**LAB 1**

**1.What is ORM?**

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

**How ORM Maps C# Classes to Database Tables**:

* C# Classes ➜ Database Tables
* Class Properties ➜ Table Columns
* Objects ➜ Rows (Records)
* Relationships between classes (navigation properties) map to foreign keys.

**Benefits of ORM**

| **Benefit** | **Description** |
| --- | --- |
| **Productivity** | Automatically handles SQL commands like INSERT, UPDATE, DELETE. |
| **Maintainability** | Changes in C# models easily reflect in the DB via migrations. |
| **Abstraction from SQL** | You work with objects instead of writing SQL queries manually. |

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

| **Feature** | **EF Core** | **EF Framework (EF6)** |
| --- | --- | --- |
| **Platform** | Cross-platform (.NET Core & .NET 5+) | Windows-only (.NET Framework) |
| **Lightweight** | Yes | No |
| **Performance** | Faster in many cases | Slower for large models |
| **Features** | Async LINQ, No-tracking queries, Compiled queries | Mature but lacks modern features |
| **Flexibility** | Highly extensible | Limited extensibility |

1. **EF Core 8.0 Features:**

* **JSON Column Mapping:**

Store complex object structures inside a single JSON column (for SQL Server 2022+).

* **Compiled Models:**

Pre-compiled models to improve startup and query performance.

* **Interceptors:**

Hook into EF Core operations to log or modify behaviors.

* **Improved Bulk Operations:**

Better support for bulk inserts/updates.

**LAB 2**

**Product.cs**

namespace RetailInventorySystem.Data

{

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**

namespace RetailInventorySystem.Data

{

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

}

**AppDBContext.cs**

using Microsoft.EntityFrameworkCore;

using RetailInventorySystem.Data;

public class AppDbContext : DbContext

{

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

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

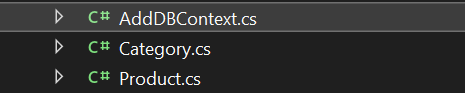
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

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

}

}



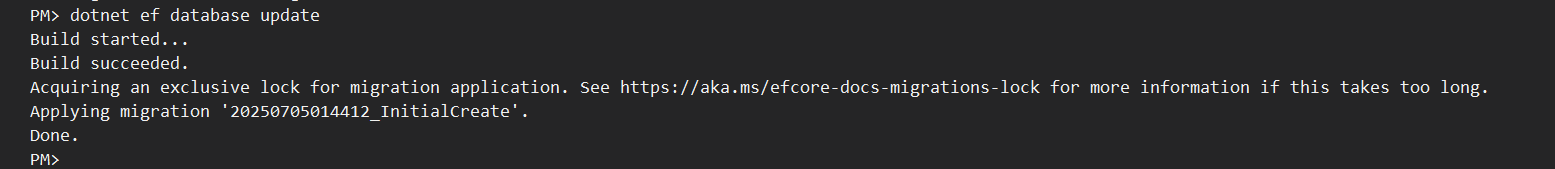
**LAB 3**

A black screen with white text

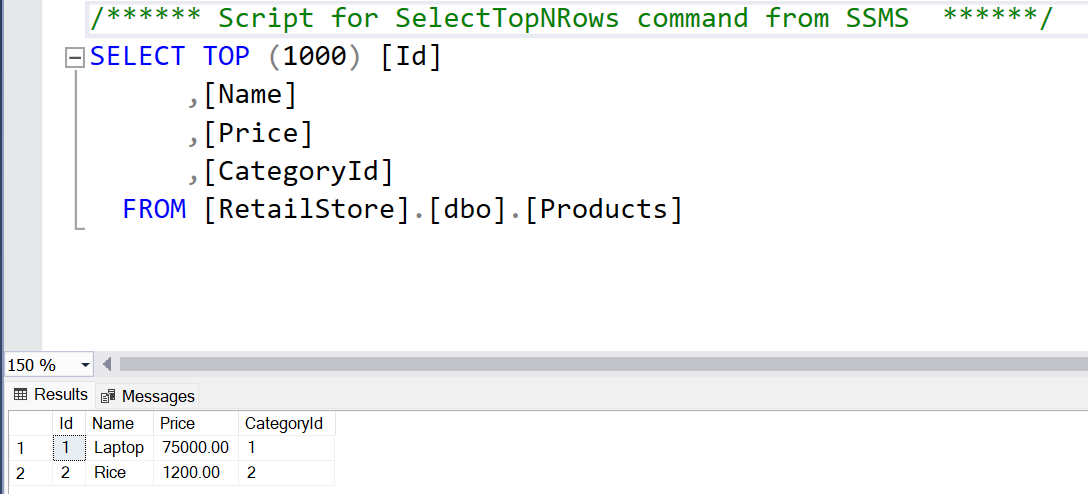
AI-generated content may be incorrect.

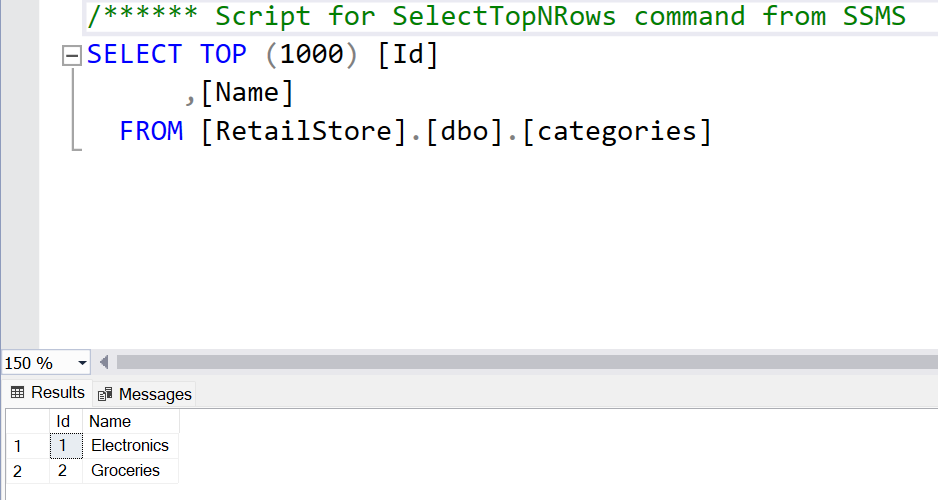
A screenshot of a computer

AI-generated content may be incorrect.



Snapshots of SQL server Management Studio:





**LAB 4**

**Product.cs**

using RetailInventorySystem.Data;

using System;

using System.Linq;

using System.Threading.Tasks;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

// Prevent duplicate seeding

if (!context.Categories.Any() && !context.Products.Any())

{

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

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

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

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);

await context.SaveChangesAsync();

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

}

else

{

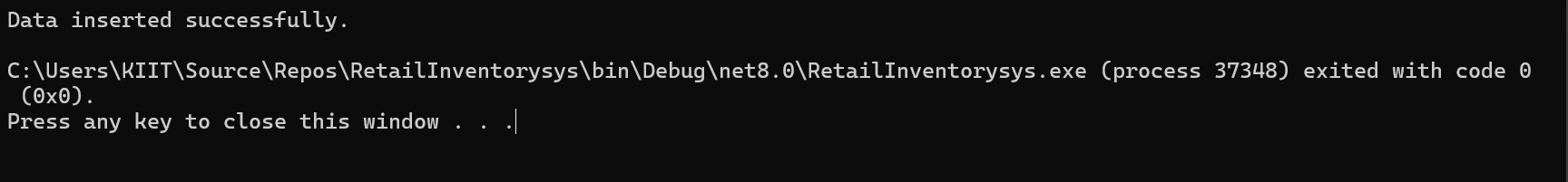
Console.WriteLine("Data already exists. Skipping seeding.");

}

}

}

**Output:**



**LAB 5**

**Updated Program.cs**

using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

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

foreach (var p in products)

Console.WriteLine($"{p.Name} - INR{p.Price} - {p.Category?.Name}");

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

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

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

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

}

}

**Output:**

