**Week3-HANDSON**

**Module 5-** **EF Core 8.0**

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

**1) What is ORM?**

* Object-Relational Mapping (ORM) is a programming technique that allows developers to interact with database records as native programming language objects.
* Mapping ORM maps C# classes, such as Product or Category, to database tables like Products or Categories. Class properties (e.g., Product.Name) correspond directly to table columns (Products.Name).
* Benefits-
  + Productivity: Reduces boilerplate code by generating SQL automatically for CRUD operations.
  + Maintainability: Makes it easier to keep the data model and database schema in sync through features like migrations.
  + Abstraction: Allows developers to work with C# objects rather than raw SQL, focusing on the business domain instead of database syntax.

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

| **Feature** | **EF Core** | **EF 6.x** |
| --- | --- | --- |
| Cross-Platform | Yes (Windows, macOS, Linux) | No (Windows only) |
| Performance | Optimized for performance | Less optimized |
| Modularity | Lightweight and modular | Monolithic |
| LINQ Support | Improved LINQ support with new features | Traditional LINQ support |
| Tracking Changes | Efficient change tracking | More resource-intensive tracking |
| Migrations | Fluent migrations with improved capabilities | Traditional migrations |
| Batching | Supports command batching | Does not support command batching |
| Eager/Explicit/Lazy Loading | Supports all loading types, with better control | Supports all, but with performance overhead |
| New Features | Regular updates with new features | Limited updates, maintenance mode |

EF Core is the recommended choice for new applications, providing cross-platform support, modern features, and active development. While EF6 remains stable and mature, it is limited to the Windows platform and lacks many recent advancements.

**3) EF Core 8.0 Key Features**

EF Core 8.0 introduces powerful features that enhance modern data access scenarios:

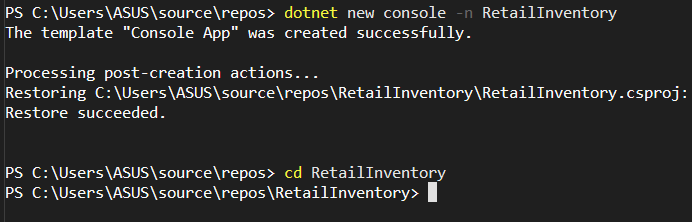
* JSON Column Mapping Enables storing and querying JSON data inside SQL Server columns, such as product attributes or specifications, without changing the table schema.
* Compiled Models Allows precompilation of entity mappings at build-time, significantly improving startup and runtime performance, particularly for applications with large schemas.
* Interceptors Provides hooks into EF Core operations like queries or commands, enabling developers to add logging, auditing, or custom business logic.
* Enhanced Bulk Operations Improves the efficiency of high-volume inserts and updates, which is essential for scenarios like adjusting stock levels or syncing inventory data.

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

Execute below given lines in terminal to create “RetailInventory”

**dotnet new console -n RetailInventory**

**cd RetailInventory**

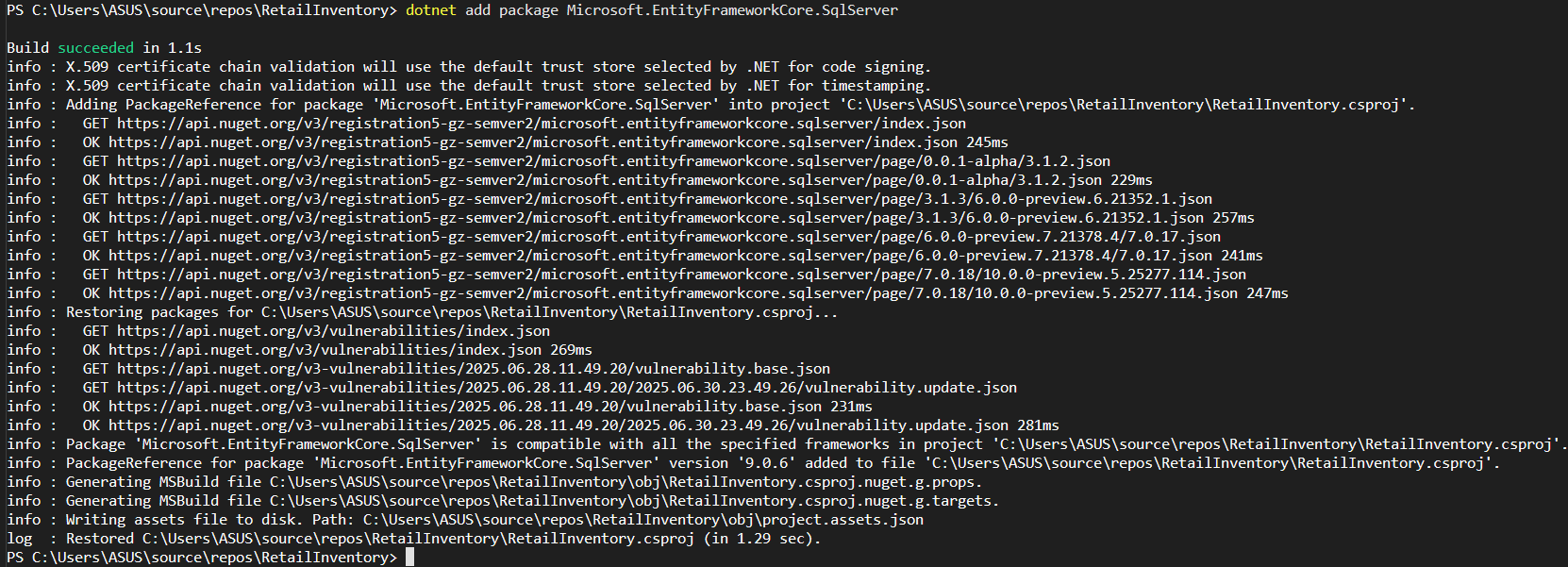


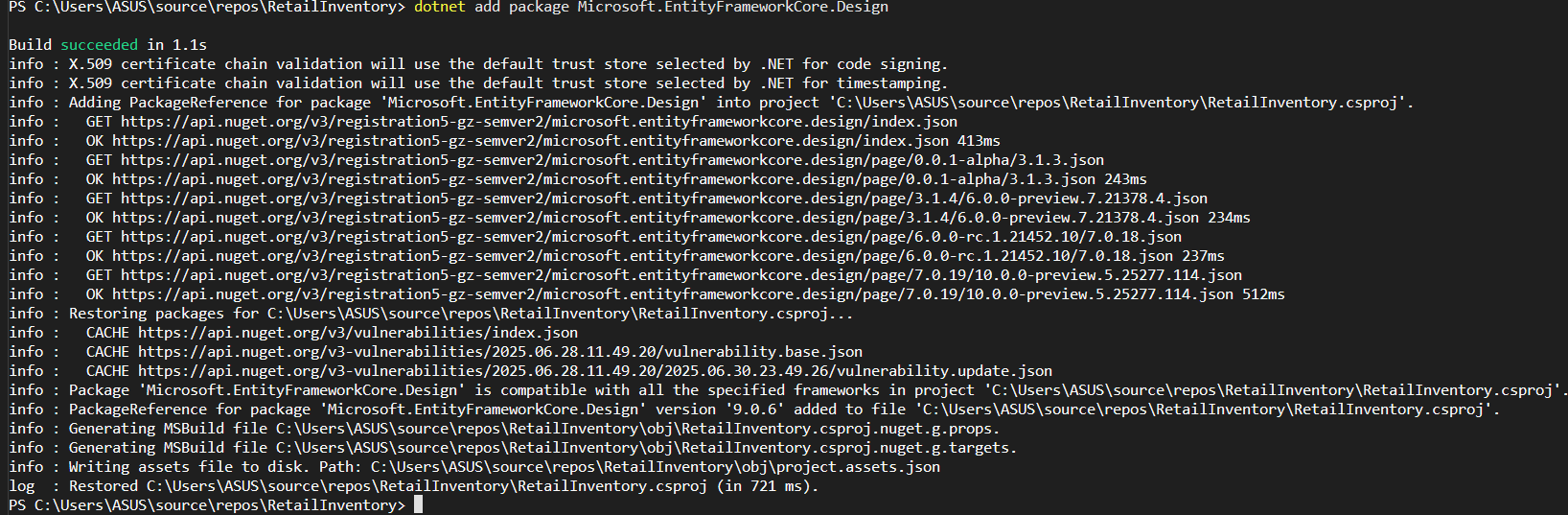
**5. Install EF Core Packages:**

Execute below given lines in terminal

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

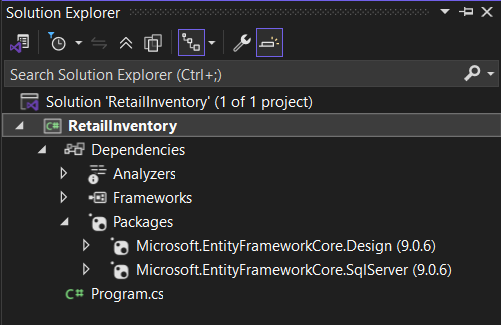
**dotnet add package Microsoft.EntityFrameworkCore.Design**

****

****

**Output:**

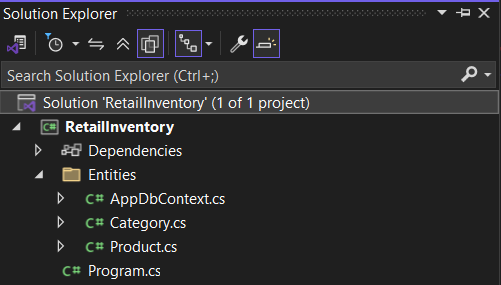
After successfully installing the packages, you can verify them from Solution Explorer under the Dependencies => Packages folder, as shown in the image below.

****

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

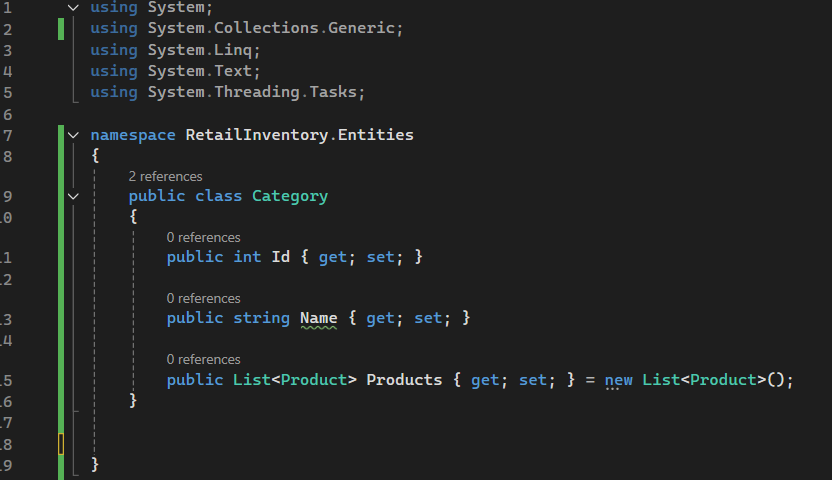
**Step1:**

Create Folder Entities inside Project and create Category.cs, Product.cs & AppDbContext.cs:

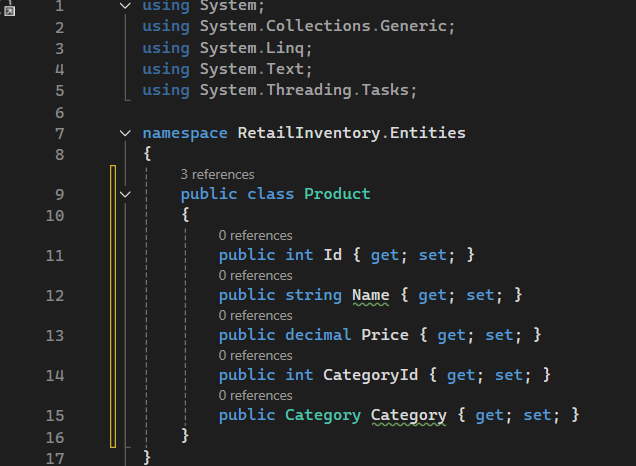
****

**Step2:**

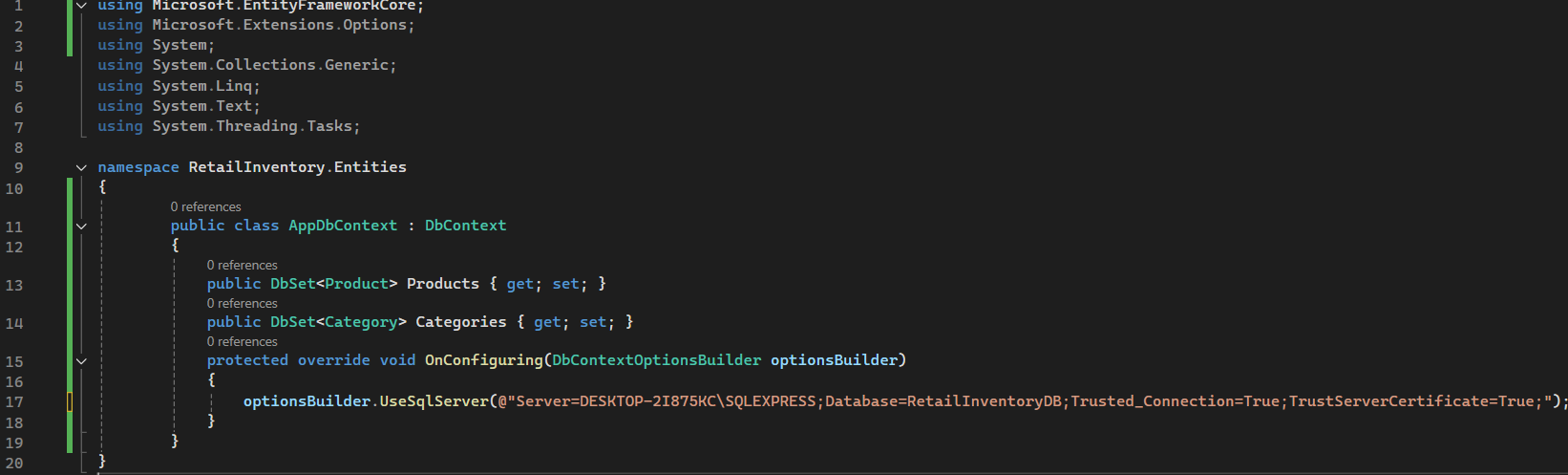
Type the below code inside Category.cs

****

Type the below code inside Product.cs

****

Type the below code inside AppDbContext.cs



**NOTE:**

"Server=DESKTOP2I875KC\SQLEXPRESS;Database=RetailInventoryDB;Trusted\_Connection=True;TrustServerCertificate=True;"

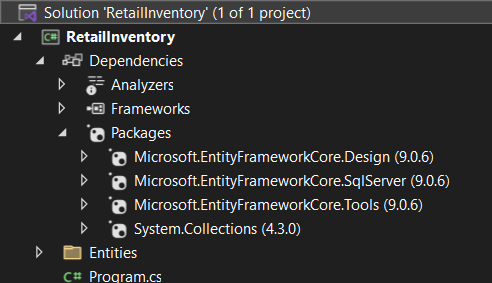
* Server= DESKTOP2I875KC\SQLEXPRESS: Specifies the database server’s name or network address.
* Database=RetailInventoryDB: Indicates the name of the database to connect to or create if it doesn’t exist yet.

**Step3:**

**Install** EF Core tools package

Tools-> NuGet Package Manager->Package Manager Console

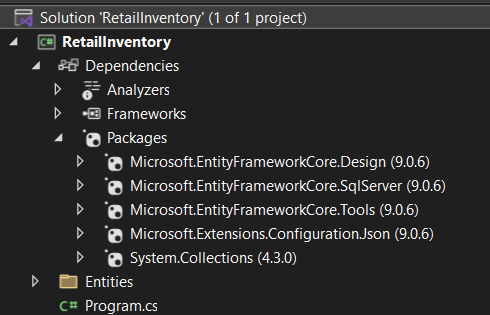
**Install-Package Microsoft.EntityFrameworkCore.Tools**

****

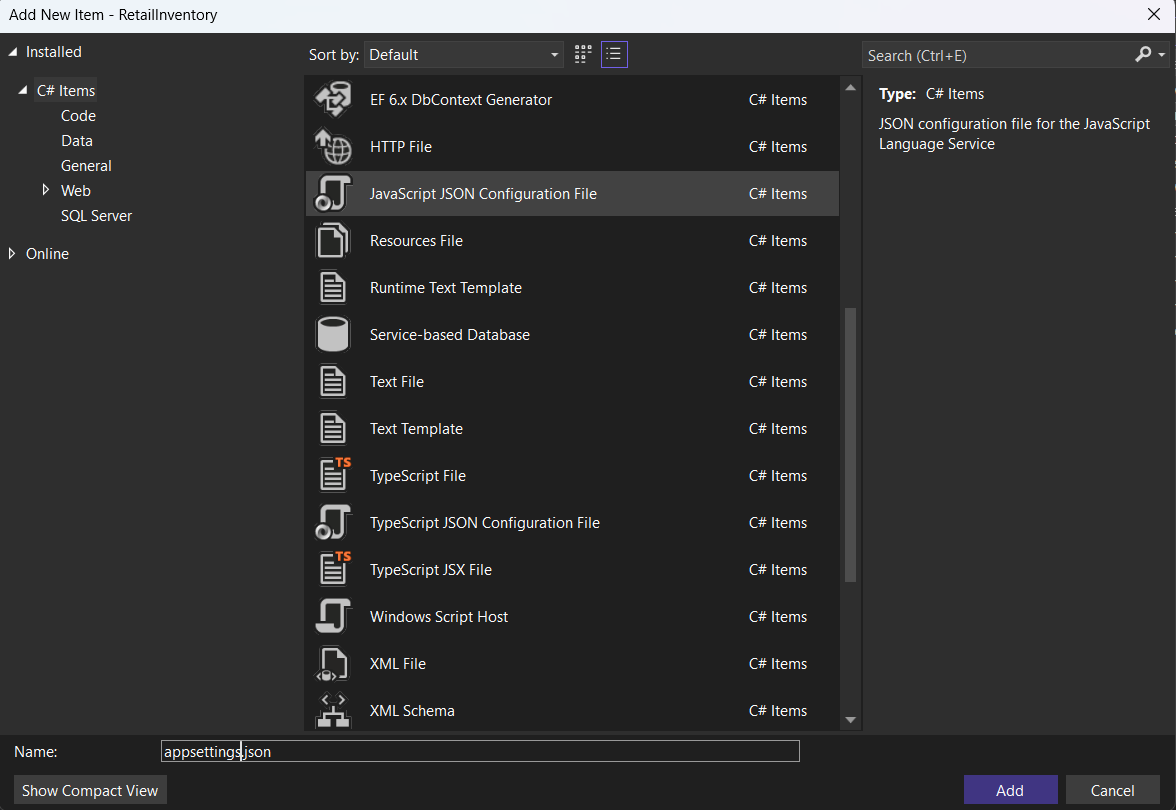
**Step4:**

Type below in Package Manager Console to Add Connection String in appsettings.json

**Install-package Microsoft.Extensions.Configuration.Json**

****

Add new Item to the project and then select JavaScript JSON Configuration File. Provide the file name as **appsettings.json**



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

**Step1:** **Install EF Core CLI (if not already):**

Open terminal and type

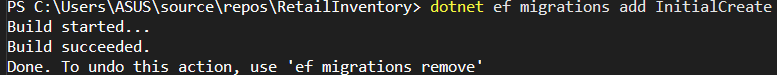
dotnet tool install --global dotnet-ef

****

**Step2:** **Create Initial Migration:**

type

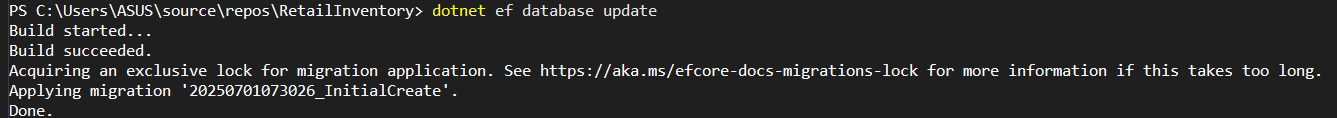
dotnet ef migrations add InitialCreate

****

**Step3:** **Apply Migration to Create Database:**

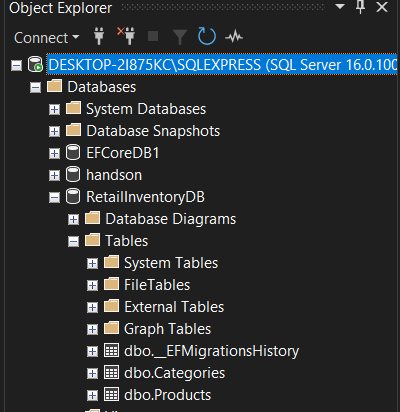
type

dotnet ef database update

****

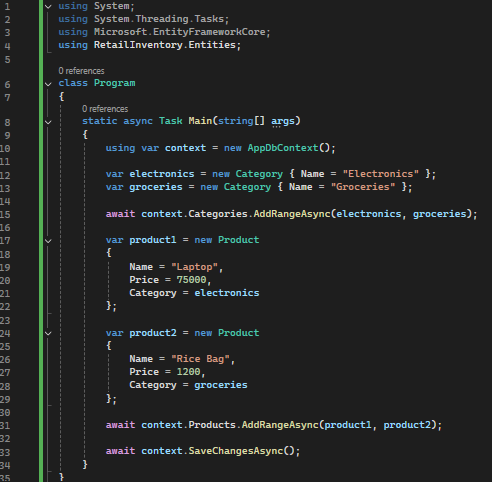
**Step4:** **Verify in SQL Server:**

Open SQL Server Management Studio (SSMS) or Azure Data Studio and confirm that tables Products and Categories are created.

****

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

**Step1: Type the below code inside Program.cs to insert data inside table**

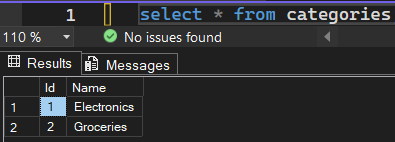
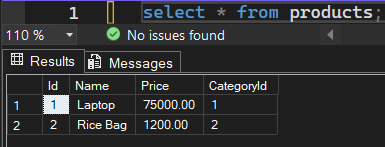
****

**Step2:** Run the App:

Open Terminal and type

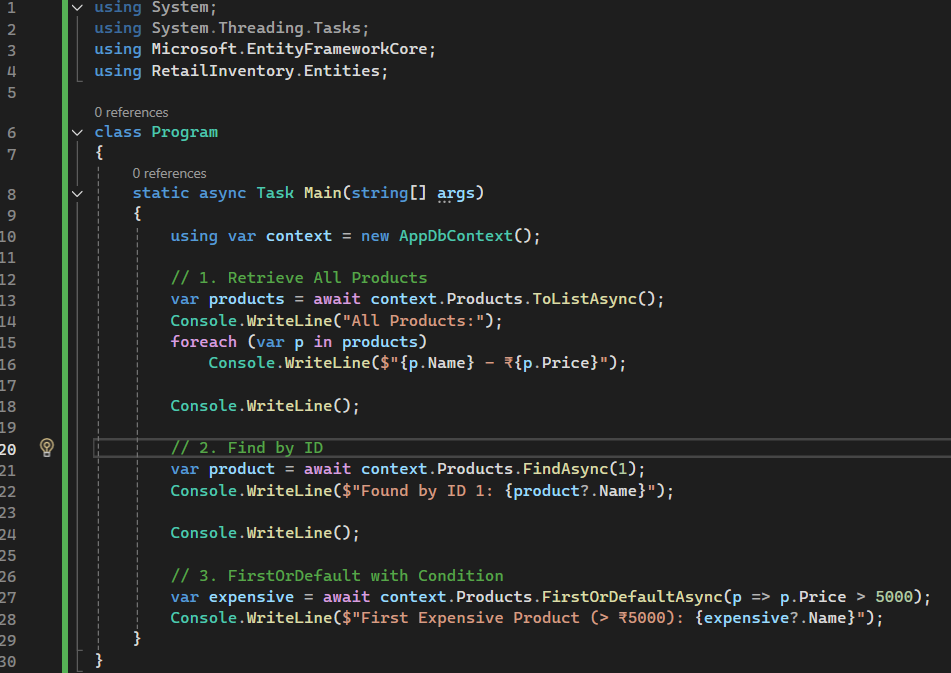
**dotnet run**

then check at SQL SERVER



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

Step1: write the below code inside Program.cs



Step2:run “dotnet” in terminal

