**LAB1:Understand ORM with Retail Inventory System**

1. **What is ORM?**

**ORM (Object-Relational Mapping)** is a technique that connects object-oriented programming languages like C# with relational databases like SQL Server.

It **maps C# classes to database tables**, and properties to columns.

With ORM, developers can **interact with the database using C# objects** instead of writing raw SQL queries.

**Benefits:**

* **Abstraction** from SQL – less manual query writing
* **Productivity** – faster development with fewer lines of code
* **Maintainability** – changes in schema can be reflected in code using migrations
* **Type-safety and IntelliSense** – compile-time checks reduce runtime errors.

1. **EF Core vs EF Framework:**

| **Feature** | **EF Core** | **EF Framework (EF6)** |
| --- | --- | --- |
| Platform | Cross-platform (.NET Core) | Windows-only (.NET Framework) |
| Performance | Faster, optimized | Slower in some cases |
| Features | Supports LINQ, async/await, interceptors | Lacks some modern features |
| Migrations | Built-in and streamlined | Supported |
| JSON Support | Yes (EF Core 8 has JSON mapping) | No |
| Flexibility | Modular, lightweight | Heavy and older architecture |

1. **EF Core 8.0 Features:**

· **JSON Column Mapping**  
 Store complex objects directly in JSON columns.

· **Compiled Models**  
Improve startup and query performance by pre-generating models.

· **Interceptors**  
 Hook into EF Core operations to log, modify, or block certain actions.

· **Bulk Operations Support**  
Improved integration with libraries like EFCore.BulkExtensions.

**LAB2:Setting up Database Context**

1. **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; }

}

1. **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)

{

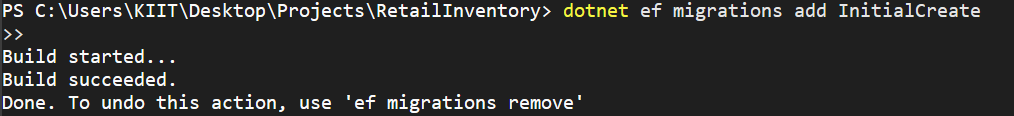
optionsBuilder.UseSqlServer("Server=localhost\\SQLEXPRESS;Database=RetailInventoryDB;Trusted\_Connection=True;TrustServerCertificate=True;");

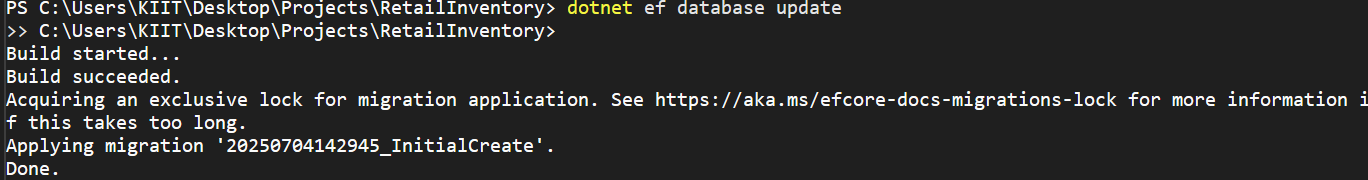
}

}

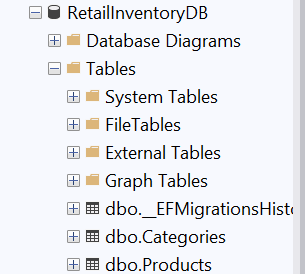
**LAB3:Using EF Core CLI**

**Building migration:**





**SSMS:**



**LAB4:Inserting Initial data into database**

**Inserting Data:**

using System;

using System.Threading.Tasks;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

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

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

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

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

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

await context.Products.AddRangeAsync(laptop, rice);

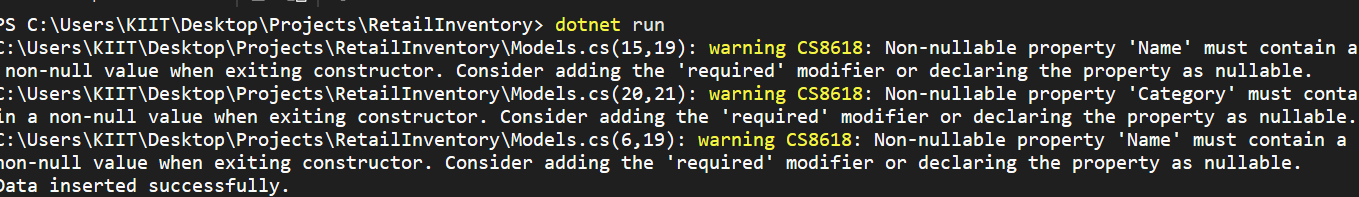
await context.SaveChangesAsync();

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

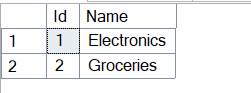
}

}

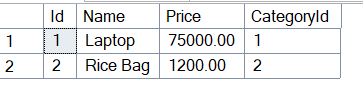
**Executing code:**



**Table ‘Category’:**

****

**Table ‘Product’:**



**LAB5: Retrieving data from Database**

1. Retrieve all products
2. Find by ID
3. First or default with condition

**Program.cs**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

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 (1):");

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

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

Console.WriteLine("\n First Product with Price > ₹50,000:");

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

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

}

}

**Output:**

