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

**Step 1:**

****ORM (Object-Relational Mapping)** is a technique that lets you interact with a relational database using object-oriented code instead of SQL queries directly.**

**-> In **C#**, an ORM maps your C# **classes** to database **tables**, class **properties** to table **columns**, and class **instances** to **rows**.**

**-> For example, a Student class maps to a Students table, with Id, Name, Email properties mapping to columns.**

**->This allows developers to work with **objects**, and the ORM handles translating those actions into SQL under the hood.**

****Benefits:****

**-> **Productivity**: Write less boilerplate SQL.**

**-> **Maintainability**: Strongly typed code, refactoring is easier.**

**-> **Abstraction**: Decouples data access logic from raw SQL queries.**

**Step 2:**

**EF Core vs EF Framework (EF6):**

| **Feature** | **EF Core** | **EF Framework (EF6)** |
| --- | --- | --- |
| Platform | Cross-platform (.NET Core, .NET 5+) | Windows-only (.NET Framework) |
| Modern Features | LINQ, async queries, compiled queries, NoSQL support | Limited to relational DBs, older async support |
| Performance | Lightweight, faster, actively improved | More mature but heavier |
| Use case | New apps, cross-platform, cloud | Legacy apps on .NET Framework |

**Step 3:**

**JSON Column Mapping**: Map JSON columns in the database directly to C# types.

**Compiled Models**: Improves startup and query performance by pre-compiling model metadata.

**Interceptors**: Add hooks into EF Core’s execution pipeline — log, audit, or modify commands.

**Better Bulk Operations**: Improved performance for large inserts/updates.

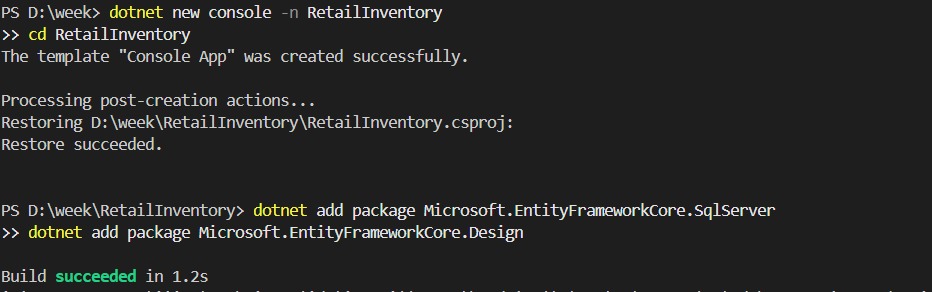
**Step 4 & 5:**

**Code:**

**dotnet new console -n RetailInventory cd RetailInventory**

**dotnet add package Microsoft.EntityFrameworkCore.SqlServer dotnet add package Microsoft.EntityFrameworkCore.Design**

**Output:**

****

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

**Step 1:**

**Category.cs:**

**namespace RetailInventory.Models;**

**public class Category**

**{**

**public int Id { get; set; }**

**public string Name { get; set; }**

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

**}**

**Product.cs:**

**using RetailInventory.Models;**

**namespace RetailInventory.Models;**

**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; }**

**}**

**Step 2:**

**AppDbContext.cs:**

**using Microsoft.EntityFrameworkCore;**

**using RetailInventory.Models;**

**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=DEMONLISHER\\MSSQLSERVER01;Database=RetailInventoryDb;Trusted\_Connection=True;TrustServerCertificate=True"**

**);**

**}**

**}**

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

**Step 1 (Install EF CLI):**

**Code:**

**dotnet tool install --global dotnet-ef**

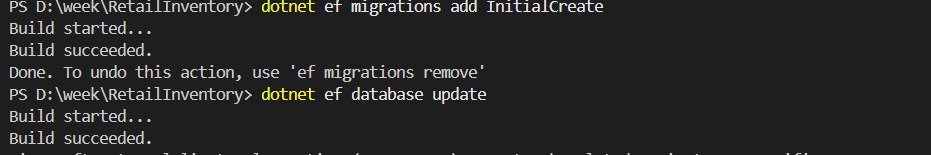
**Step 2 (Add a migration**)**:**

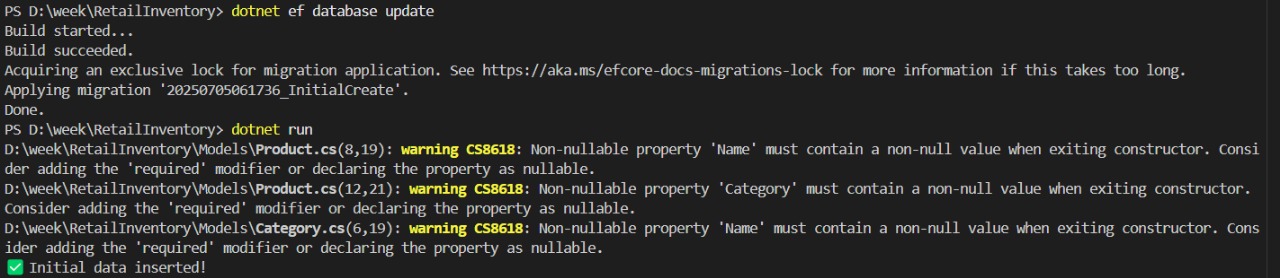
**dotnet ef migrations add InitialCreate**

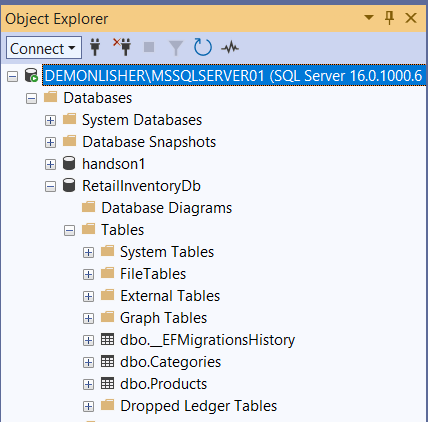
**Step 3 (Apply the migration):**

**dotnet ef database update**

**OUTPUTS OF STEP 3 AND 4:**

****

****



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

**Step 1 (Insert Data in Program.cs)**:**:**

**using RetailInventory.Models;**

**using var context = new AppDbContext();**

**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 };**

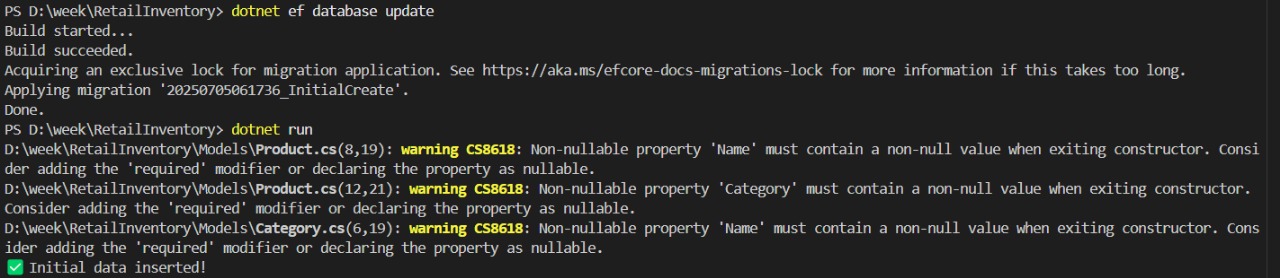
**var product2 = new Product { Name = "Rice Bag", Price = 1200, Category = groceries };**

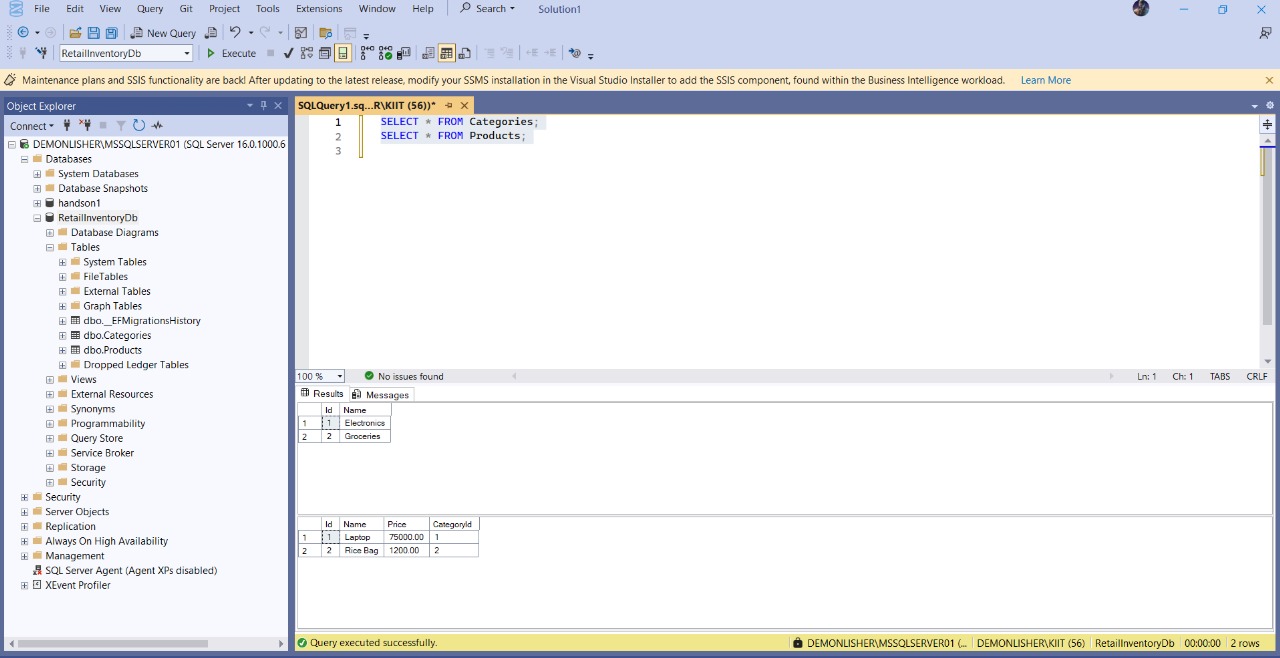
**await context.Products.AddRangeAsync(product1, product2);**

**await context.SaveChangesAsync();**

**Console.WriteLine(" Initial data inserted!");**

**OUTPUTS:**

****

****

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

**Step 1 (Retrieve All Products):**

**using Microsoft.EntityFrameworkCore;**

**using RetailInventory.Models;**

**using var context = new AppDbContext();**

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

**Console.WriteLine(" All Products:");**

**foreach (var p in products)**

**{**

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

**}**

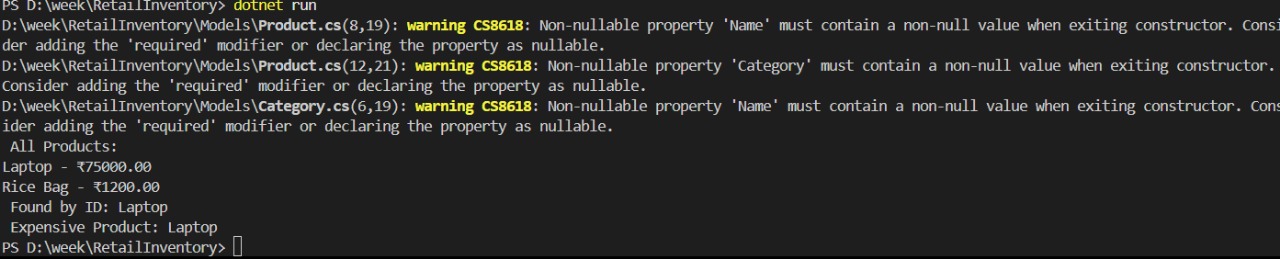
**var product = await context.Products.FindAsync(1);**

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

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

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

**OUTPUTS OF ALL THREE STEPS:**

****