**Cognizant Deep Skilling Week-3**

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

## ****What is** ORM (Object-Relational Mapping)?**

## An ORM is a library that automatically translates between objects in your application code (e.g., Product, Category classes) and rows in relational database tables (e.g., Products, Categories). Instead of hand-writing SQL for every CRUD operation, you work with strongly-typed C# objects; the ORM takes care of generating and executing the SQL behind the scenes.

**Benefits:-**

**Productivity** – Write far less boiler-plate SQL; let IntelliSense and compiler catch errors.

**Maintainability** – Schema changes mostly require changing class definitions and running a migration.

**Abstraction** – You think in terms of objects and LINQ queries, not SELECT, INSERT, JOIN strings.

## 2. EF Core vs. Entity Framework 6:-

| **Feature** | **EF Core 8.0** | **Entity Framework 6 (EF6)** |
| --- | --- | --- |
| Platform | Windows, Linux, macOS | Windows-only |
| .NET versions | .NET 6/7/8, ASP.NET Core | .NET Framework 4.x |
| Performance | Faster (compiled models, no tracking optimizations) | Slower, older SQL pipeline |
| Modern APIs | Full async/await LINQ, IAsyncEnumerable, ExecuteUpdate/Delete | Limited async, no bulk ops |
| Extensibility | Interceptors, value converters, provider plug-ins | Fewer hooks, rigid provider model |
| Maturity | Newer; still adding features | Very stable, feature-complete |
| Use when… | Cross-platform apps, micro-services, cloud-native workloads | Legacy WinForms/WPF, mature LOB apps tied to .NET Framework |

## 3. Headline Features in EF Core 8.0:-

**JSON column mapping** – Store complex objects or lists in a SQL Server 2022 NVARCHAR(max)/JSON column and query them natively with LINQ.

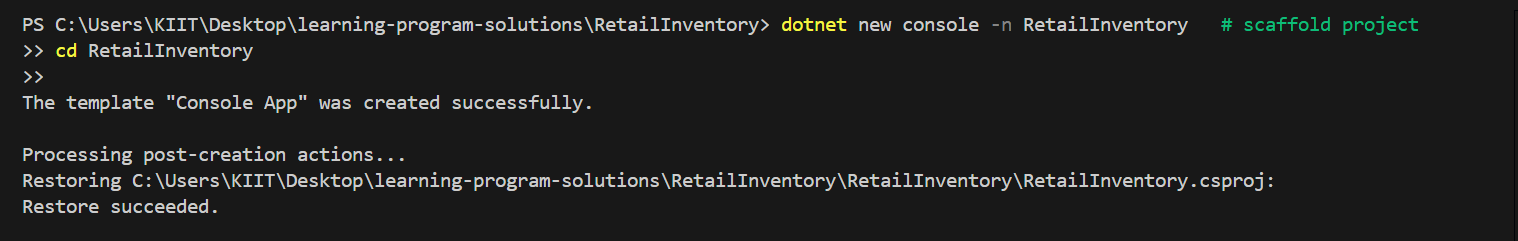
**Compiled models (AOT)** – Pre-generate the model at build time; cold-start and first query run 2-5× faster.

**Better bulk operations** – ExecuteUpdateAsync / ExecuteDeleteAsync send set-based SQL rather than loading entities one-by-one.

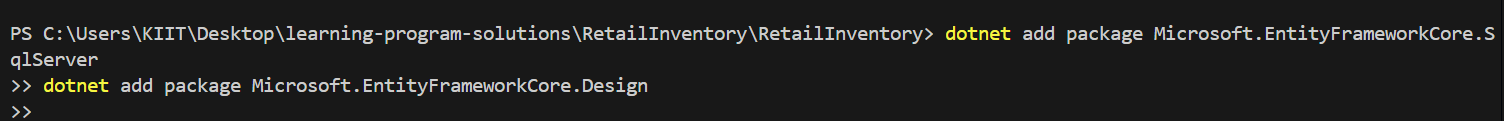
**Interceptors & logging** – Hook into SaveChanges, CommandExecution, etc., for auditing or multitenancy.

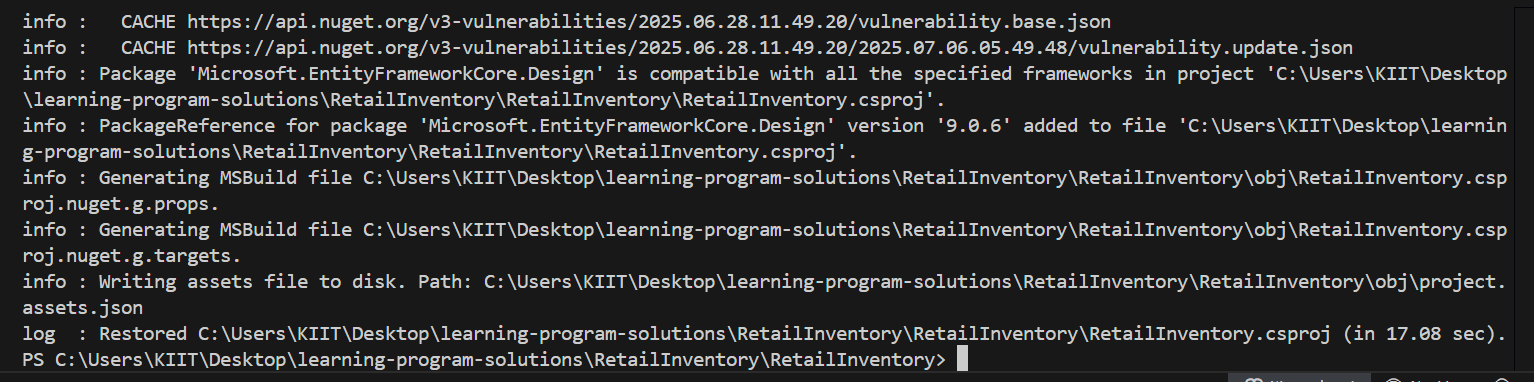
**Improved scaffolding & migrations** – Cleaner diffing, nullable reference-type awareness.

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



1. **Add EF Core Packages:-**





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

RetailInventory:

Category.cs:

namespace RetailInventory.Models;

public class Category

{

public int Id { get; set; }

public string Name { get; set; } = null!;

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

}

Product.cs:

namespace RetailInventory.Models;

public class Product

{

public int Id { get; set; }

public string Name { get; set; } = null!;

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; } = null!;

}

AppDbContext.cs:

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory;

public class AppDbContext : DbContext

{

public DbSet<Product> Products => Set<Product>();

public DbSet<Category> Categories => Set<Category>();

protected override void OnConfiguring(DbContextOptionsBuilder b)

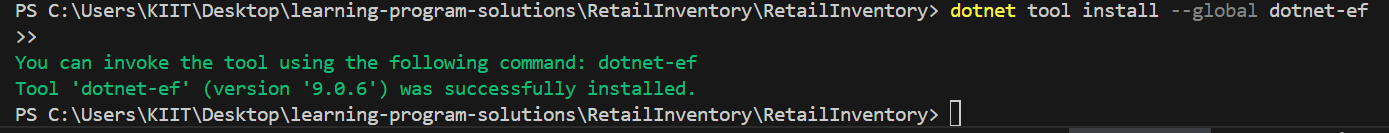
=> b.UseSqlServer(

"Server=localhost;Database=RetailDb;Trusted\_Connection=True;TrustServerCertificate=True");

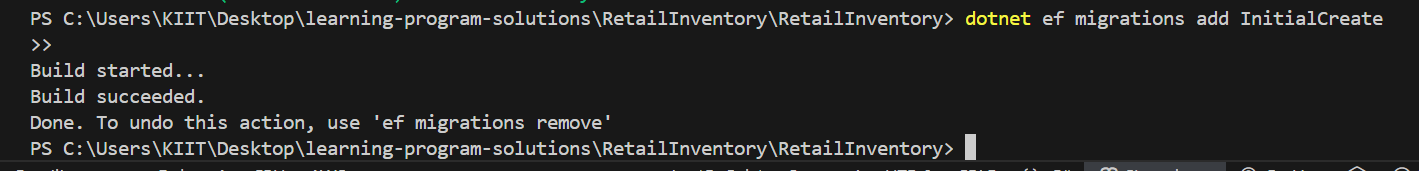
}

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

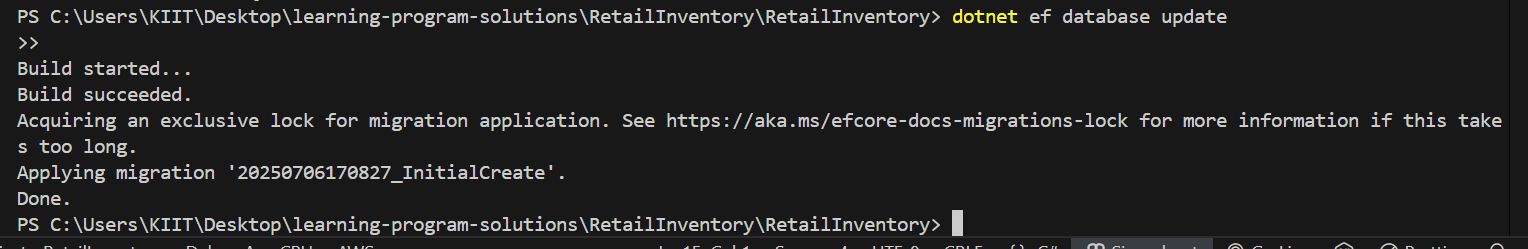
**1. Install EF Core CLI (if not already):**



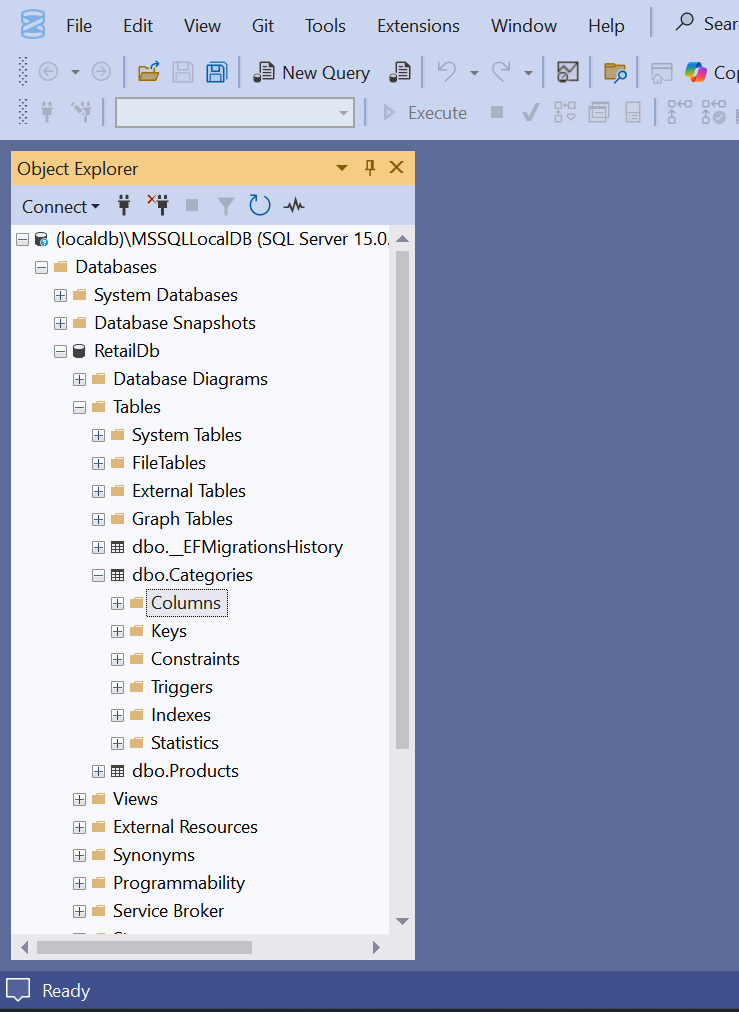
**2. Create Initial Migration:**



**3. Apply Migration to Create Database:**



**4. Verify in SQL Server:**



Lab 4: Inserting Initial Data into the Database:-

1. **Insert Data in Program.cs:**

using Microsoft.EntityFrameworkCore;

using RetailInventory;

using RetailInventory.Models;

using var ctx = new AppDbContext();

await ctx.Database.EnsureCreatedAsync();

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

{

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

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

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

await ctx.Products.AddRangeAsync(

new Product { Name = "Laptop", Price = 75\_000m, Category = electronics },

new Product { Name = "Rice Bag", Price = 1\_200m, Category = groceries });

await ctx.SaveChangesAsync();

Console.WriteLine("✔ Seeded initial data.");

}

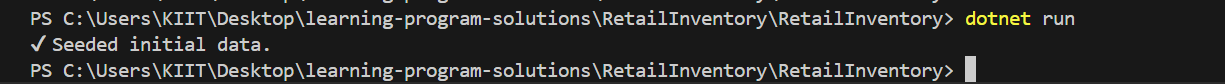
else

{

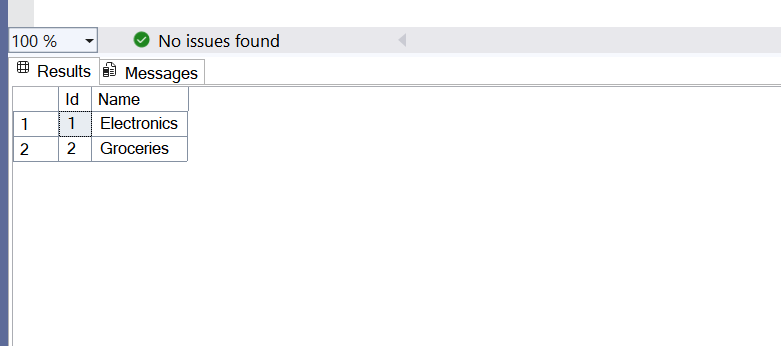
Console.WriteLine("Database already seeded.");

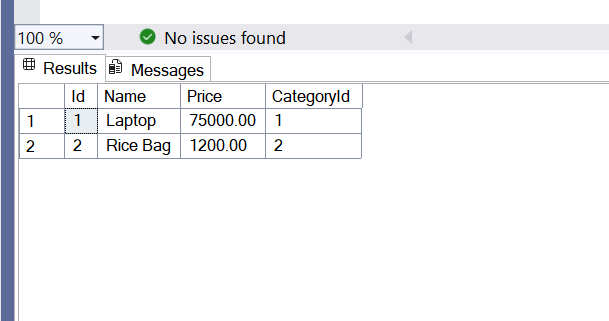
}

1. **Run the App:**



3. Verify in SQL Server:





Lab 5: Retrieving Data from the Database:-

1. **Retrieve All Products:**

var all = await ctx.Products

.Include(p => p.Category)

.ToListAsync();

all.ForEach(p => Console.WriteLine($"{p.Name} ({p.Category.Name}) - ₹{p.Price:N0}"));

var laptop = await ctx.Products.FindAsync(1);

Console.WriteLine(laptop is null ? "Not found" : $"Found: {laptop.Name}");

var pricey = await ctx.Products

.Where(p => p.Price > 50\_000m)

.OrderByDescending(p => p.Price)

.FirstOrDefaultAsync();

Console.WriteLine(pricey is null ? "Nothing pricey" : $"Expensive: {pricey.Name}");

**OUTPUT:-**

