EF Core 8.0 Guided Hands-On Exercises

Lab 1:Understanding ORM With a Retail Inventory Sysytem

**1. What is ORM?**

Object-Relational Mapping (ORM) is a programming technique used to convert data between incompatible type systems—in this case, between C# objects and relational database tables.

**Benefits:**

**Productivity:** Write C# code instead of SQL for most data operations.

**Maintainability:** Changes in the model are reflected in the database via migrations.

**Abstraction:** Developers work with objects, not raw SQL, reducing errors and complexity.

**2. EF Core vs EF Framework:**

* **EF Core:**
  1. Cross-platform (.NET 6/7/8, Windows, Linux, macOS)
  2. Lightweight, modular, and supports modern features (LINQ, async, compiled queries)
  3. Actively developed and recommended for new projects
* **EF Framework (EF6):**
  1. Windows-only (.NET Framework)
  2. More mature, but less flexible and not cross-platform
  3. Lacks some modern features

**3. EF Core 8.0 Features:**

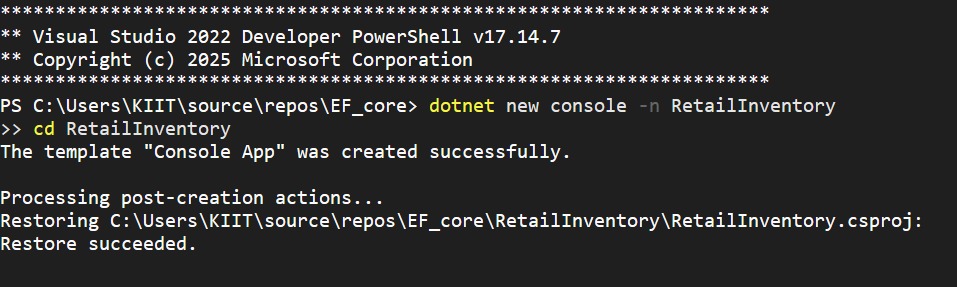
**JSON column mapping:** Store and query JSON data directly in SQL Server columns.

**Compiled models:** Faster startup and query execution.

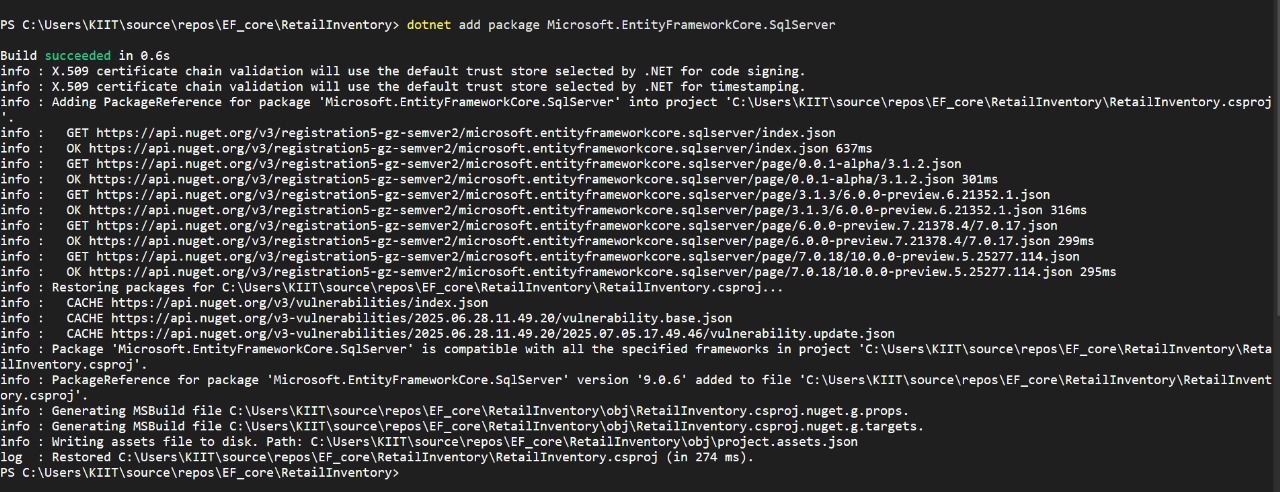
**Interceptors:** Hook into EF Core’s operations for logging, validation, etc.

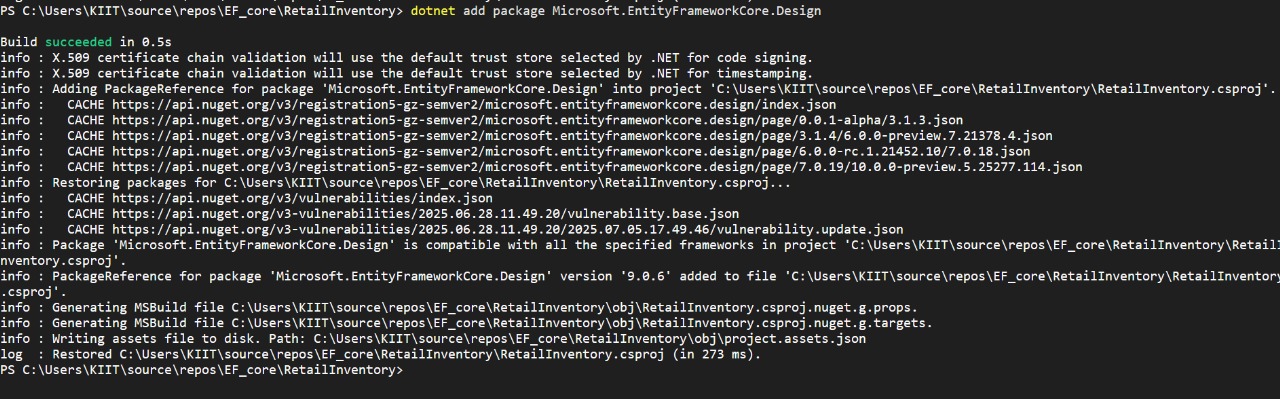
**Bulk operations:** Improved support for batch inserts, updates, and deletes.

**Create a .NET Console App:**



**Install EF Core Packages:**

****

****

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

**Create Models:**

using System.Collections.Generic;

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

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; }

}

**Create AppDbContext:**

using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext

{

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

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

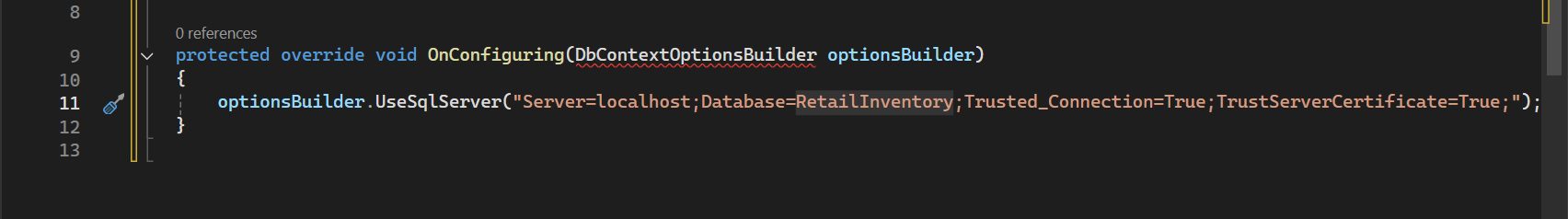
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("Your\_Connection\_String\_Here");

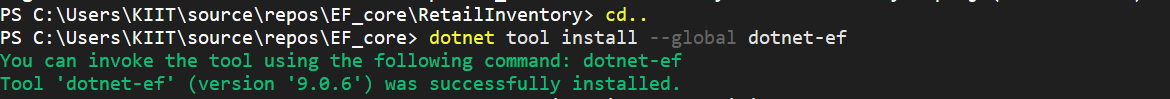
Server=(localdb)\\mssqllocaldb;Database=RetailInventoryDb;Trusted\_Connection=True;

**Add Connection String in appsettings.json (optional for ASP.NET Core).**



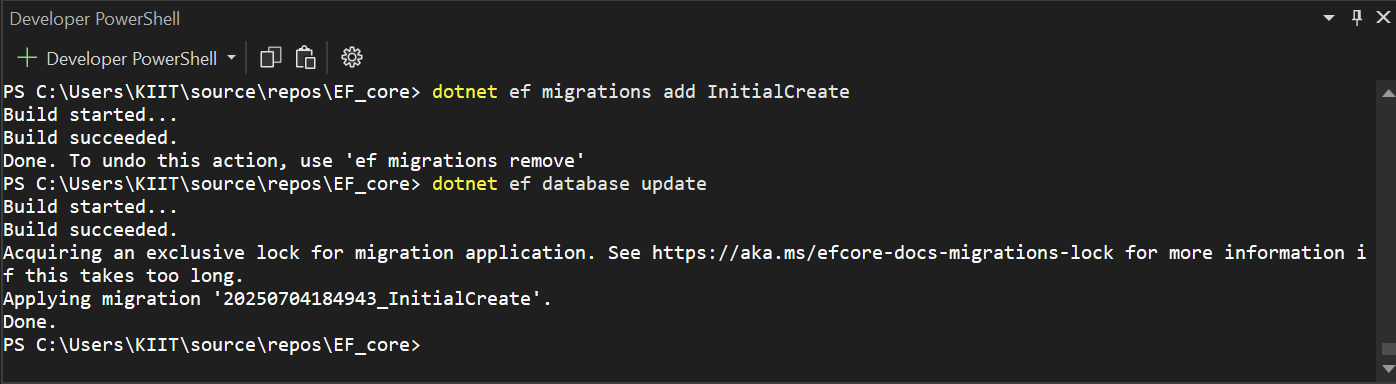
# Lab 3: Using EF Core CLI to Create and Apply Migrations

**Install EF Core CLI**

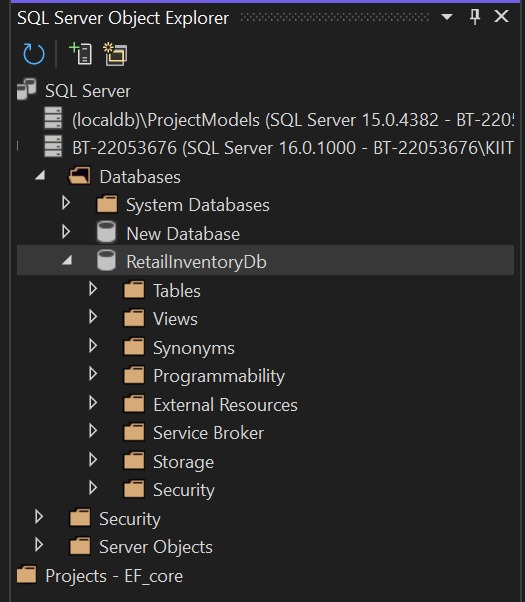
****

**Create Initial Migration**

**Apply Migration to Create Database:**



**Verify in SQL Server:**



# Lab 4: Inserting Initial Data into the Database

**Insert Data in Program.cs:**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

// Ensure database is created (optional safety check)

await context.Database.EnsureCreatedAsync();

// Add initial categories

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

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

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

// Add initial products

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);

// Save to database

await context.SaveChangesAsync();

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

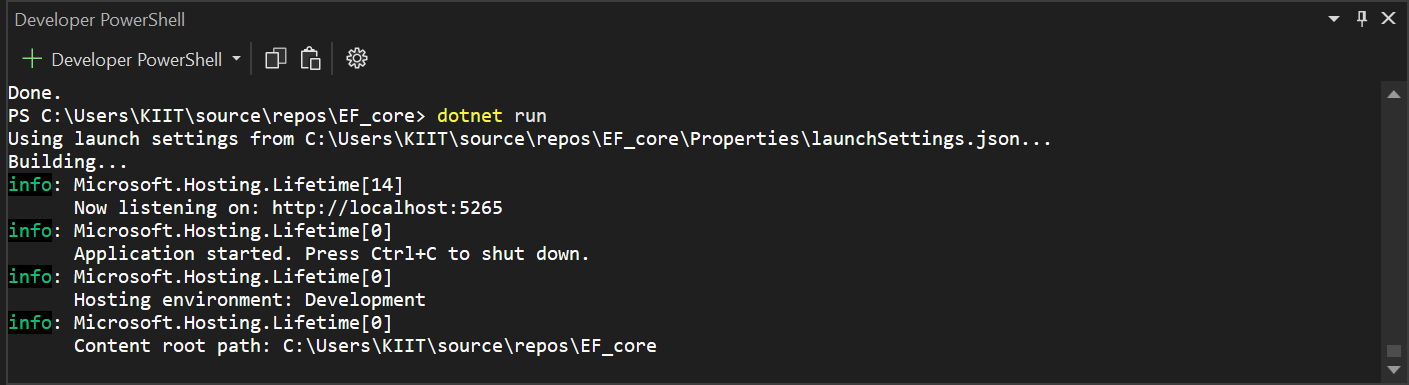
}

}

SELECT \* FROM Categories;

SELECT \* FROM Products;

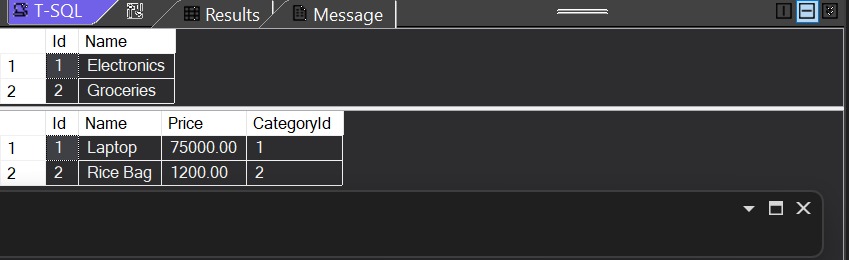
**Run the App:**

****

**Verify in SQL Server:**

SELECT \* FROM Categories;

SELECT \* FROM Products;



# Lab 5: Retrieving Data from the Database

**Retrieve All Products:**

**Find by ID:**

**FirstOrDefault with Condition**

**Insert Data in Program.cs:**

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

var hasData = await context.Products.AnyAsync();

if (!hasData)

{

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

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

var p1 = new Product { Name = "Laptop", Price = 75000, Category = electronics };

var p2 = new Product { Name = "Rice Bag", Price = 1200, Category = groceries };

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

await context.Products.AddRangeAsync(p1, p2);

await context.SaveChangesAsync();

}

Console.WriteLine("All Products:");

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

foreach (var p in products)

{

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

}

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

Console.WriteLine("Found by ID: " + (found != null ? found.Name : "Not Found"));

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

Console.WriteLine("Expensive Product: " + (expensive != null ? expensive.Name : "None"));

}

}

