs

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

// 🔹 1. Retrieve all products

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

Console.WriteLine(" All Products:");

foreach (var p in products)

{

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

}

// 🔹 2. Find product by ID

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

Console.WriteLine($"\n Found by ID (1): {product?.Name ?? "Not found"}");

// 🔹 3. Get first product with price > ₹50,000

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

Console.WriteLine($"\n First Expensive Product (> ₹50000): {expensive?.Name ?? "Not found"}");

}

}

OUTPUT:

