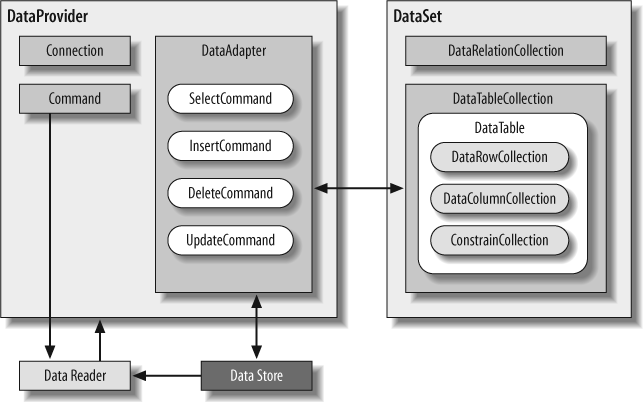
# Giới thiệu về ADO.NET

Trong phần này ta sẽ tìm hiểu về thư viện dùng để truy cập CSDL ADO.NET

ADO.NET là một tập hợp các thư viện cho phép ta tương tác với CSDL như SQL Server, My SQL Server,..

Kiến trúc cơ bản của ADO.NET gồm nhiều thành phần



Thu viện gồm nhiều lớp cấu trúc được chia ra làm 2 phần chính:

* **DataProvider** là các thư viện lớp cung cấp chức năng để kết nối đến những nguồn dữ liệu (VD: SQL Server,..) Cho phép thi hành các chức năng để thực hiện truy vấn CSDL từ câu lệnh INSERT, SELECT,…
* **DataSet** là một tập hợp các thư viện lớp cho phép tạo ra những đối tượng để quẻn lý dữ liệu và không phụ thuộc vào nguồn dữ liệu đó đến từ đâu. Nói cách khác DataSet là ánh xạ của CSDL về máy local
  + Trong DataSet tập hợp nhiều đối tượng DataTable, mỗi đối tượng DataTable tương ứng với một bảng trong CSDL mà chúng ta truy vấn

## DataProvider

Trong DataProvider gồm nhiều thành phần:

* **DbConnection**: là đối tượng cung cấp tài nguyên để kết nối đến CSDL
* **DbCommand**: là đối tượng sử dụng **DbConnection** để thực hiện các truy vấn
* **Data Reader**: là đối tượng dùng để đọc những kết quả truy vấn của **DbCommand**
* **DataAdapter:** gồm nhiều **DbCommand** (SelectCommand, InsertCommand,DeleteCommand, UpdateCommand). Đối tượng này giống như một cầu nối giữa nguồn dữ liệu (SQL Server) và **DataSet.** Khi có yêu cầu thì nó có thể lấy dữ liệu từ nguồn về **DataSet** và khi dữ liệu chỉnh sửa trong **DataSet** thông qua **DataAdapter** nó sẽ cập nhật trở lại dữ liệu nguồn (SQLServer)

Trong thư viện .NET có cung cấp DataProvider để truy cập vào SQL Server

Để cài đặt package ta sử dụng lệnh sau:

dotnet **add** package System.Data.SqlClient

# Tạo SQL Server chạy trên Docker

Dữ liệu mẫu:

<https://github.com/xuanthulabnet/learn-cs-netcore/releases/tag/exsqldocker>

Sau khi tải về và giải nén ta có thư mục

A screenshot of a computer

Description automatically generatedThư mục này chứa các config của docker

A screenshot of a computer program

Description automatically generatedNội dung file này bao gồm:

* Phiên bản đang chạy: 2017-latest
* Tên Container là: sqlserver-xtlab
* Khi SQLServer chạy nó thiết lập password cho tài khoản SA(Super Admin)
* Có tạo ra 1 volume để lưu trữ CSDL
* Cổng mặc định của SQL Server là cổng 1433

Truy cập thư mục MSSQL bằng terminal

A computer screen with white text

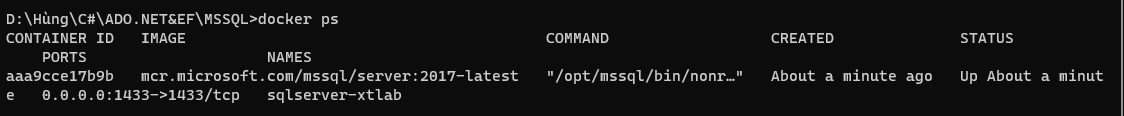
Description automatically generatedSử dụng lệnh dir để kiểm tra thành phần folder

Ta chạy lệnh để khởi tạo file

A screenshot of a computer program

Description automatically generated

Sau khi chạy lệnh ta kiểm tra bằng lệnh docker ps



Ta thấy ta đang có 1 container là sqlserver-xtlab đang chạy

Như vậy ta đang chạy SQLServer trên docker

Đối với SQL server này chúng ta có thể truy cập được từ địa chỉ localhost:1433 hoặc thông qua địa chỉ ip là 127.0.0.1 với tài khoản truy cập là “sa” password: “Password123”

Chúng ta có thể sử dụng công cụ Azure Data Studio để quản lý SQL server

A screenshot of a computer

Description automatically generatedĐây là ứng dụng Azure Data Studio, nó cho phép chúng ta kết nối đến SQL server và quản lý SQL server

Chúng ta có thể tạo ra kết nối đến SQL server đangc hạy trên Docker

**A screenshot of a computer

Description automatically generated**

Sau khi kết nối:

A screenshot of a computer

Description automatically generated

Phục hồi CSDL mẫu:

Mở terminal và chạy lệnh sau:

A black screen with white text

Description automatically generated

Chúng ta đang ở trong container chạy sql server

Trong file docker-compose khi chúng ta ánh xạ thư mục bk tương ứng là /var/opt/mssql/backup

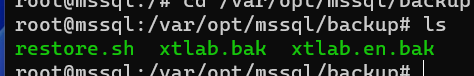
A close up of text

Description automatically generated

Tiếp tục đi đến thư mục

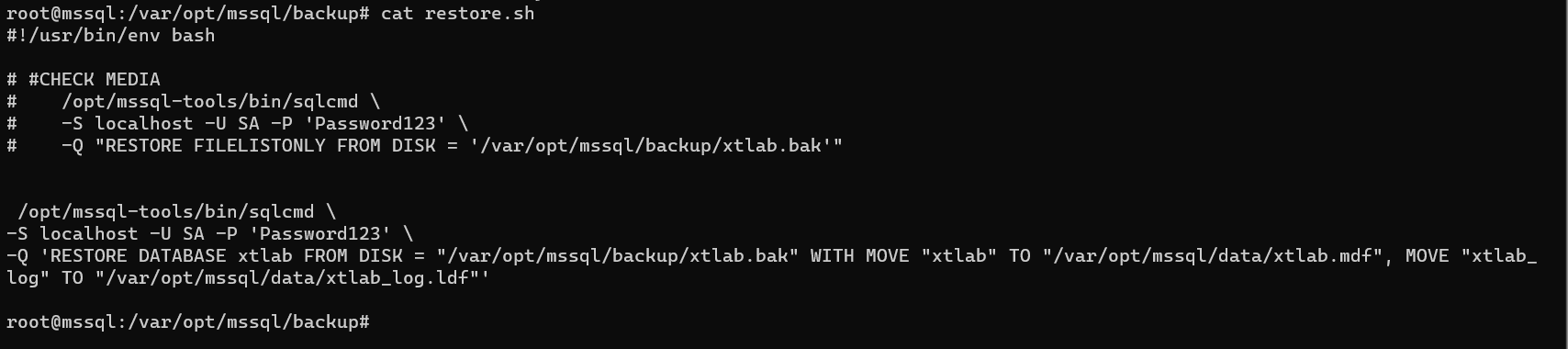
A black background with white text

Description automatically generated

Trong thư mục backup có file lưu trữ csdl mẫu là xtlab.bak

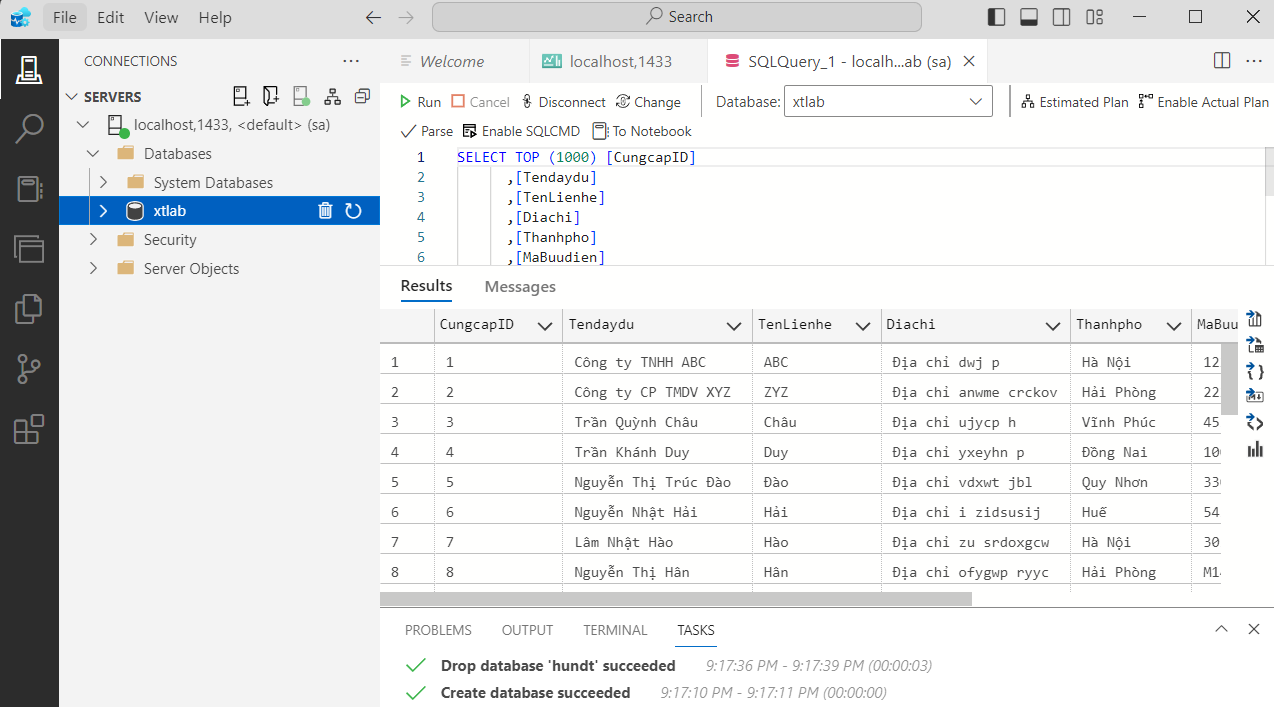
Chúng ta sẽ phục hồi CSDL này để trở thành csdl chạy trên docker

Nội dung file restore.sh



Sau khi hiển thị nội dung ta chạy file này ./restore.sh

Kết quả:

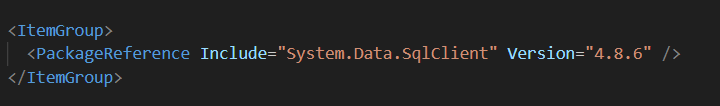


# DbConnection - Thực hành sử dụng SqlClient

* Trong phần này ta sẽ thực hành trên SqlClient để tạo ra 1 Connection kết nối đến CSDL trên SQL Server mà chúng ta dndag chạy trên docker
* Connection là đối tượng tài nguyên cơ bản đầu tiên mà ta cần tạo ra nếu muốn thực hiện kết nối đến SQL

Do chúng ta sử dụng SqlClient nên trong dự án cần add package **System.Data.SqlClient**





## Tạo chuỗi kết nối với SQL Server

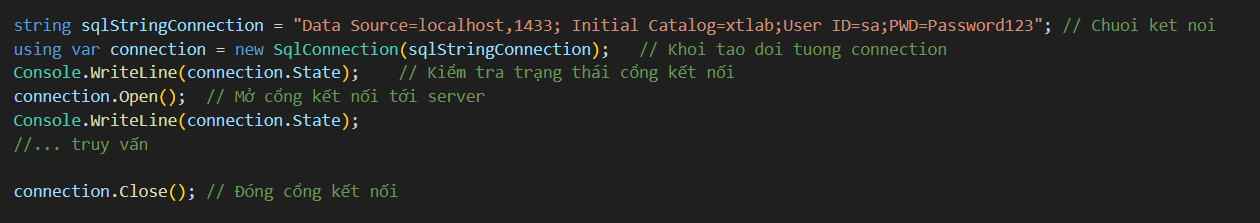
* Chuỗi kết nối có các thành phần, thông số để kết nối đến server
* Thông tin của chuỗi kết nối này được viết là các cặp key=value;

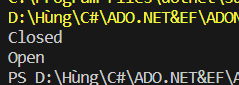




Ta sẽ sử dụng chuỗi kết nối này để làm tham số cho SqlConnection

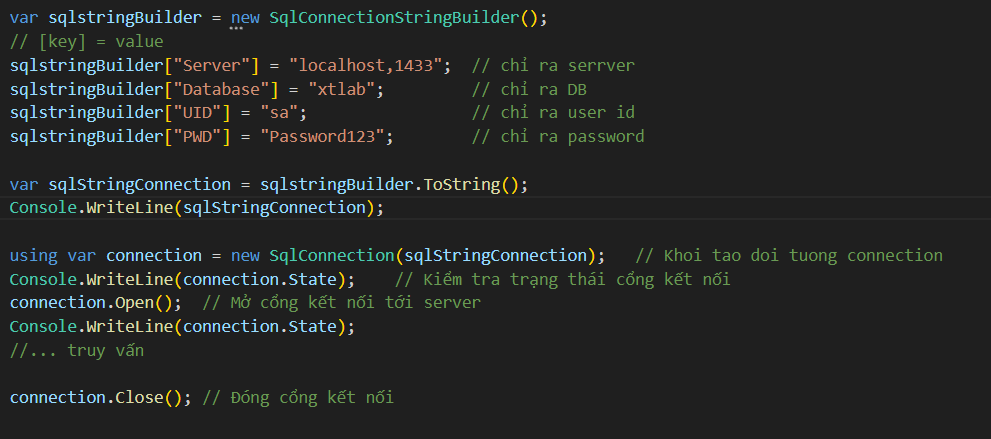
Sau khi có connection để mở kết nối đến server ta thực hiện

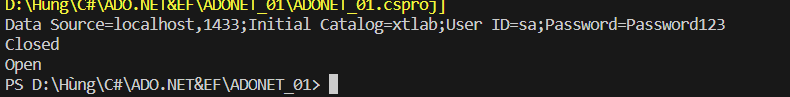




Khi cổng kết nối được mở ta có thể thực hiện các truy vấn sau khi thực hiện các truy vấn ta cần đóng lại chuỗi kết nối để giải phóng tài nguyên

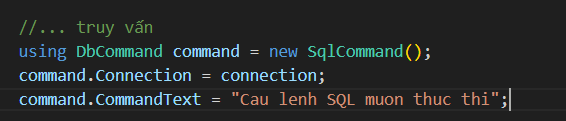
Trong .NET ta sẽ dùng đối tượng **SqlConnectionStringBuilder** để tạo ra chuỗi kết nối



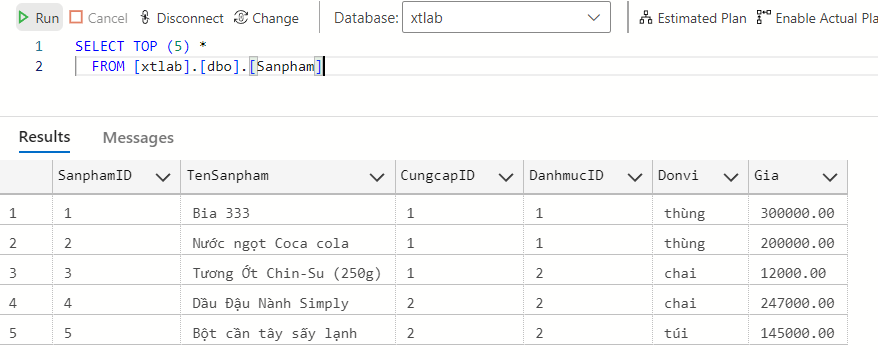


# DbCommand

* Đối tượng sử dụng để thực hiện câu truy vấn SQL đến CSDL



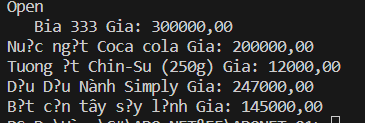
Giả sử trong SQL ta có câu lệnh để lấy ra 5 sản phẩm đầu tiên của bảng sản phẩm



Trong code ta thực hiện như sau:



Kết quả:



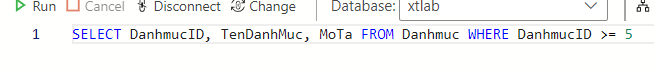
## SqlCommand

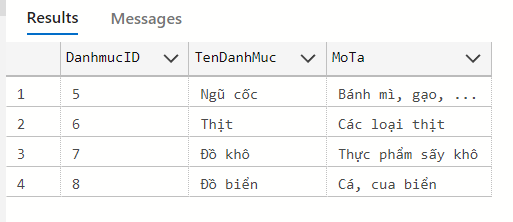
Như bài trước sau khi mở cổng kết nối 

Có 3 cách truy vấn như trong hình và mỗi cách có một công dụng khác nhau

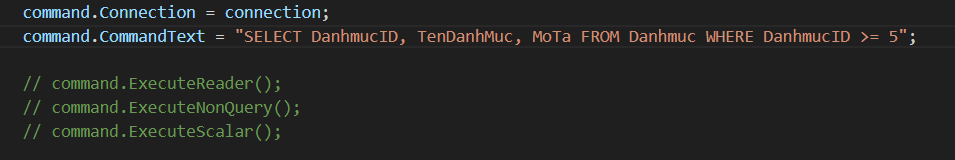
### Truyền tham số vào câu truy vấn SQL

Giả sử ta truy vấn như sau:





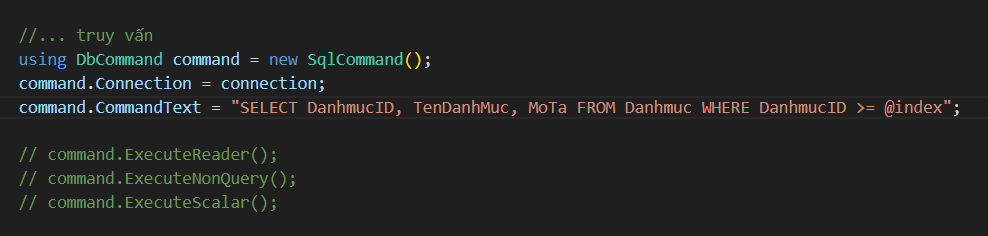
Như phần trước ta hoàn toàn có thể sử dụng câu truy vấn này trong code qua đối tượng SqlCommand



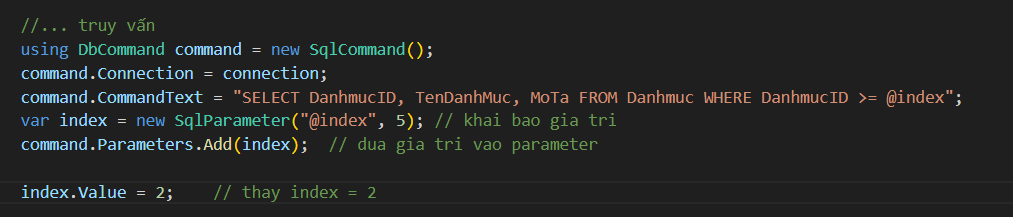
Sau đó khi chúng ta truy vấn với 1 trong 3 câu lệnh bên dưới câu truy vấn SQL sẽ được thực thi

Nhưng lúc này ta muốn DanhmucID có thể thay đổi mà không phải giá trị cứng

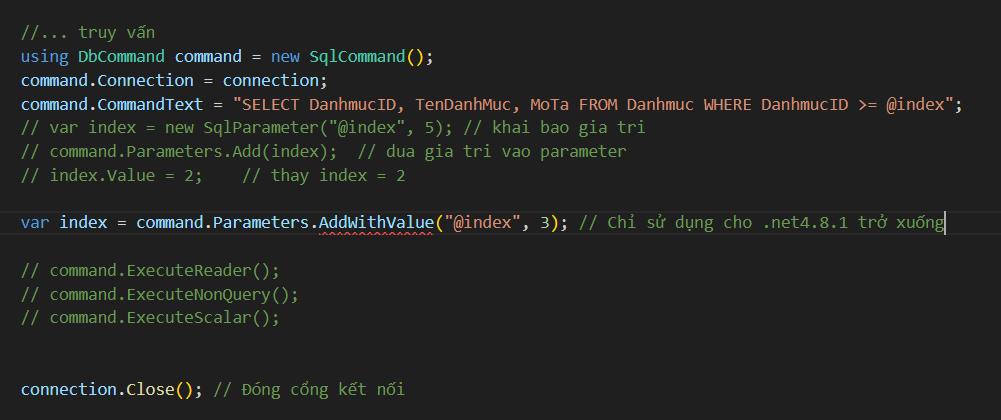
Để làm được điều này ta sẽ truyền vào trong câu lệnh SQL 1 tham số như sau:



Lúc này khi truy vấn SqlCommand sẽ tìm các tham số trong thuộc tính parameter

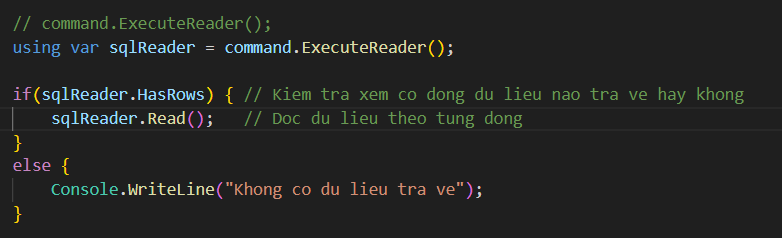


Ngoài cách này ta có thể làm như sau:



### ExecuteReader()

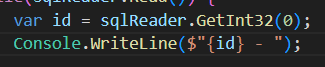
* Phương thức này trả về đối tượng SqlDataReader, và từ đối tượng này ta có thể đọc được các đối tượng truy vấn được

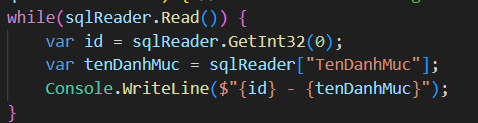


Phương thức Read() nếu gọi lần đầu tiên nó sẽ đọc dữ liệu dòng đầu tiên, tiếp đó con trỏ sẽ nhảy sang dòng tiếp theo, nếu thực hiện Read() lần thứ 2 thì nó sẽ đọc lần thứ 2,… cứ như vậy cho đến hết

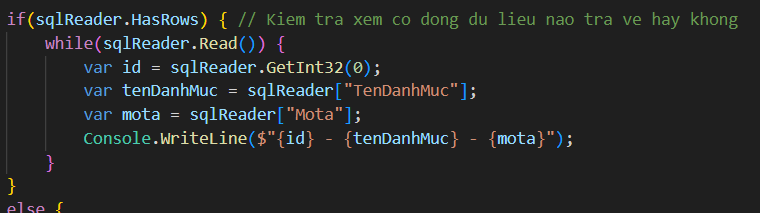
* Phương thức Read() nếu trả về True nghĩa là có dòng dữ liệu
* Phương thức Read() nếu trả về False nghĩa là đến cuối của dữ liệu không còn dữ liệu cần đọc

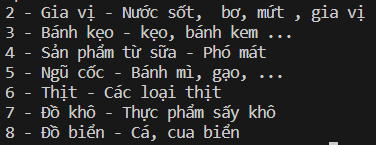
Vì vậy để đọc hết toàn bộ dữ liệu ta sẽ làm như sau:

Cách này đọc dữ liệu thông qua chỉ mục

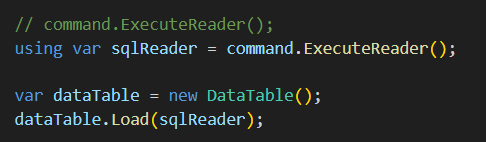


Cách này đọc dữ liệu thông qua Tên của trường dữ liệu



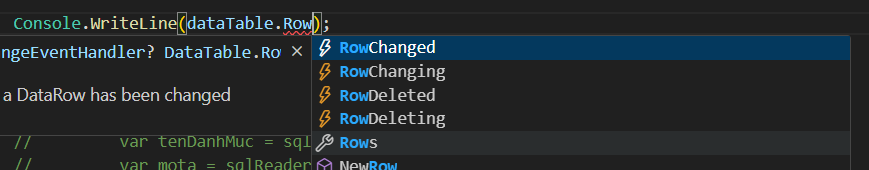


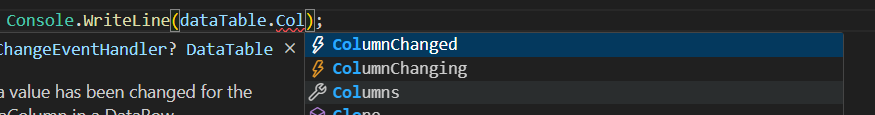
Sau khi có sqlReader thì toàn bộ dữ liệu truy vấn sẽ đổ ra 1 đối tượng là DataTable



Sau khi viết thế này thì toàn bộ dữ liệu truy vấn sẽ được lưu trong đối tượng dataTable

Trong đối tượng này gồm có nhiều hàng và cột





Mỗi hàng gồm các trường dữ liệu thì ta có thể duyệt qua cái hàng này để xử lý và sử dụng dữ liệu

Kết luận:

Chúng ta sử dụng ExecuteReader() khi câu truy vấn trả về một tập kết quả có nhiều dòng

### ExecuteScalar()

* Phương thức này sẽ trả về 1 giá trị là giá trị ở dòng 1 cột 1 kể cả câu truy vấn có trả về một tập hợp các dòng dữ liệu

Giả sử ta truy vấn được kết quả như sau:

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

Kết quả:

A screenshot of a computer

Description automatically generatedA black screen with white text

Description automatically generated trùng với hàng 1 cột 1

Vậy ứng dụng của phương thức này là sử dụng trong trường hợp kết quả truy vấn chỉ có 1 giá trị

VD: ta có câu truy vấn SQL sau:

A screenshot of a computer screen

Description automatically generatedCâu truy vấn này cho ta kết quả là đếm số lượng của user

Vì vậy ta có thể áp dụng ExecuteScalar()

### ExecuteNonQuery()

* Khi thi hành truy vấn nó sẽ không lấy tập kết quả mà truy vấn được từ server mà nó trả về tổng số dòng bị tác động bởi câu truy vấn đó
* Thông thường sử dụng phương thức này trong trường hợp thực hiện các câu lệnh như INSERT, UPDATE, DELETE

Vd: Ta sẽ thực hiện câu lệnh SQL để chèn thêm dữ liệu

* Chèn thêm 1 dòng vào bảng user
* A screenshot of a computer

  Description automatically generated

A computer screen with text and numbers

Description automatically generated

A screenshot of a computer

Description automatically generated

## Sử dụng SqlCommand để gọi StoreProcedure

* Tức là gọi thủ tục

Ví dụ:

A screenshot of a computer

Description automatically generatedTa dùng SQL để lấy ra name của id = 4

A screenshot of a computer

Description automatically generatedCú pháp SQL bên sẽ gộp 2 table để thành 1 table mới

Toàn bộ truy vấn trên ta sẽ chuyển thành 1 StoreProcedure

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedSau khi viết lại thành procedure ta run sẽ tạo 1 proceduce trong Programmability > Stored Procedures

Sau khi có procedure ta có thể sử dụng nhưu sau:

A screenshot of a computer

Description automatically generated

Giờ ta muốn trong code C# cũng có thể gọi được procedure này ta làm như sau:

A screen shot of a computer program

Description automatically generated

A black screen with white text

Description automatically generated

# DataSet, DataAdapter

A diagram of a data store

Description automatically generatedDataSet

* DataSet là một cấu trúc dữ liệu phức tạp
* Bên trong nó bao gồm các DataTable, đối tượng DataTable tương tự như những table của CSDL

DataAdapter

* Trong DataAdapter gồm nhiều những DbCommand cụ thể là các:
  + SelectCommand
  + InsertCommand
  + DeleteCommand
  + UpdateCommand

Thông qua DataAdapter chúng ta có thể ánh xạ DataSet và CSDL SQL (Data Store)

Ta có thể yêu cầu DataAdapter lấy dữ liệu từ SQL Server(Data Store) đổ vào các table trong DataSet, sau khi có những dữ liệu này rồi ra có thể chỉnh sửa, cập nhật, thêm mới các dữ liệu vào DataTable và thông qua DataAdapter nó sẽ cập nhật lại vào CSDL (Data Store)

## DataSet

DataSet là một cấu trúc dữ liệu nó biểu diễn, ánh xạ CSDL

Khởi tạo DataSet



Trong dataSet có thuộc tính là Table, thuộc tính này chứa tập hợp các DataTable, mỗi DataTable là ánh xạ của một Table trong CSDL

A screenshot of a computer

Description automatically generated

Ta cũng có thể gán tên khi khởi tạo table

A black background with white text

Description automatically generated

Cấu trúc của DataTable tương tự như bảng của CSDL, gồm nhiều trường dữ liệu gọi là các cột, nhiều dòng dữ liệu (record)

Chúng ta có thể thêm các cột và hàng như sau:

A screen shot of a computer program

Description automatically generated

Chúng ta cũng có thể lấy ra tên table và nội dung các cột các hàng như sau:

A screen shot of a computer code

Description automatically generated A screenshot of a computer

Description automatically generated

Ta có thể tạo sẵn 1 hàm để hiển thị cột:

A computer screen shot of a program

Description automatically generated A computer screen shot of text

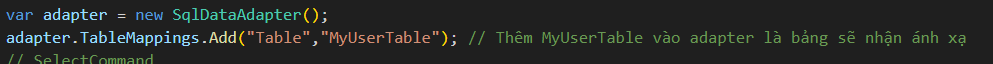
Description automatically generated

A screen shot of a computer

Description automatically generated

## DataAdapter

* Đây là đối tượng làm cầu nối án xạ giữa nguồn dữ liệu thật và DataSet



### SelectCommand

Muốn DataAdapter có thể lấy dữ liệu về ta cần thiết lập thuộc tính SelectCommand, thuộc tính này được gán bằng 1 SqlCommand nó được thực thi khi ta lấy dữ liệu

A screen shot of a computer code

Description automatically generated

Khi ta gọi phương thức Fill của adapter SelectCommand sẽ được thực thi

Lúc này trong DataSet sẽ có table

A black screen with white text

Description automatically generated

A screen shot of a computer code

Description automatically generated

Lúc này chúng ta đã ánh xạ được bảng User trong CSDL vào bảng MyUserTable trong DataSet

### InsertCommand

Lúc này nếu chúng ta thao tác với dữ liệu DataTable trong DataSet chúng ta có thể yêu cầu DataAdapter cập nhật lại CSDL

A computer screen with text

Description automatically generated A screenshot of a computer

Description automatically generated

Ta thấy kết quả không có gì thay đổi vì để chèn dữ liệu mới thì trong dataAdapter phải thiết lập thuộc tính InsertCommand, phương thức này được thực thi khi ta chèn dữ liệu mới vào dữ liệu

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Lưu ý khi chèn phải đảm bảo đầy đủ các trường dữ liệu not null

var adapter = new SqlDataAdapter();

            adapter.TableMappings.Add("Table","MyUserTable"); // Thêm MyUserTable vào adapter là bảng sẽ nhận ánh xạ

            // SelectCommand

            string sqlConnectString = "select \* from dbo.[user]";

            adapter.SelectCommand = new SqlCommand(sqlConnectString, connection);

            // InsertCommand

            adapter.InsertCommand = new SqlCommand(

                "insert into dbo.[user] (id, name, gender) values (@id, @name, @gender)",

                 connection);

            adapter.InsertCommand.Parameters.Add("@id", SqlDbType.Int, 32, "id");

            adapter.InsertCommand.Parameters.Add("@name", SqlDbType.NVarChar, 50, "name");

            adapter.InsertCommand.Parameters.Add("@gender", SqlDbType.VarChar, 6, "gender");

            var dataSet = new DataSet();

            adapter.Fill(dataSet);  // Đổ dữ liệu vào dataSet

            // Lấy ra bảng

            DataTable table = dataSet.Tables["MyUserTable"];

            ShowTable(table);

            // Thao tác với DataTable

            var row = table.Rows.Add();

            row["id"] = "20";

            row["name"] = "Hoàng Thanh Tùng";

            row["gender"] = "male";

            // Cập nhật lại nguồn dữ liệu

            adapter.Update(dataSet);

### DeleteCommand

Giả sử ta muốn xóa dòng 15 là dong ta vừa thêm trong table:

A black background with white text and yellow letters

Description automatically generated

Tuy nhiên để làm được điều này ta cần thiết lập thuộc tính DeleteCommand

A screen shot of a computer

Description automatically generated

Ở đây ta cần xác định:

* Nguồn để lấy dữ liệu từ datatable là ở cột nào
* 
* Đối với thao tác xóa hay cập nhật ta cần phải thiết lập tham số đó có thuộc tính là SourceVersion
* 

Lúc này ta có thể xóa được dữ liệu khi thực thi:

A screenshot of a computer

Description automatically generated

### UpdateCommand

* Thuộc tính này sẽ được thực thi khi có thao tác khi chỉnh sửa các dòng dữ liệu
* Giả sử ta sẽ sửa sđt của dòng 1 thành 031201002658



A black background with white text and numbers

Description automatically generated

A screen shot of a computer code

Description automatically generated

Kết quả:

A screenshot of a computer

Description automatically generated

# EF (Entity Framework)

## Giới thiệu

* Là 1 thư viện được xây dựng như là một tầng phía trên cảu ADO.NET cho phép ánh xạ những đơn vị của CSDL ví dụ như các bảng, các CSDL, các mối quan hệ vào những đối tượng mà chúng ta mô tả trong code C#, những đối tượng này là những class, cấu trúc. Từ đó ta có thể sử dụng LINQ để truy vấn lấy các dữ liệu về, thao tác với dữ liệu

## Entity Framework Core

* Là mã nguồn mở
* Code của toàn bộ nền tảng này có trên github
* Nó có thể làm việc trên nhiều loại CSDL

Tích hợp vào dự án:

dotnet **add** package System.Data.SqlClient

dotnet **add** package Microsoft.EntityFrameworkCore

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

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

dotnet **add** package Microsoft.Extensions.DependencyInjection

dotnet **add** package Microsoft.Extensions.Logging

dotnet **add** package Microsoft.Extensions.Logging.Console

A screen shot of a computer program

Description automatically generated

Như đã nói EF tạo ra mối liên hệ ánh xạ giữa những thành phần của CSDL thật vào những đơn vị code (class – entity)

Sau khi ánh xạ, ta có thể sử dụng linq để truy vấn

Các thành phần của CSDL thật ta gọi là các Entity (Database, Table)

Ví du: Giả sử ta có 1 CSDL trên SQL Server ví dụ tên của CSDL này là data01, trong CSDL này tạm thời có 1 bảng product

### Bước 1: Tạo class biểu diễn CSDL

A screenshot of a computer

Description automatically generatedThông thường khi khai báo các CSDL ta sẽ khai báo trong thư mục có tên là: Model hoặc Database hoặc Entities

Trong EF đối tượng để biểu diễn 1 CSDL là những đối tượng kế thừa từ class **DbContext**

A screen shot of a computer

Description automatically generated

Với khai báo như vậy thì các đối tượng ProductDbContext biểu diễn một CSDL

Bất kì khi nào 1 DbContext được tạo mới thì nó sẽ thi hành 1 phương thức có tên là **OnConfiguring**

Chúng ta thường nạp chồng phương thức này để cấu hình kết nối tới CSDL

A screen shot of a computer program

Description automatically generated

Phương thức này chạy khi đối tượng DbContext mới được tạo ra

Tại đây chúng ta sẽ thực hiện rất nhiều cấu hình liên quan đến CSDL của chúng ta

optionsBuilder.UseSqlServer(); // Cho biết DbContext làm việc với CSDL Sql Server

Để làm việc với SQL Server ta cần có chuỗi kết nối tới SQL Server

A screen shot of a computer program

Description automatically generated

Tất nhiên thì đến thời điểm này trong Server vẫn chưa có CSDL data01

### Bước 2: Sử dụng DbContext Tạo CSDL

#### Tạo CSDL

Ta tạo 1 phương thức dùng để tạo Database:

A screen shot of a computer code

Description automatically generated

A screen shot of a computer

Description automatically generated 

Để tạo ra CSDL từ DbContext ta làm như sau:

dbContext.Database.EnsureCreated();

hoặc

dbContext.Database.EnsureCreatedAsync();

Nếu là phương thức bất đồng bộ

Phương thức này thi hành sẽ kiểm tra trên server nếu CSDL đó không có thì nó sẽ tạo ra CSDL đó, nếu trong CSDL đó không có các table do DbContext đó biểu diễn nó sẽ tạo ra những bảng đó

Kết quả chạy trả về true nếu thành công và false nếu thất bại

A screen shot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generatedA black screen with white text

Description automatically generated

#### Xóa CSDL

Ta cũng tạo 1 phương thức để xóa DB:

A screen shot of a computer program

Description automatically generated

A black background with white text

Description automatically generatedA screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Database data01 đã được xóa

Lưu ý nó chỉ xóa được khi database tồn tại

### Bước 3: Khai báo Model, Entity, DbSet

Ở bước trên ta thấy rằng tại thời điểm này nó chưa biểu diễn 1 table nào trong CSDL đó

Ta sẽ tạo 1 class Product để biểu diễn các phần tử, các dòng của Product

A screenshot of a computer program

Description automatically generated

Bây giờ chúng ta sẽ sử dụng class này trong DbContext

Trong DbContext có các thuộc tính public DbSet

DbSet là biểu diễn 1 bảng của CSDL, mỗi dòng của bảng của CSDL đó nó biểu diễn 1 đối tượng lớp nào đó

Ở đây chúng ta khai báo ra 1 DbSet tức là bảng biểu diễn các Product

A screen shot of a computer program

Description automatically generated

Lúc này CSDL trong DbContext nó biết là nó có 1 table mà mỗi dòng của table đó tương ứng với 1 phần tử kiểu product

Lúc này ta sẽ sử dụng các thuộc tính(Attribute) mà các thuộc tính này miêu tả dữ liệu tương ứng với CSDL mà nó sẽ được sử dụng bởi EF

A screen shot of a computer program

Description automatically generatedThuộc tính Table cho biết tên của table

Khi có thuộc tính này thì Thuộc tính product trong DbContext tương ứng với table có tên là TableName

Có rất nhiều các attribute được sử dụng cho property của class

A screen shot of a computer program

Description automatically generatedAttribute **Key** : khai báo khóa chính

**Required :** khai báo trường này không được null

**StringLength(50) :** cho biết số kí tự tối đa cho phép

Tại thơi điểm này trên server vẫn chưa có Database vì ta chưa chạy phương thức CreateDatabase()

Lúc này ta thực thi phương thức CreateDatabase() thì trong DbContext đã có 1 table là myproduct tương ứng với thuộc tính product trong DbContext

A black background with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

CSDL data1 được tạo và chứa table myproject gồm 3 cột tương ứng với các thuộc tính khai báo trong model

Để đổi tên DB ta cần xóa DB bằng phương thức DropDatabase và sau đó CreateDataBase lại

A screen shot of a computer program

Description automatically generated

 A black screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

### Bước 4: Thao tác với dữ liệu bằng EF

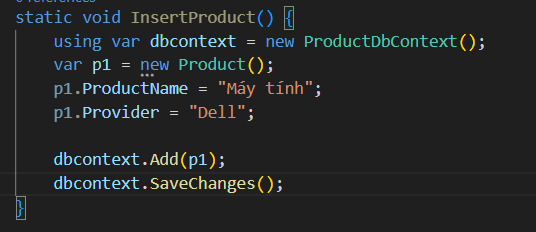
#### Insert – chèn dữ liệu

Chèn mới những dòng dữ liệu vào database

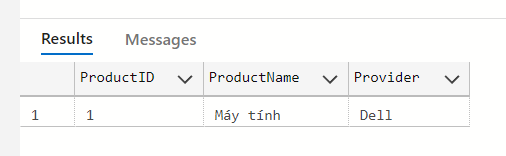
Ta sẽ tạo 1 phương thức InsertProduct để thêm dòng dữ liệu vào table

Để chèn dữ liệu vào table trong phương thức ta cần những điều kiến sau:

* Model: ở đây là Product
* Gọi phương thức Add hoặc AddAsync -> DBContext xuất hiện dữ liệu thêm
* SaveChange hoặc SaveChangeAsync để ra lệnh cho dbcontext cập nhật vào SQL server

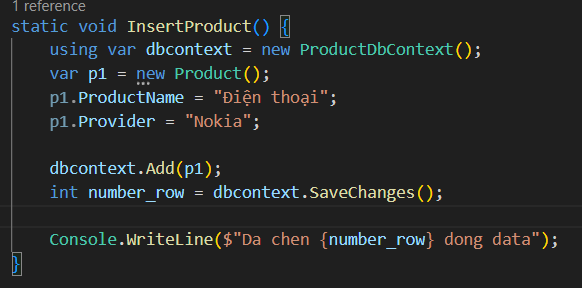
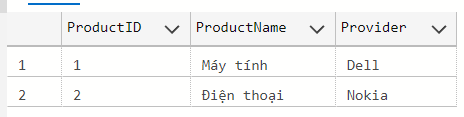


Sau khi gọi phương thức InsertProduct:



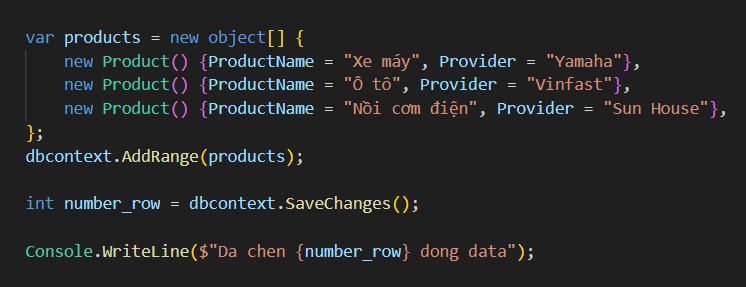
Lưu ý phương thức SaveChange() phải gọi khi Insert, Update, Delete data

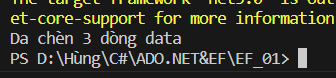
Phương thức này trả về số dòng bị tác động

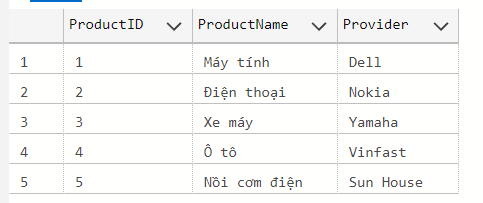




Để chèn nhiều dòng dữ liệu 1 lúc ta có thể sử dung phương thức AddRange

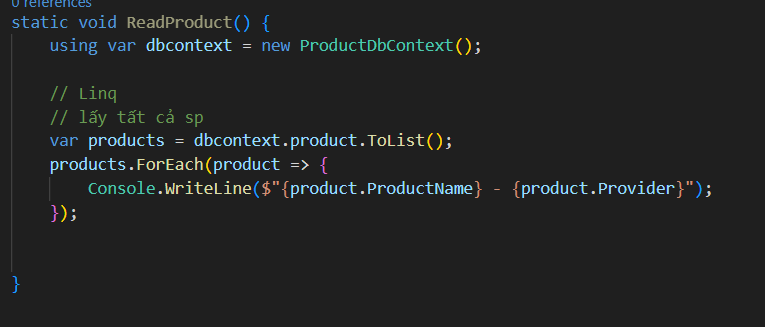


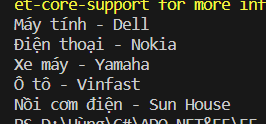




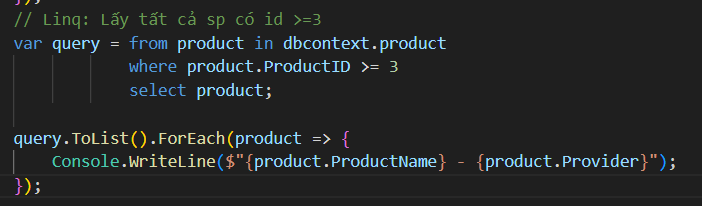
#### Read - Đọc dữ liệu

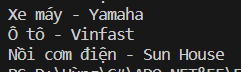
Đối với EF nó sử dụng Linq để truy vấn lấy dữu liệu, nguồn truy vấn là các thuộc tính biểu diễn Table trong DbContext





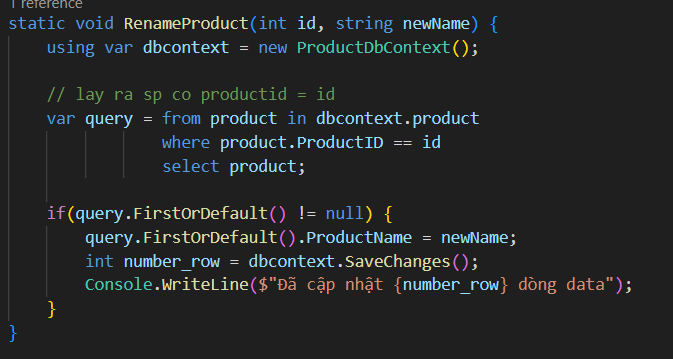
Sử dụng LINQ

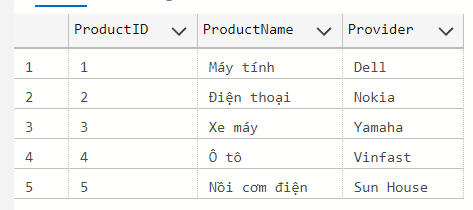




#### Update – Cập nhật dữ liệu

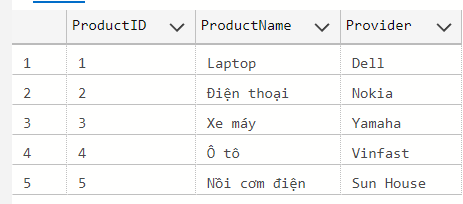
Ta tạo 1 phương thức dùng để đổi tên sp



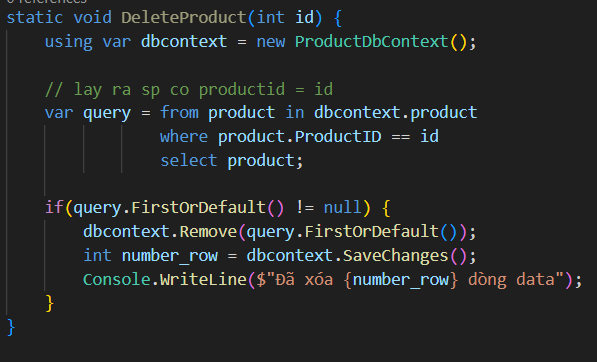


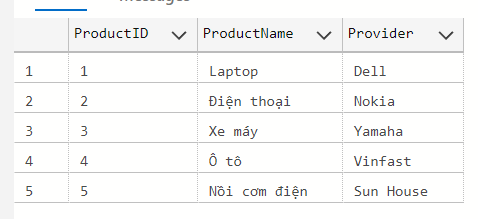
Ta sẽ đổi máy tính thành laptop



#### Delete – Xóa dữ liệu





Thử xóa dòng 3



## Logging

Khi ta thực hiện truy vấn cập nhật CSDL dựa trên dbcontext, EF thì chúng ta không phải làm việc trực tiếp với các câu lệnh SQL mà thư viện logging tự động phát sinh ra câu lệnh SQL tương ứng và nó thi hành

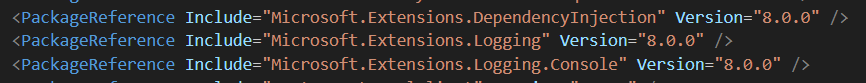
Trong trường hợp ta muốn giám sát nó thực sự đnag hoạt động như thế nào, nó tạo ra những câu truy vấn SQL ra làm sao thì ta có thể sử dụng những kỹ thuật để ghi lại thông tin đó

Để làm điều này trong dự cấn cần tích hợp đầy đủ 3 package sau:

dotnet **add** package Microsoft.Extensions.DependencyInjection

dotnet **add** package Microsoft.Extensions.Logging

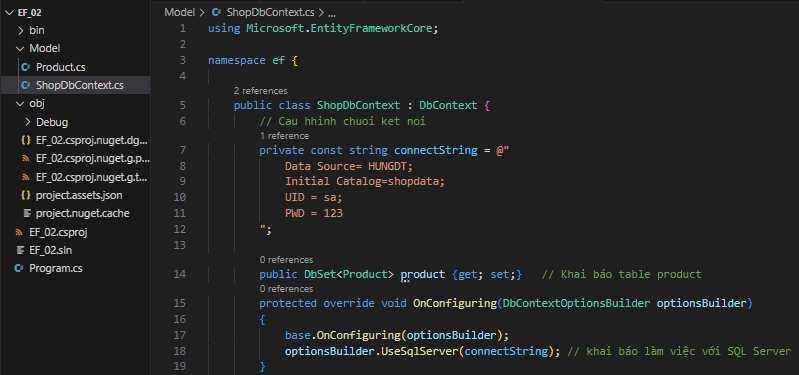
dotnet **add** package Microsoft.Extensions.Logging.Console



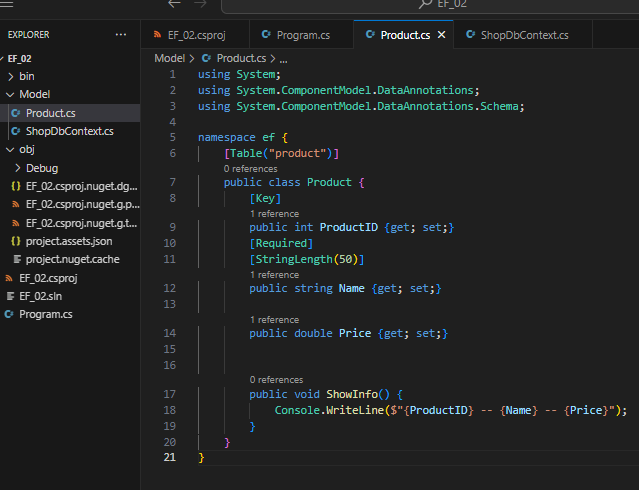
# Tạo Model, thiết lập các mối quan hệ

* Trong phần này ta sẽ tìm hiểu kỹ hơn trong việc tạo ra các model, tạo ra các dbContext có nhiều bảng

Ta sẽ tạo csdl mới có tên là shopdata

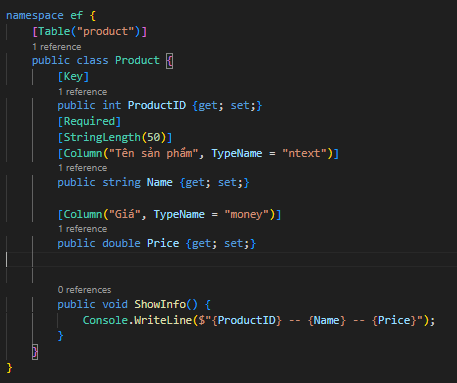
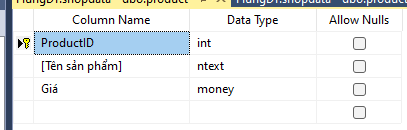


Class Product



## Tìm hiểu thêm về Attribute

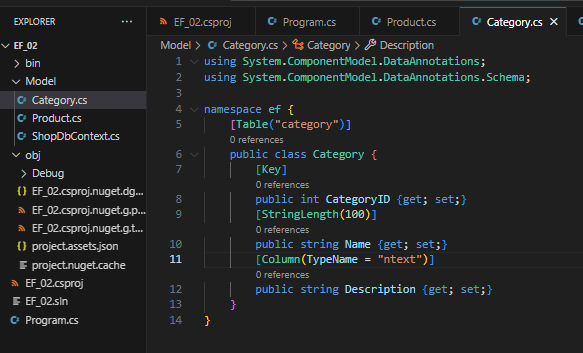
|  |  |
| --- | --- |
| Table(“tên table”) | Chỉ ra model nào đó tương ứng với 1 table trên SQL server |
| Key | Thiết lập khóa chính được áp dụng cho Property để ấn định property của class đó là PK |
| Require | Yêu cầu trường dữ liệu khác null |
| StringLength(độ dài) | Thiết lập độ dài của trường dữ liệu  String -> nvarchar |
| Column(TypeName = “Kiểu dữ liệu trên SQL server”) | Chỉ định kiểu dữ liệu trên SQL server |
| Column(“Tên cột”, TypeName = “Kiểu dữ liệu trên SQL server” ) | Mặc định khi tạo trường dữ liệu trên CSDL sẽ trùng tên với property trong class được ánh xạ  Sử dụng thuộc tính này để đổi tên Cột trên CSDL |

## Xây dựng thêm model (Mối quan hệ 1 – nhiều)

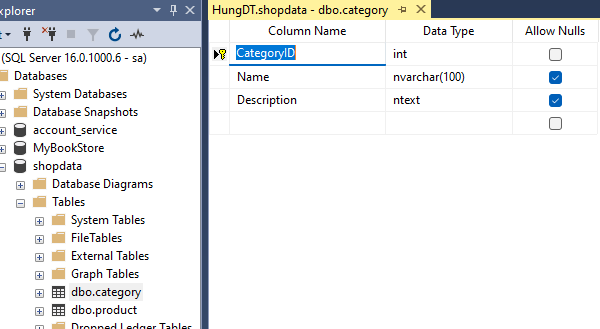
Ta sẽ xây dựng thêm 1 model nữa tên là Category

Category là 1 bảng biểu diễn danh mục sp, mỗi 1 sp sẽ thuộc 1 danh mục nào đó



Trong DbContext khai báo table:



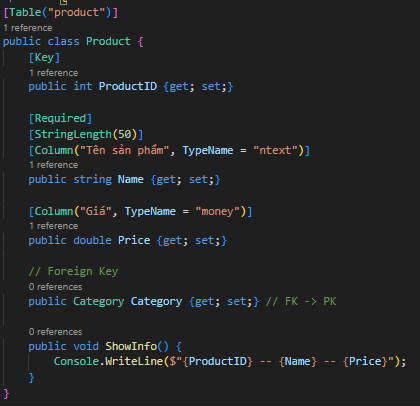


2 bảng này có sự liên hệ, mỗi sp thuộc 1 category nào đó nhưng hiện tại chưa có mối quan hệ nào

### Tạo mỗi liên hệ giữa 2 bảng

Ta xác định bảng Category là bảng chỉnh, trong 1 category có các sp thì bảng sp đó là bảng con, là bảng phụ thuộc

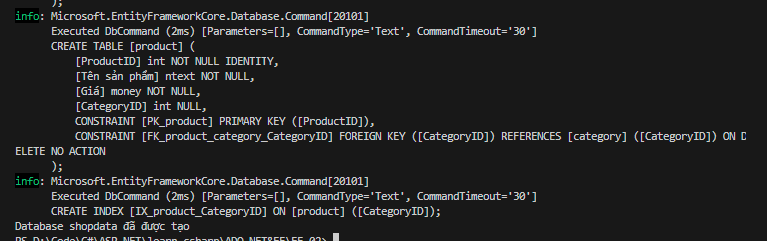
Để tạo ra bảng phụ thuộc, trong phần khai báo model ta tạo ra 1 property có kiểu là Category

Bằng cách khai báo trong product như vậy

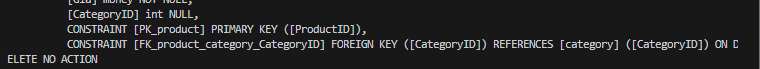
Căn cứ vào property EF sẽ tạo ra mối liên hệ và mối liên hệ này tạo ra khóa ngoại (FK)

FK sẽ tham chiếu đến PK của bảng chính

Ta xóa DB và Tạo lại:

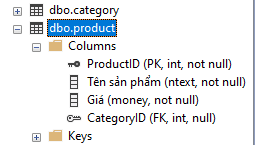


Ta thấy trong log khi tạo ra bảng Product nó đã tạo ra 1 trường dữ liệu là CategoryID

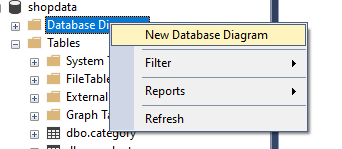
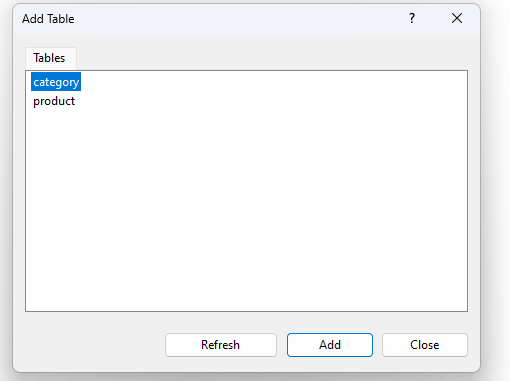


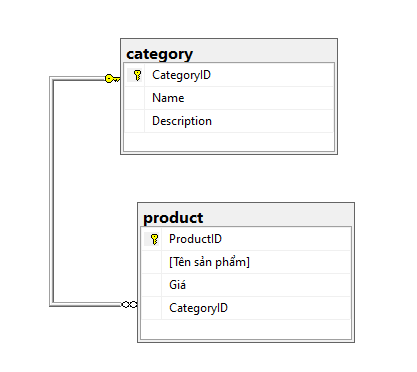
Trường dữ liệu này được thiết lập là khóa ngoại và tham chiếu đến bảng category dự trên khóa ngoại này

Như vậy mỗi khóa ngoại CategoryID thì mỗi Product chúng ta xác định được nó thuộc category nào, tác động của mối quan hệ này là khi chúng ta xóa 1 category thì những sản phẩm của category đấy không bị xóa (ON DELETE NO ACTION)



Để tạo mô hình trực quan ta làm như sau:

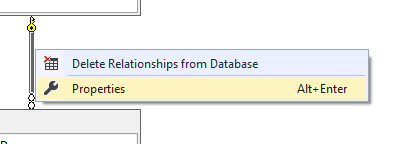
2 bảng này có mỗi liên hệ

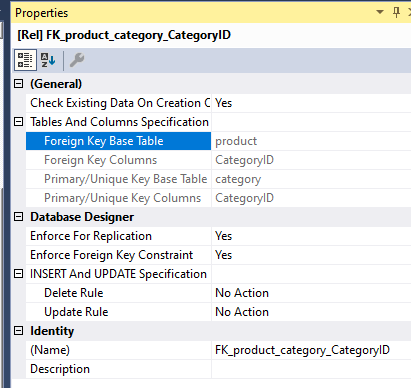
Dấu vô cực ý là quan hệ nhiều thể hiện khóa ngoại

Chìa khóa là khóa chính

Nhìn vào đây ta sẽ hiểu rằng mối quan hệ giữa category và product là qua hệ 1 - nhiều

Để xem chi tiết mối liên hệ ta có thể chuột phải vào mối liên hệ

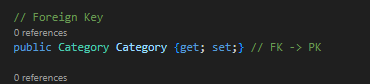


 Mối liên hệ này được thiết lập giữa trường CategoryID của bảng Product và trường CategoryID của bảng Category

Phần INSERT and UPDATE

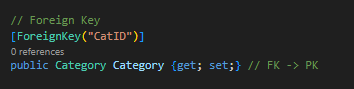
Cả delete và update đều là No Action nghĩa là khi xóa đi 1 category thì các sp thuộc category đó không bị xóa đi

Như vậy khi ta khai báo trong model product 1 model khác thì nó sẽ tạo ra mối liên hệ giữa 2 model và qua đó nó sẽ tạo ra mối quan hệ giữa các bảng trên SQL server



Dựa vào khai báo này thì nó sẽ căn cứ vào category truy ngược lại tìm ra khóa chính của category có tên là CategoryID, nó sẽ tự động tạo ra 1 khóa ngoại trong Product và FK này có tên tương ứng với tên khóa chính

Để đổi tên không theo mặc định ta làm như sau:



Nếu chúng ta muốn trong Model Product có luôn thuộc tính là CatID tương ứng với trường dữ liệu CatID ở trên bảng product

Tại đây chúng ta hoàn toàn có thể khai báo 1 thuộc tính



Ở đây property Category hoàn toàn có thể có giá trị là null nên CatID cần thêm dấu hỏi để nó phù hợp

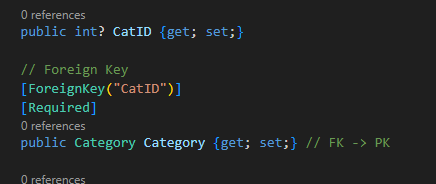


Dấu hỏi này có nghĩa là CatID có thể gán bằng null

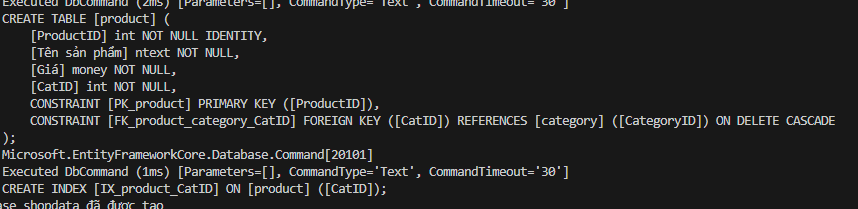
Lúc này trong model Product sẽ có thêm property là CatID

Điều này khi chạy lại nó vẫn tạo ra bảng y như cũ, nhưng lần này ta có thể thao tác với property này

Khi 1 product được thêm vào thì bắt buộc product đó phải thuộc 1 Category lúc này ta có thể thêm thuộc tính require

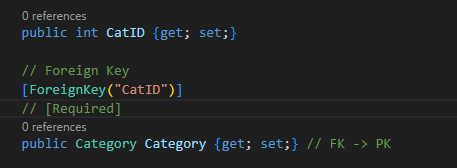


Chạy lại chương trình:



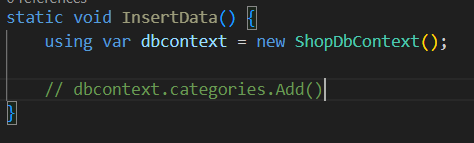
Trường CatID được thiết lập là not null, bây giờ mói liên hệ giữa Category và product là ON DELETE CASCADE tức là khi ta xóa 1 Category thì những product thuộc Category đó cũng bị xóa theo

Cách viết như này cũng tương tự:

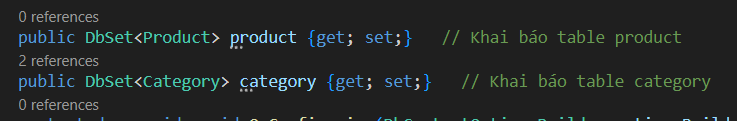


### Chèn dữ liệu mẫu vào CSDL

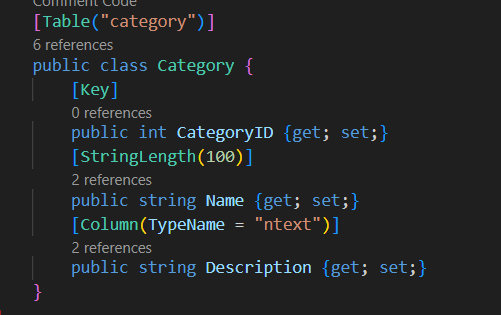
Ta sẽ tạo ra 1 phương thức để thực hiện chèn dữ liệu mẫu

Tạo phương thức InsertData để thêm dữ liệu vào bảng

Trong phương thức ta phải tạo ra dbContext



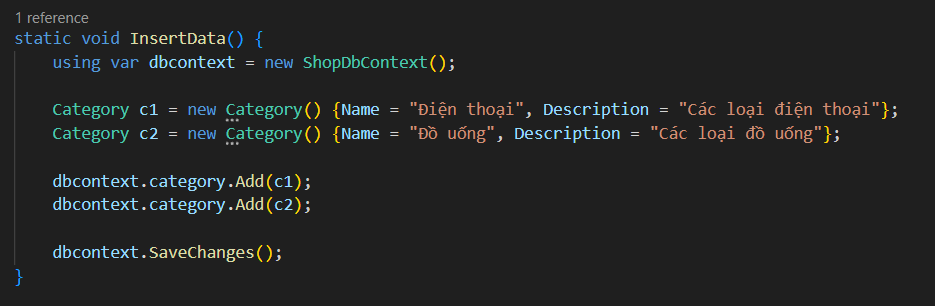
Trong ShopDBContext có 2 table trước tiên ta sẽ thêm category vào bảng category

Model Category đại điện cho đối tượng category có property CategoryId là khóa chính nên nó tụ động phát sinh

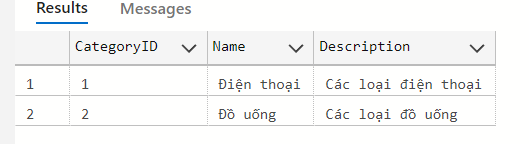
Chúng ta sẽ thiết lập 2 thuộc tính còn lại



Sau đó ta sẽ thêm vào table thông qua dbcontext và gọi phương thức SaveChange



Kết quả:



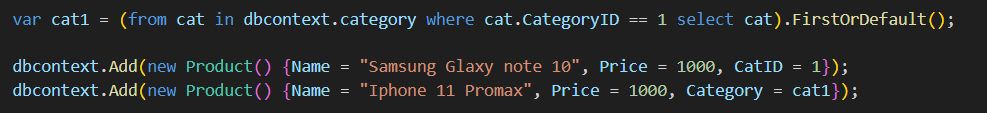
Tiếp theo ta thêm product

Ta có thể dùng nhưu cách bên trên hoặc ta có thể làm như sau:

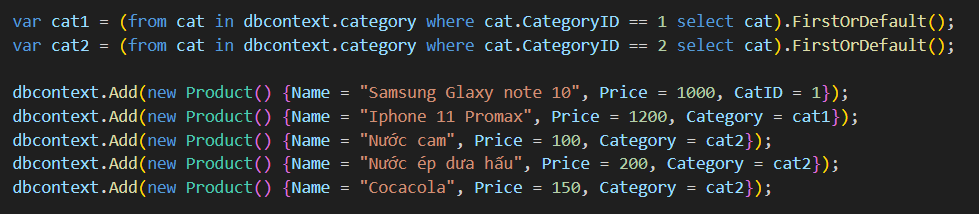


Ta có thể khai báo category theo property CatID hoặc có thể sử dụng theo property Category nếu đã có table của Category như sau:

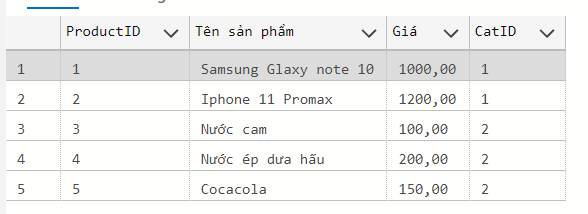
Sử dụng LinQ hãy nhớ import System.Linq



Ta thêm tương tự 1 số product thuộc CategoryID 2

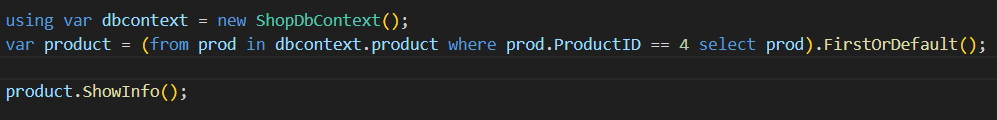


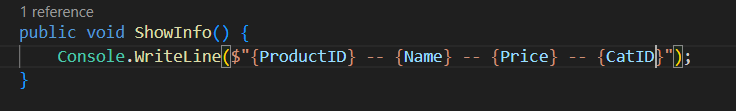
Kết quả khi chạy dự án:



### Truy vấn lấy thuộc tính tham chiếu Reference Navigation

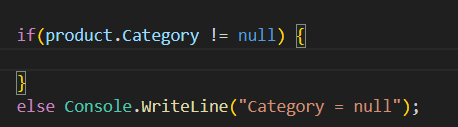
Thử truy vấn lấy ra nước ép dưa hấu:







Ta thấy CatID = 2 ta thử kiểm tra xem Category có tồn tại hay không bằng cách kiểm tra?



Kết quả:



Vậy là property là 1 model tham chiếu đến table khác khi ta truy vấn nó sẽ không tự động lấy về

Nếu ta muốn lấy được property là một Model tham chiếu đến 1 table khác thì ta làm như sau:

* Khi chúng ta truy vấn được product nghĩa là trong DbContext có tồn tại một đối tượng được sử đụng dể iams sát sự thay đổi của product, đối tượng đó là Entity Entry

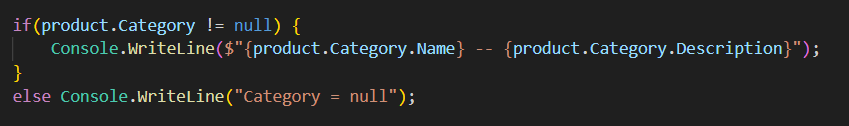


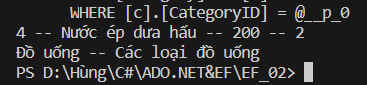
Lúc này ta sử dụng đối tượng Entry để nạp dữ liệu tham chiếu tới 1 model khác thông qua phương thức Reference

Phương thức này nhận thâm số là 1 delegate có dạng như sau:

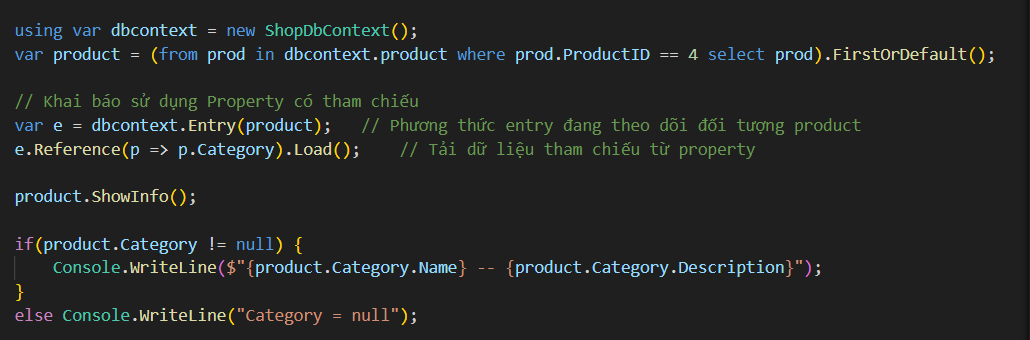


Sau khi thực hiện phương thức Load của entry thì nó sẽ truy vấn và lấy về dữ liệu Category sau đó ta có thể sử dụng dữu liệu này





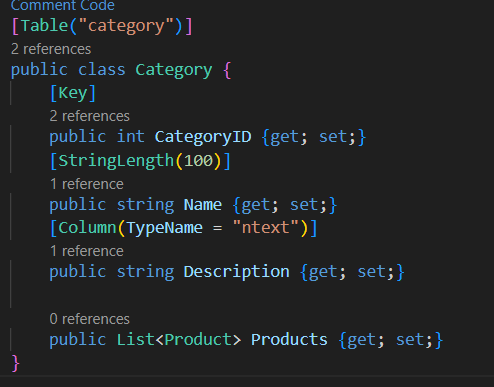
Tóm tắt:



### Tham chiếu tập hợp Collection naviagtion

Product và Category là mối quan hệ 1 - nhiều, trong product mối liên hệ này được tạo ra thông qua Foreign Key

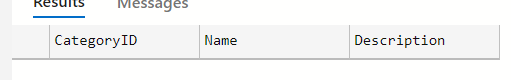
Nếu ta xét trong Category thì mỗi Category có nhiều Product thuộc category đó, vì vậy trong Category ta có thể tạo ra 1 property là một tập hợp các product

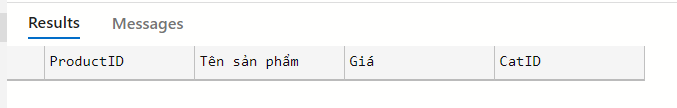


Nếu 1 property trong model là 1 tập hợp chứa những model khác thì cái này gọi là điều hướng tập hợp Collection naviagtion thì nó không tạo ra các foreign key

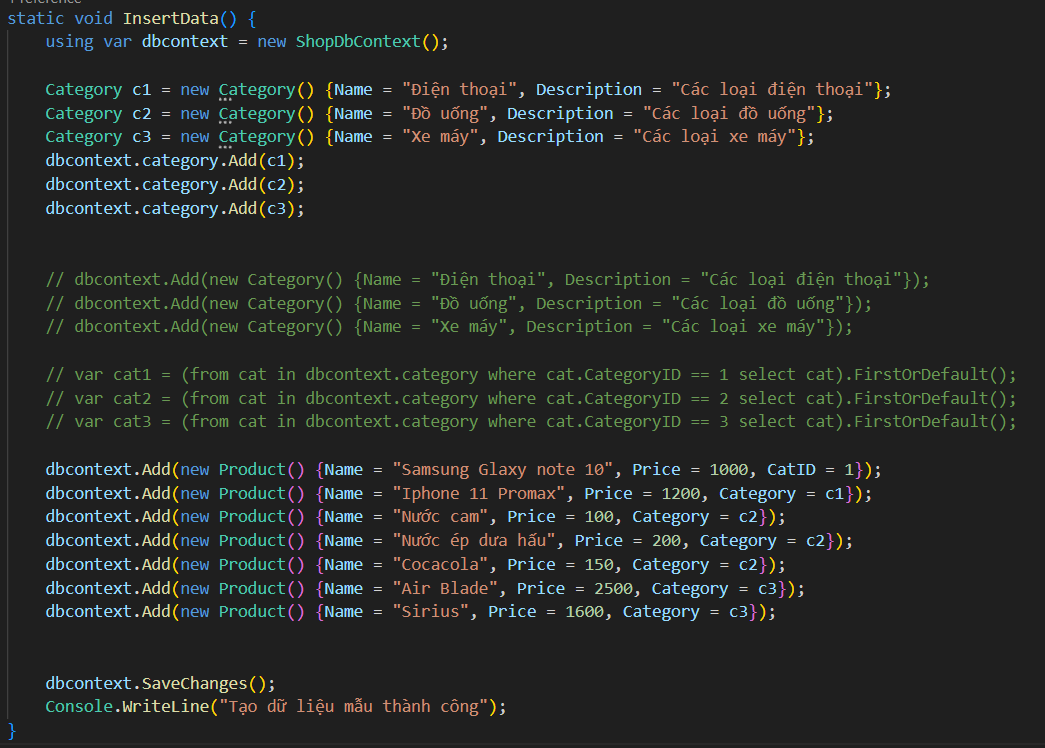
Ta sẽ tạo lại csdl:

Thì ta thấy rằng table products và category vẫn y nguyên như cũ

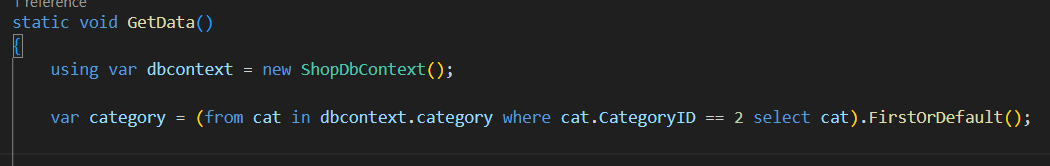




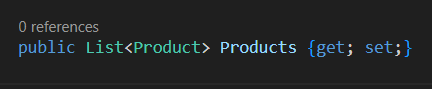
Tiếp đó ta chạy Insert để render lại dữ liệu mẫu



Lấy ra category có id = 2

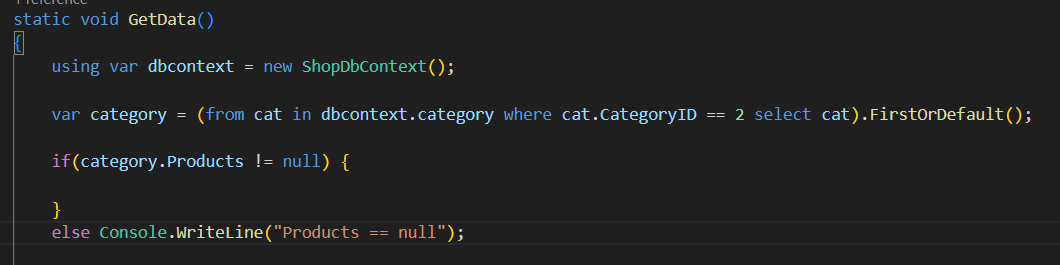


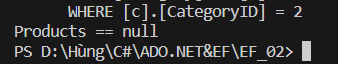
Ta biết là trong category này có property là Products



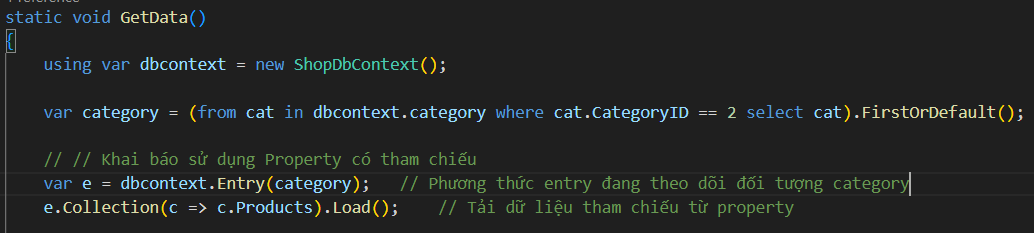
Những nếu ta lấy ra th nóì sẽ không tự động lấy ra được các product thuộc category này

Chúng ta kiểm tra thử xem

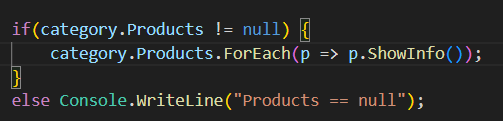




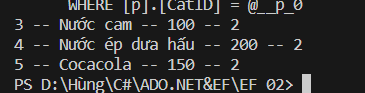
Để lấy được ra Products ta làm như sau:



Sau khi khai báo entry ta thử lấy dữ liệu products



Kết quả:



### Tổng kết

Đến đây ta đã biết được 2 khái niệm:

* **Reference navigation**: tạo ra tham chiếu đến 1 model khác
  + Tham chiếu từ 1 model này đến 1 model khác
  + Khai báo tạo ra Reference navigation thì sẽ tạo ra foreign key
  + Hình thành mối quan hệ 1 - nhiều giữa các bảng
* **Collect navigation**:
  + Không ảnh hưởng đến SQL server, ở đây nó chỉ tạo ra những property để ta có thể truy cập và sử dụng
  + Không tạo ra Foreign Key

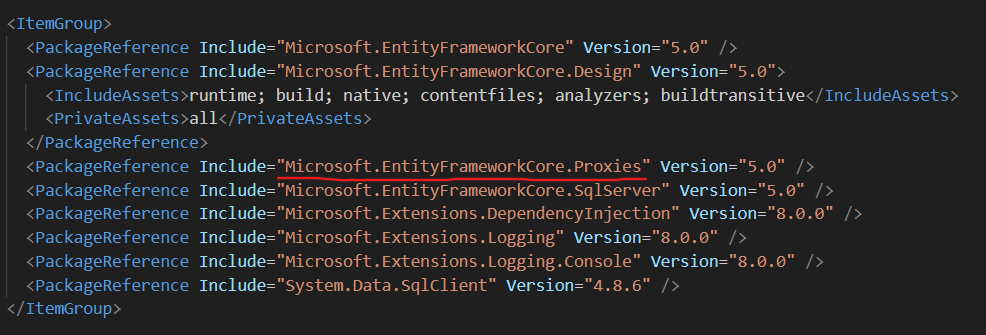
Trong cả 2 trường hợp này khi ta truy vấn để lấy dữ liệu ra thì nó khong tự động nạp navigation vào mà chúng ta phải làm bằng một cách thủ công là phải truy vấn để lấy ra đối tượng **entity entry** sau đó tùy vào navigation đó là reference hay collect sẽ gọi phương thức tương ứng sau đó Load dữ liệu

## Tự động tải các reference

Để tự động tải các reference ta sẽ thông qua 1 thư viện của **Microsoft.EntityFrameworkCore** là **Proxies**

Ta cài đặt package sau:

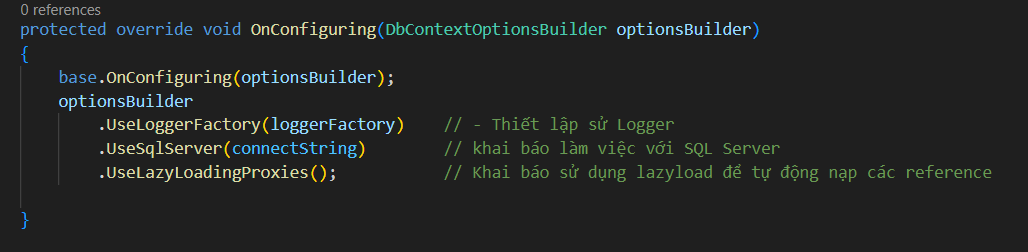
**dotnet add package Microsoft.EntityFrameworkCore.Proxies**

****

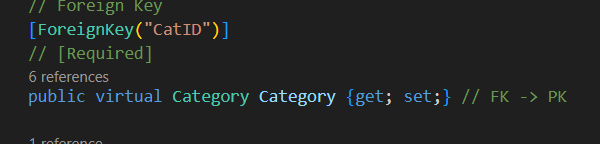
Lưu ý phiên bản

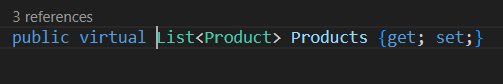
Package Proxies này cho phép chúng ta áp dụng kỹ thuật **lazy load**

Ta sẽ khai báo trong DBContext sử dụng lazyLoad tức là tự động nạp các reference

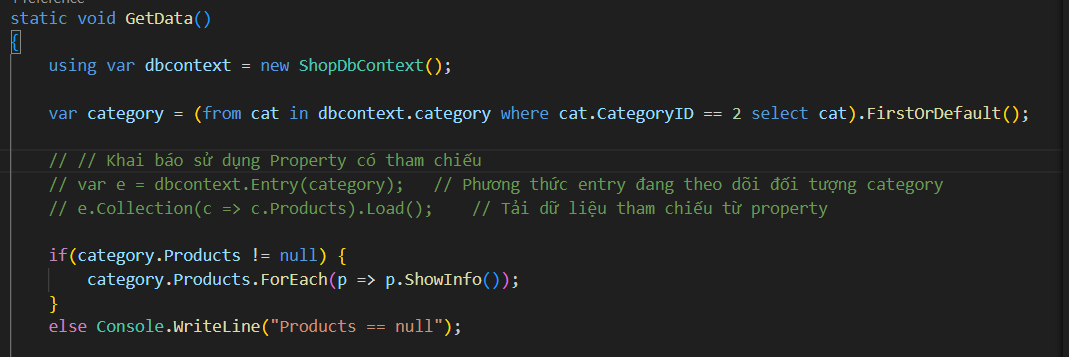


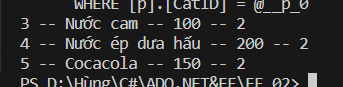
Sau khi khai báo sử dụng Proxies để sử dụng thì trong các property ta cần phải khai báo từ khóa virtual như sau:





Lúc này khi ta nạp category thì tất cả các product trong nó được nạp theo



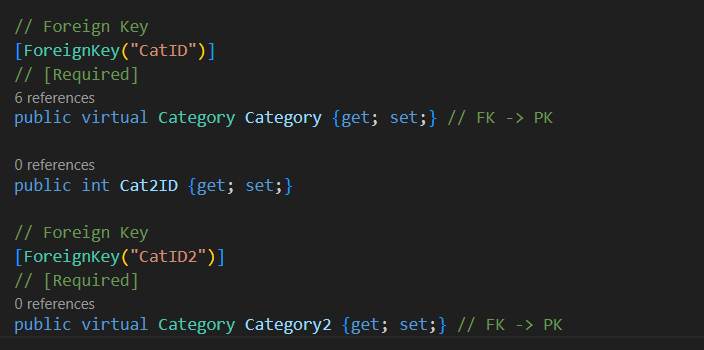


Nếu ta áp dụng kỹ thuật lazyload này thì trên thực tế code chạy sẽ khá là nặng nên không khuyến khích sử dụng

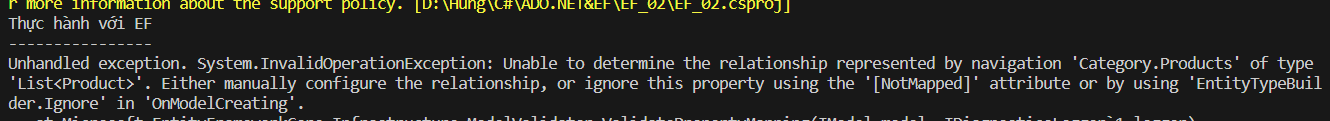
## Inverse Property

Trong model Product ta thấy rằng 1 product chỉ thuộc 1 category nào đó, tuy nhiên trong thực tế ta một product có thể thuộc 2 category

Nếu vậy ta sẽ khai báo để tạo ra thêm 1 FK nữa tức là nó tham chiếu đến 2 category thông thường theo suy nghĩ ta sẽ làm như sau:



Nhưng nếu làm vậy chương trình sẽ bị lỗi



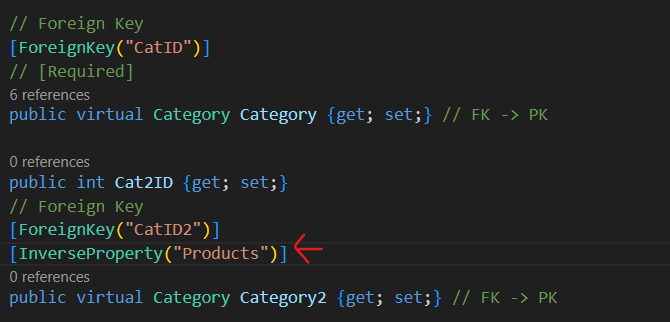
Lý do là:

Trong Category, khi lấy ra 1 category thì sẽ lấy được ra products thuộc category

Các products thuộc category lấy ra được căn cứ vào FK, vấn đề ở đây là có 2 FK do đó collection product không biết sử dụng FK nào

Lúc này ta phải chỉ rõ trong product là collection tương ứng với những FK nào (CatID2 hay CatID)

Ví dụ ta thiết lập collection trong Category sẽ tương ứng với CatID2 tại đây ta sẽ thiết lập 1 attribute là **“điều hướng ngược”** tới cái property nào của Category:



Kết quả:

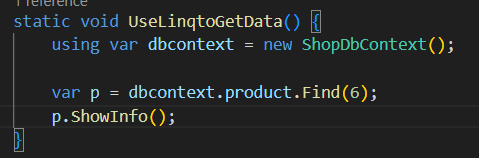


Lưu ý không được có 2 điều kiện ON DELETE CASCADE, vì SQL có thể dẫn tới 1 vòng lặp xóa

## Thực hành truy vấn dữ liệu với Linq

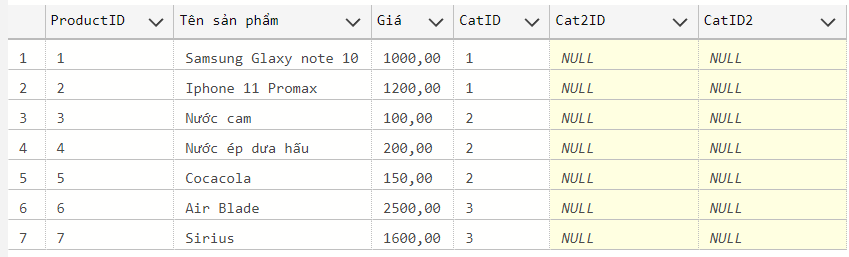
Ta thấy rằng trong product có 1 PK là ProductID, trong EF hõ trơ việc tìm 1 hàng nào đó theo PK thông qua phương thức Find

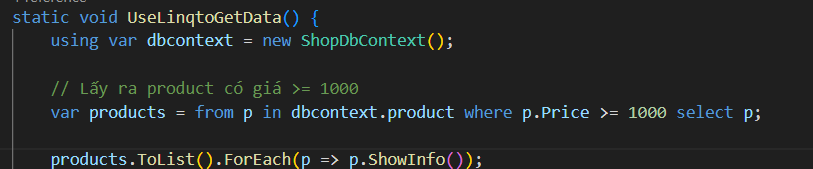
VD lấy ra product có id = 6;

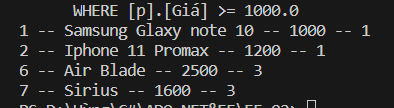




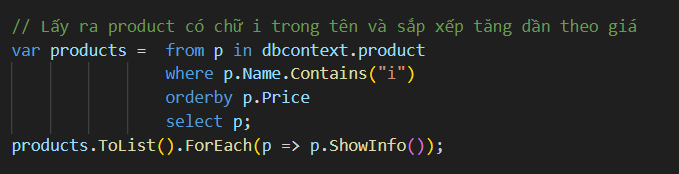
Thử áp dụng lấy ra các product có price >=1000



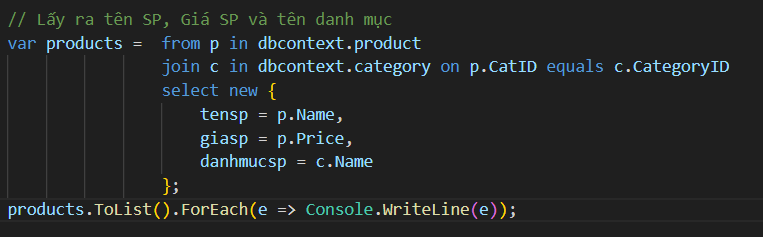


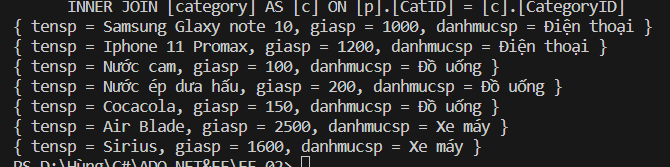


Thêm VD:



VD: truy vấn Join giữa 2 table

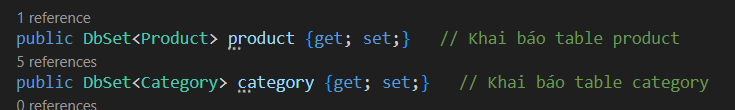


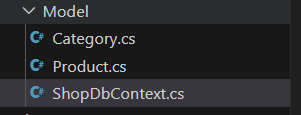


# Fluent API

## Vấn đề

Trong bài trước ta đã tạo ra 1 dbcontext là shop context

Trong ShopDbContext chứa 2 table tương ứng với 2 model



Khi ta tạo ra các model ta đã dùng các attribute để cấu hình các entity và các mối quan hệ

Sử dụng attribute thì sẽ dễ đọc và thiết lập được một cách nhanh chóng

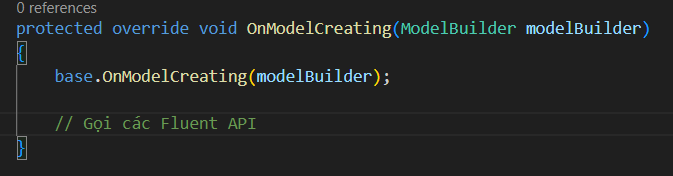
Tuy nhiên có nhiều các mối quan hệ, các thiết lập àm bản thân những attribute này không giải quyết được hoặc giải quyết rất phức tạp

Lúc này ta có thể sử dụng đến các API trong EF là Fluent

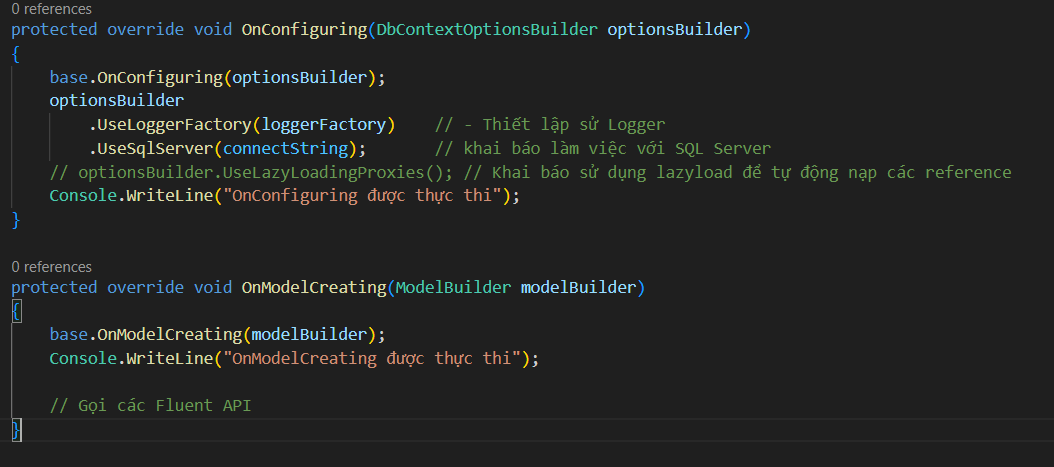
## Sử dụng

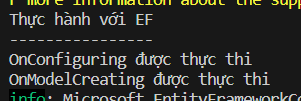
Để sử dụng Fluent API ta sẽ thiết lập như sau:

* Trong ShopDbContext ta đã nạp chồng 1 phương thức là OnConfiguring, phương thức này được gọi khi khởi tạo ra BbContext
* Có 1 phương thức nữa cũng được gọi khi khởi tạo DbContext là OnModelCreating. Trong model này các Fluent API sẽ được thực thi



Phương thức này được thực thi sau khi hoàn thành OnConfiguring

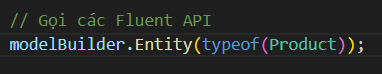




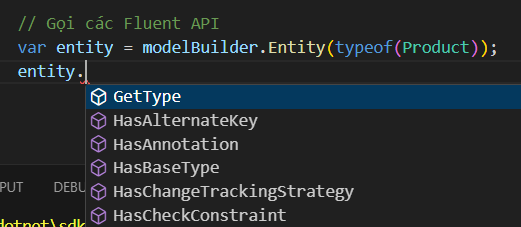
## Lấy ra và cấu hình các entity

* Trong phương thức OnModelCreating có 1 đối tượng kiểu ModelBuilder
* Tại đây ta có thể lấy ra được các Entity (Các đối tượng chứa cấu hình của các model) từ đối tượng này ta sẽ gọi được những phương thức của Fluent API

VD: ta thiết lập cho Product, ta sẽ lấy entity biểu diễn cho model product này

Tham số của phương thức này là chỉ ra kiểu của Model

Phương thức này sẽ trả về 1 đối tượng là EntityTypeBuilder

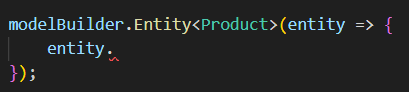
Từ đối tượng EntityTypeBuilder này ta gọi được các phương thức của nó

Các phương thức này là các Fluent API

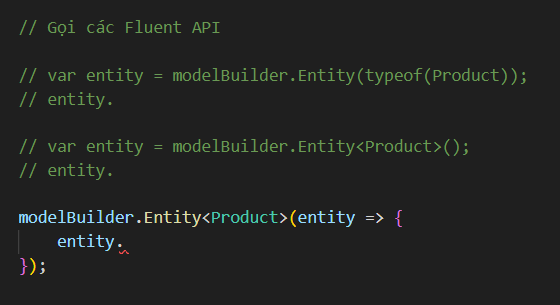
Chúng ta còn 1 cách khác để sử dụng fluent api mà không phải sử dụng typeof



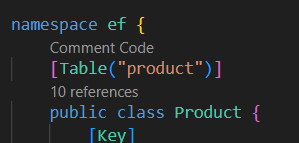
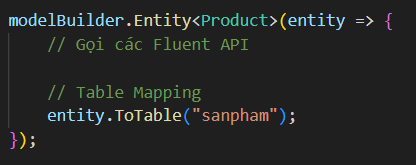
Còn 1 cách viết nữa mà chúng ta sẽ ưu tiên sử dụng:



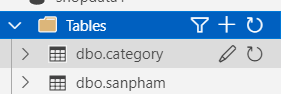
Như vậy ta có 3 cách gọi fluent api và ta sẽ ưu tiên sử dụng cách 3



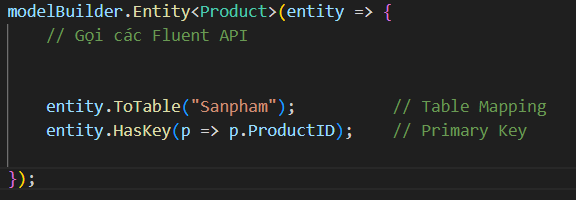
## Ánh xạ bảng với ToTable (Table mapping)

  Sử dụng như thế này tương tự như sử dụng Attribute

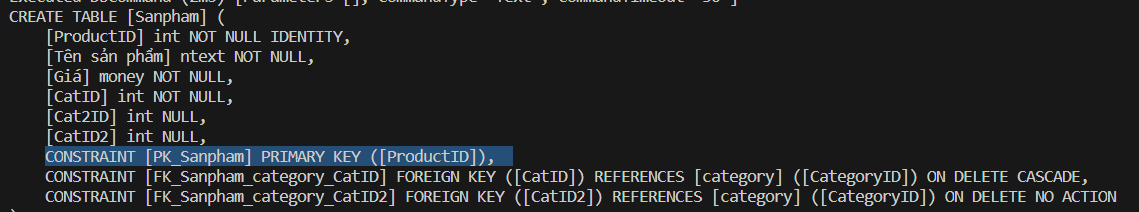
Kết quả:



## Thiết lập khóa chính với HasKey

Phương thức HasKey nhận tham số là 1 biểu thức chỉ ra trường dữ liệu là PK

Kết quả:

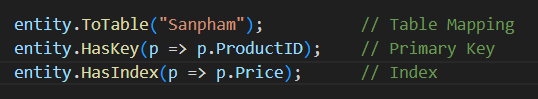


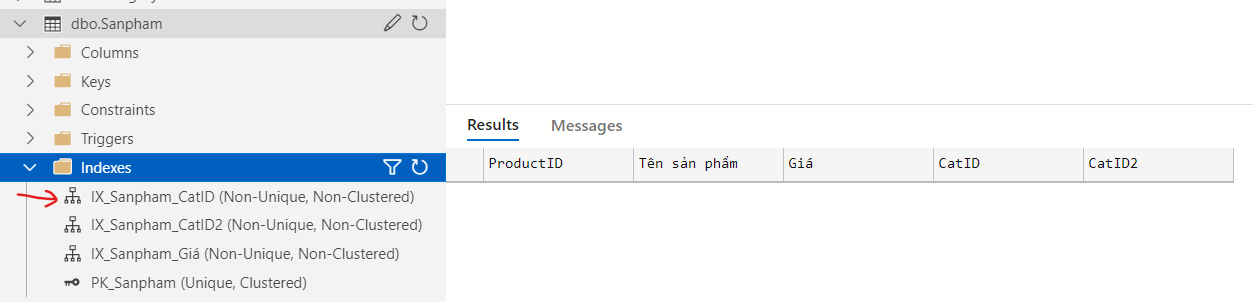
## Đánh chỉ mục (index)

Index là phần kiến thức thuộc SQL Server nhằm tăng tốc độ truy vấn, tốc độ tìm kiếm

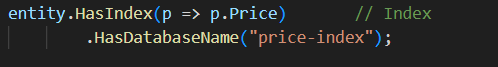
Trong .Net Core không sử dụng được Attribute để tạo ra các index nhưng fluent API thì có thể

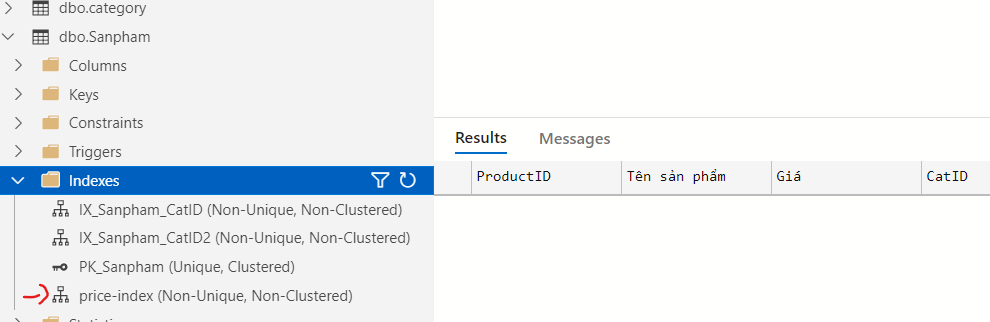
VD ta đánh chỉ mục cho price



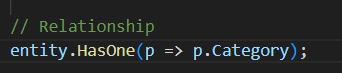


Ta cũng có thể đặt tên cho chỉ mục như sau:





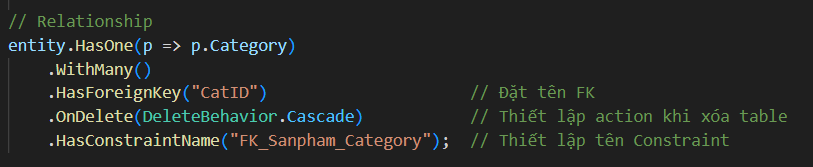
## Mối liên hệ giữa các model

Phương thức HasOne dùng để khai báo mối quan hệ 1 – nhiều nhận tham số là phần 1 trong 1 – nhiều ở đây là Category

Phương thức này trả về 1 đối tượng là ReferenceNavigationBuilder và tạo ra mối quan hệ 1 - nhiều



Ta có thể thiết lập thêm thông qua đối tượng ReferenceNavigationBuilder mà nó trả về



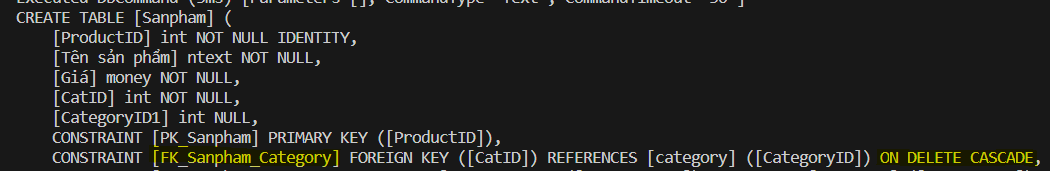
**.WithMany:** Thiết lập thuộc tính ở phần nhiều. ta sẽ không điền gì vì Category không có peroperty nào là các product

**.HasForeignKey** dùng để đặt tên cho FK

**.OnDelete** dùng để cấu hình hành động khi xóa table:

* **DeleteBehavior.Cascade**: Khi xóa category thì các product có mối liên hệ sẽ bị bị xóa đi theo
* **DeleteBehavior.SetNull**: Khi xóa category thì các product có mối liên hệ không bị xóa đi theo. Lưu ý trường dữ liệu tương ứng trong trường hợp này phải được phép null

**.HasConstraintName:** Thiết lập tên Constraint



Tiếp tục khởi tạo FK thứ 2:

entity.HasOne(p => p.Category2)

      .WithMany(c => c.Products)

      .HasForeignKey("CatID2")

      .OnDelete(DeleteBehavior.Restrict)

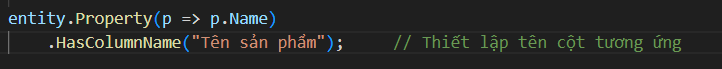
      .HasConstraintName("FK\_Sanpham\_Category2");

Với **HasOne** phần 1 lúc này là Category2

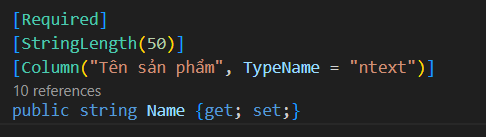
.**WithMany** chỉ ra thuộc tính tương ứng với phần nhiều nếu có. Ở phần FK trước ta đã không chỉ ra phần nhiều nhưng tại thuộc tính thứ 2 này ta đã thiết lập phần nhiều với FK thứ 2 thương ứng là Products tức là khi lấy ra 1 Category ta sẽ lấy ra luôn được tất cả product thuộc Category đó điều này trước đó ta đã chỉ ra bằng attribute InverseProperty vì vậy trong Fluent API ta sẽ sử dụng WithMany

## Thiết lập các trường (cột) dữ liệu property của bảng

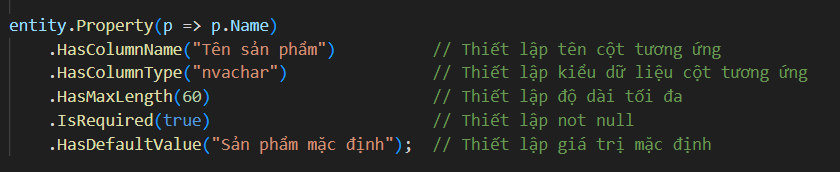
Ta sẽ sử dụng Fluent API để thiết lập các Property



Tương ứng với



Ta nhận thấy rằng ta có thể thiết lập đồng thời cả Atrribute lẫn Fluent API nhưng Fluent API sẽ có hiệu lực cuối cùng (ghi đè lên Attribute)



Ta thấy rằng nếu sử dụng được Attribute thì nên sử dụng còn nếu không thể sử dụng Attribute ta sẽ sử dụng Fluent API

## Tạo mối liên hệ 1 -1

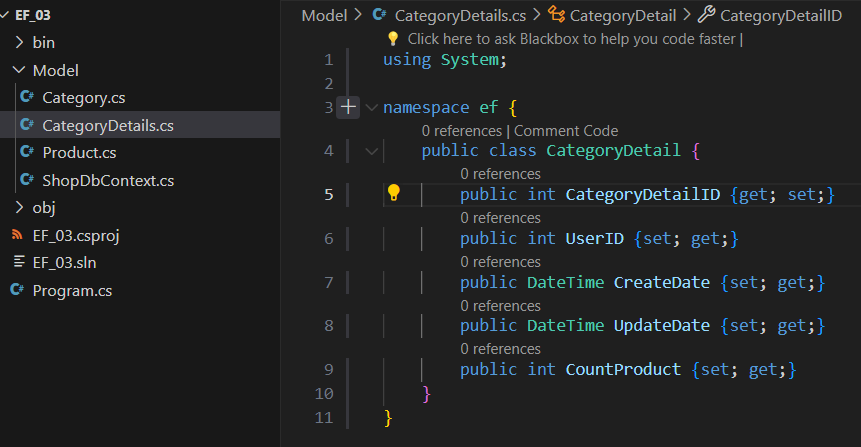
Trong thực tế 1 table thường có rất nheieuf dữ liệu

VD ta thêm các trường dữ liệu mới vào Category

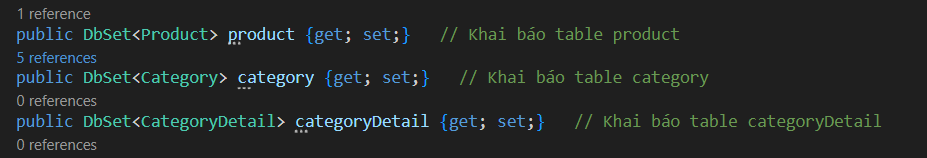


Trong tường hợp có quá nhiều property ta có nhu cầu tách nó ra thành nhiều bảng ta có thể đưa 1 số trường dữu liệu sang 1 bảng khác và table mới đó với table Category phải hình thành mối quan hệ 1 – 1

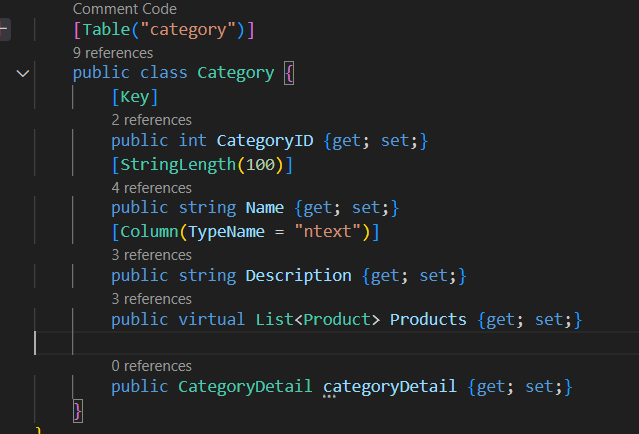
Giả sử ta tạo 1 model là CategoryDetail và chuyển một số trường dữ liệu sang đây



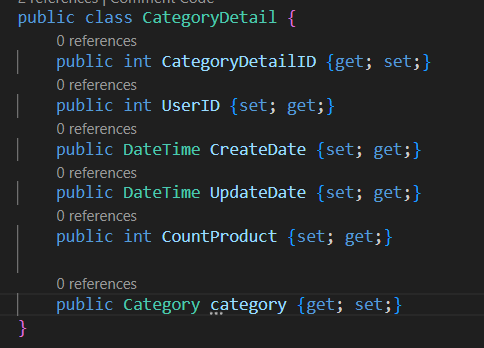
Sau đó ta sẽ đưa nó vào dbContext



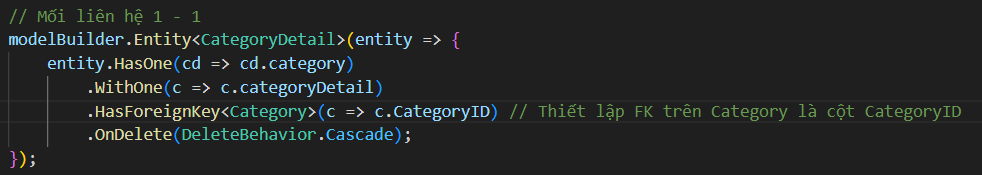
Trong Category lúc này sẽ có thêm 1 trường dữ liệu là CategoryDetail



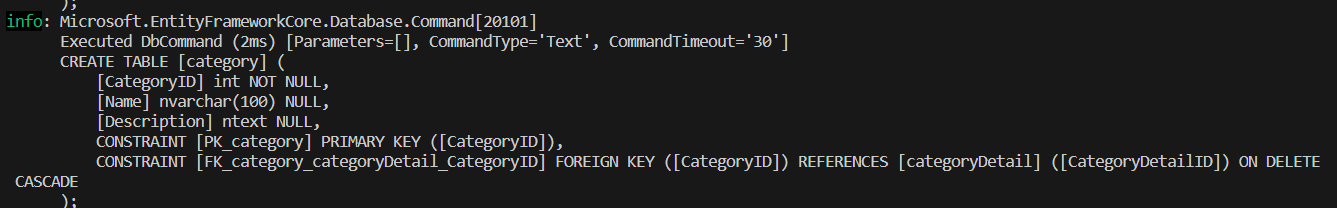
Và CategoryDetail sẽ có thêm 1 trường dữ liệu là Category

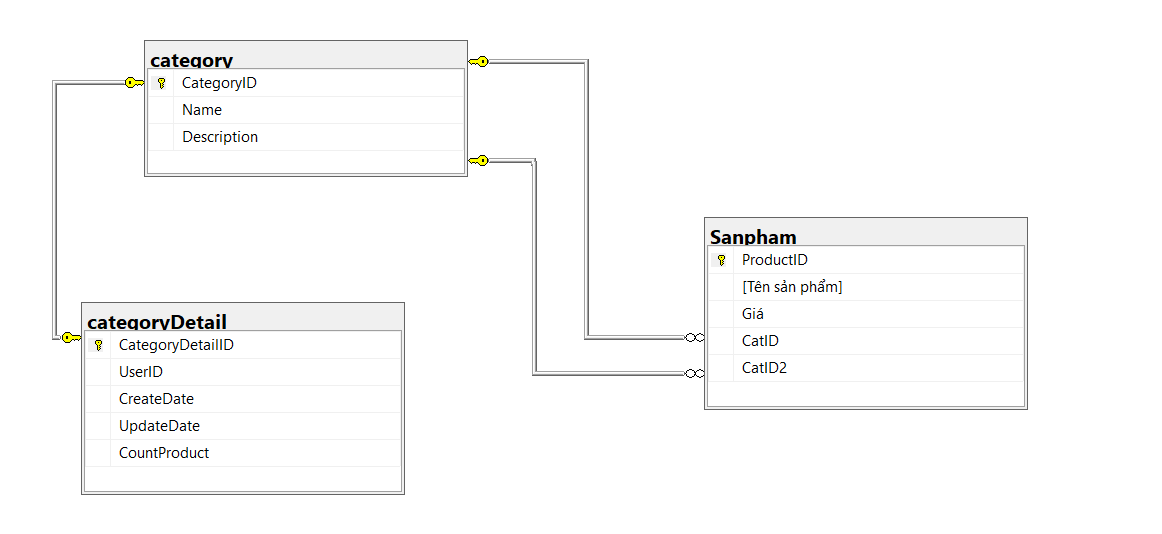


Trong Fluent API ta khai báo mối quan hệ 1 -1 như sau:



Kết quả:





# Migration

Migration là cách thức tạo ra, cập nhật, chỉ sửa cấu trúc của CSDL,

Trong đó ta viết Code để biểu diễn CSDL, Model, mối quan hệ

Từ Code đó ta sẽ cập nhật lên DB

Trước khi thực hành ta cần đảm bảo dự án đã cài đặt đủ các package:

dotnet add package System.Data.SqlClient

dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

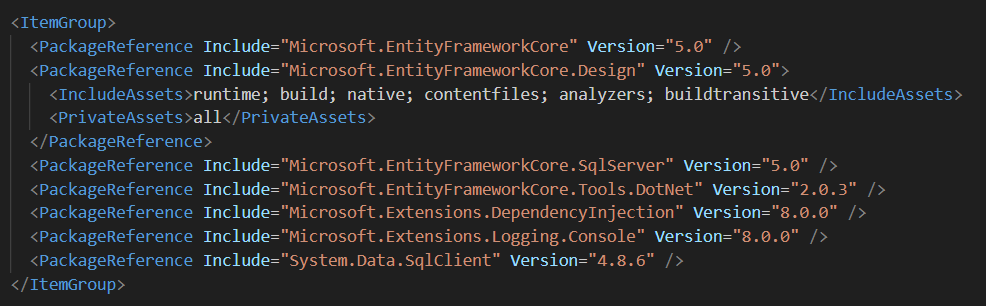
dotnet add package Microsoft.EntityFrameworkCore.Design

dotnet add package Microsoft.Extensions.DependencyInjection

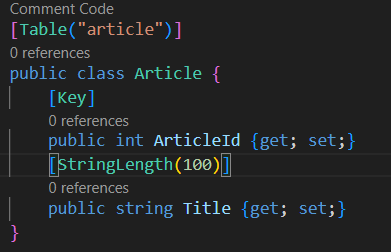
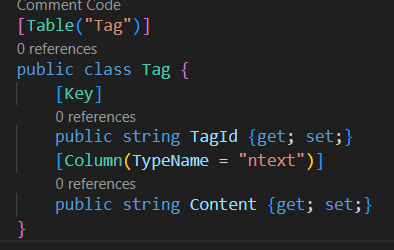
dotnet add package Microsoft.Extensions.Logging.Console

dotnet add package Microsoft.EntityFrameworkCore.Design

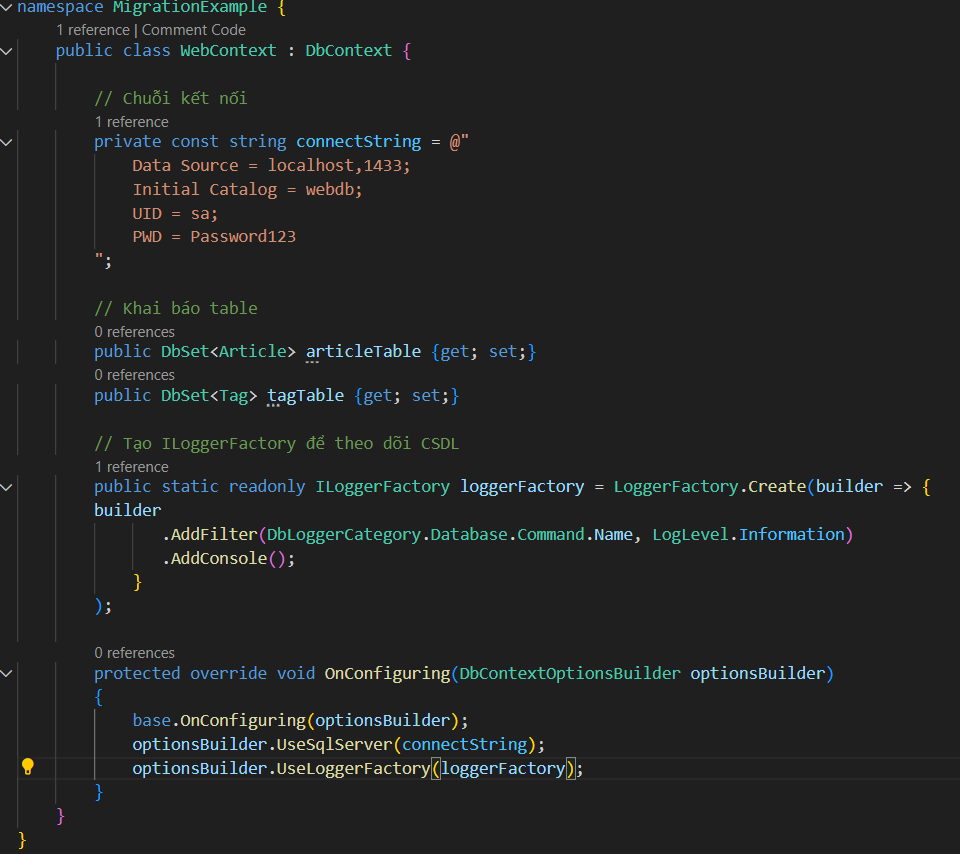
dotnet add package Microsoft.EntityFrameworkCore.Tools.DotNet



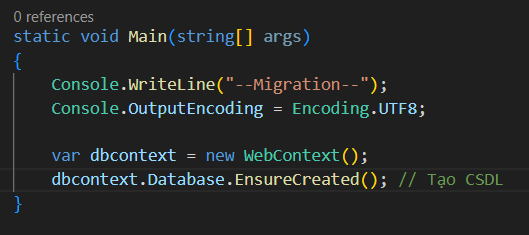
Trước mắt ta sẽ tạo ra 2 bảng Article và Tag  
Tạo model

Cấu hình DbContext



Ở bài trước ta đã tạo ra CSDL nhưu sau:



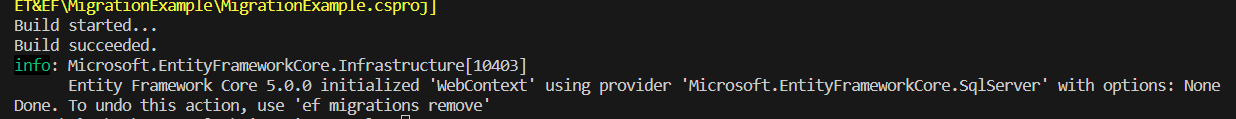
Nhưng ở bài này ta sẽ không làm như vậy mà dùng kỹ thuật migration

Qua các thao tác ở trên amwjc dù CSDL ta chưa có nhưng chúng ta đã có cấu trúc ban đầu của CSDL đó do đó ta có thể tạo ra một ảnh chụp, snapshot của CSDL ở thời điểm này. Điều này gọi là migration

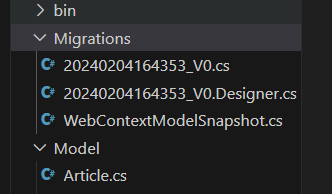
## Tạo migration

**Dotnet ef migrations add <tên migration>**





Chúng ta vừa tạo ra một migration có tên là V0

Lúc này trong dự án phát sinh 1 file có tên là Migrations trong đó nó phát sinh ra những fiel code cs cho chúng ta

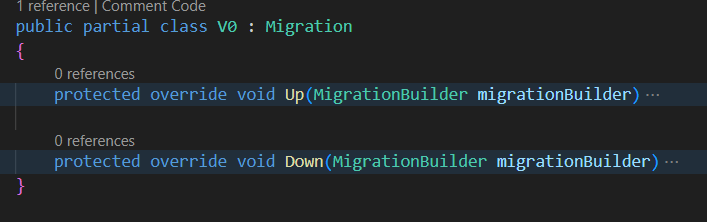
Trong đó file WebContextModelSnapshot chứa cấu trúc CSDL ở thời điểm mà ta tạo ra migration V0

Đồng thời nó cũng sinh ra 2 file tương ứng có dạng là

<time>\_V0.cs

<time>\_V0.Designerr.cs

File <time>\_V0.Designerr.cs chứa cấu trúc CSDL ở thời điểm mà ta tạo ra migration V0

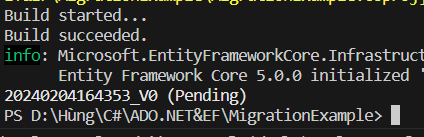
File <time>\_V0.cs chứa 2 phương thức Up và Down

Phương thức Up được thực thi khi ta sử dụng entity framework để cập nhật cấu trúc CSDL ở thời điểm V0 lên DB

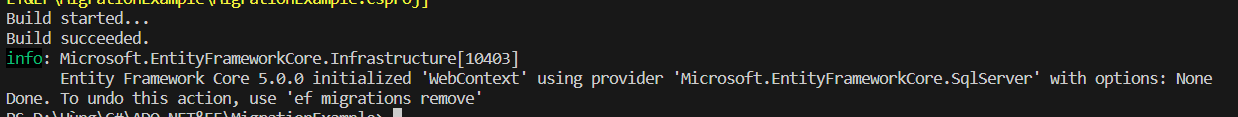
Phương thức Down sử dụng khi muốn phục hồi, hủy bỏ cập nhật của phiên bản

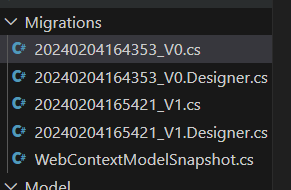
Để kiểm tra xem hiện tại có migration nào ta thực hiện lệnh:

**Dotnet ef migrations list**

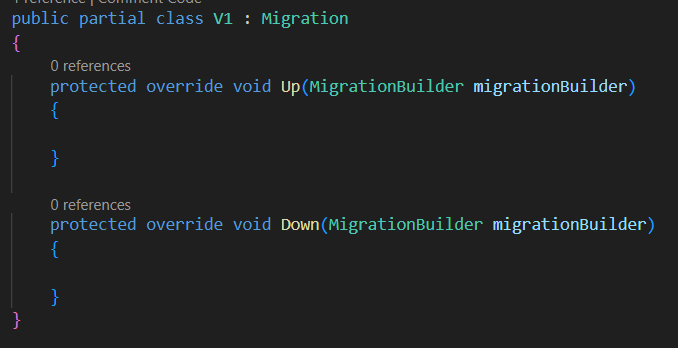
Pending là phiên bản này chưa từng được cập nhật lên DB

Ta thử tạo thêm 1 migration V1 nữa

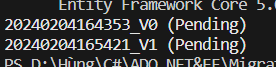


Lúc này V1.cs và V1\_Designer.cs chứa các thông tin thay đổi từ thời điểm V0 sang V1

Và do ta chưa sửa gì code nên Phương thức Up và Down trong V1 sẽ không có gì



Chúng ta kiểm tra danh sách migration



## Xóa migartion

Xóa 1 migrations ta làm như sau:

**Dotnet ef migrations remove**

Lệnh này sẽ xóa đi migration ta tạo ra cuối cùng

Kiểm tra lại danh sách sau khi xóa

## Cập nhật migration lên SQL server

Để cập nhật 1 migration lên DB ta sử dụng lệnh sau:

**Dotnet ef database update**

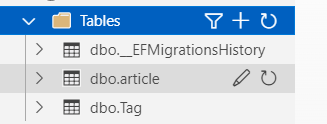
Trong trường hợp này nó sẽ update tất cả các migration cho đến migration cuối cùng

Nếu ta muốn update hoặc quay trở về 1 phiên bản nào đó thôi thì ta sẽ làm như sau:

**Dotnet ef database update <Tên migration>**

****Sau khi chạy lệnh ta kiểm tra danh sách migartion và thấy V0 không còn trạng thái pending:

Kiểm tra trên SQL server

Ta nhận thấy ngoài DB và 2 table ta tạo còn xuất hiện thêm 1 bảng nữa

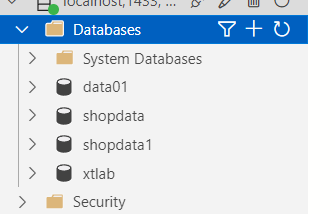
Bảng này chứa tất cả các phiên bản migration ta đã cập nhật lên DB

## Xóa Database

Để xóa DB t sử dụng lệnh sau:

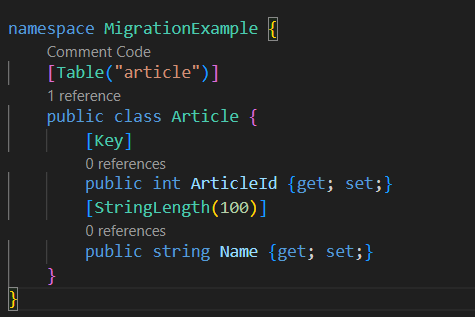
**Dotnet ef database grop -f**

****

CSDL webdb đã biến mất

Lúc này migration V0 lại chuyển về trạng thái Pending

Ta sẽ làm 1 số chỉnh sauwr nhỏ và tạo ra migration tiếp theo

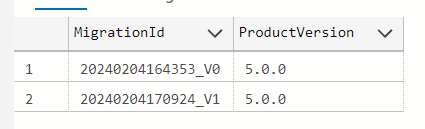
Trong model Article ta sẽ sửa property Title thành Name

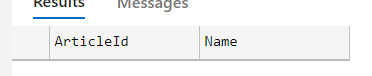
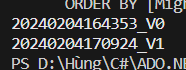
Vậy là code đã có sự thay đổi so với thời điểm V0. Để cập nhật dữ liệu lên DB ta sẽ tạo ra 1 Migration

Lúc này trong file V1.cs đã có sự thay đổi:



Sau đó ta thực hiện lệnh update



## Quay về phiên bản migration cụ thể

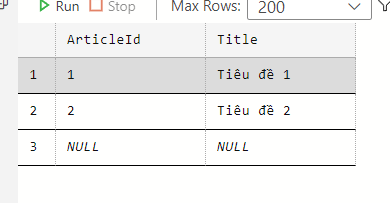
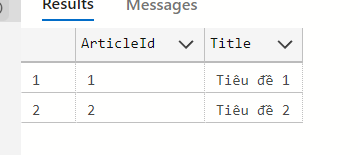
Nếu vì 1 lý do gì đó ta muốn lấy lại phiên bản code ở thời điểm V0 ta sẽ làm như sau:

**Dotnet ef migrations update <phiên bản cần lấy>**

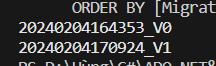
Chúng ta lưu ý rằng khi ta thực hiện migartion thì EF sẽ thay đổi cấu trúc CSDL và cố gắng giữu lại dữ liệu đã cập nhật trước đây

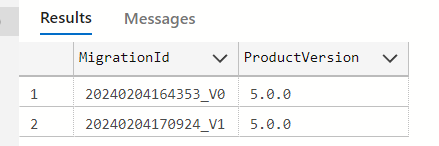
Ta thử thêm 2 dữ liệu vào article

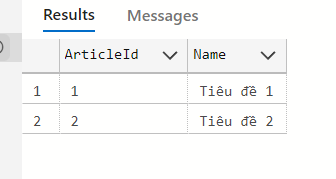
 

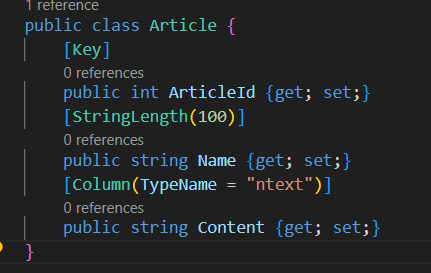
Giờ ta thử cập nhật lên V1



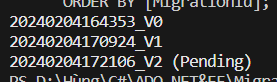




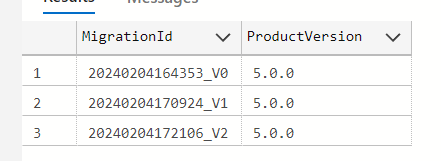
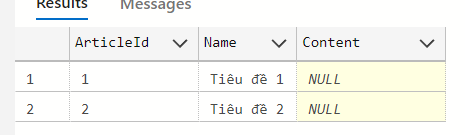
Cột title đã trở thành Name

Tiếp tục ta thêm 1 property nữa vào article

Tiếp đó ta tạo 1 phiên bản migartion V2



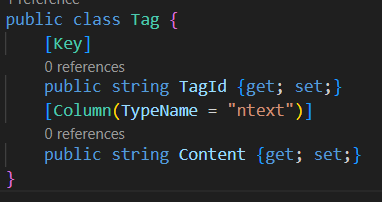
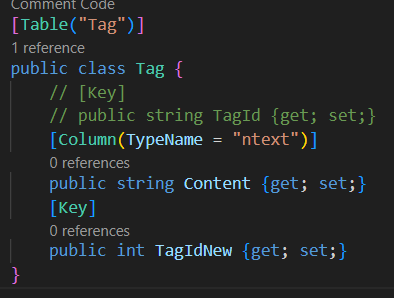
Cập nhật lên DB

## Thay đổi khóa chính của 1 table

Để thay đỏi 1 trường dữ liệu là khóa chính ta không thể làm như thông thường mà sẽ cần phải làm các bước sau:

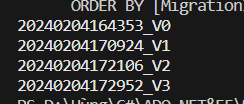
VD: đổi kiểu của TagId thành int

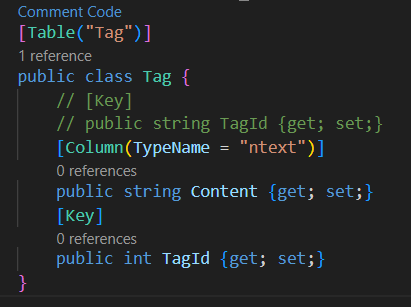


Comment khóa chĩnh cũ để migration hiểu rằng cột này sẽ bị xóa

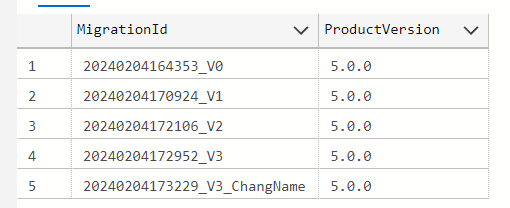
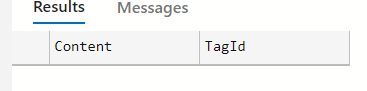
Tiếp đó tạo ta 1 property đóng vai trò làm khóa chính mới

Tạo 1 migration mới

Và sau đó update lên DB 

Đổi tên lại khóa chính

Sau đó tiếp tục tạo migration và update lên DB

Kết quả:

## Mối quan hệ nhiều - nhiều