

Bài 11: Truy vấn và xử lý dữ liệu với Entity Framework Core.....	2
1. LINQ (Language Integrated Query)	2
1.1. Truy vấn là gì và truy vấn làm gì?	3
1.2. Biểu thức truy vấn (query expression) là gì?	4
1.3. Biến truy vấn (Query variable)	4
1.4. Xây dựng biểu thức truy vấn.....	7
1.5. Cú pháp truy vấn và cú pháp phương thức	10
2. ORM và EF.....	11
2.1. ORM (Object Relational Mapping)	11
2.2. EF – ENTITY FRAMEWORK.....	12
3. Entity Framework Core (EF Core)	13
4. Truy vấn dữ liệu sử dụng EF Core	14
4.1. Thêm các package cần thiết.....	14
4.2. Tạo model	18
5. Cập nhật cơ sở dữ liệu sử dụng EF core	25
5.1. Thêm mới dữ liệu	25
5.2. Sửa dữ liệu.....	26
5.3. Xóa dữ liệu.....	26

Bài 11: Truy vấn và xử lý dữ liệu với Entity Framework Core

1. LINQ (Language Integrated Query)

LINQ (ngôn ngữ truy vấn tích hợp) là tên của một tập hợp các công nghệ dựa trên việc tích hợp các khả năng truy vấn trực tiếp vào ngôn ngữ C#. Theo truyền thống, các truy vấn dữ liệu được thể hiện dưới dạng các chuỗi đơn giản mà không cần kiểm tra kiểu tại thời điểm biên dịch hoặc hỗ trợ IntelliSense. Hơn nữa, bạn phải học một ngôn ngữ truy vấn khác nhau cho từng loại dữ liệu nguồn: cơ sở dữ liệu SQL, tài liệu XML, các dịch vụ Web khác nhau, v.v.

Với LINQ, truy vấn có cấu trúc lớp, gồm các lớp, phương thức, sự kiện. Bạn viết các truy vấn đối với các tập hợp kiểu, bằng cách sử dụng các từ khóa của ngôn ngữ và các toán tử quen thuộc. Họ công nghệ LINQ cung cấp trải nghiệm truy vấn nhất quán cho các đối tượng (LINQ to Object), cơ sở dữ liệu quan hệ (LINQ to SQL) và XML (LINQ to XML).

Đối với nhà phát triển, là người viết truy vấn, phần "tích hợp ngôn ngữ" dễ thấy nhất của LINQ là biểu thức truy vấn. Các biểu thức truy vấn được viết theo cú pháp khai báo truy vấn. Bằng cách sử dụng cú pháp truy vấn, bạn có thể thực hiện các thao tác lọc, sắp xếp và nhóm dữ liệu với việc viết code ít nhất. Bạn sử dụng cùng các mẫu biểu thức truy vấn cơ bản để truy vấn và chuyển đổi dữ liệu trong cơ sở dữ liệu SQL, ADO.NET Datasets, luồng và tài liệu XML và các tập hợp .NET.

Bạn có thể viết các truy vấn LINQ trong C# cho cơ sở dữ liệu SQL Server, tài liệu XML, các Dataset của ADO.NET và bất kỳ tập hợp đối tượng nào hỗ trợ giao diện IEnumerable hoặc IEnumerable<T>. Hỗ trợ LINQ cũng được cung cấp bởi các bên thứ ba cho nhiều Web services và các triển khai cơ sở dữ liệu khác.

Ví dụ sau đây cho thấy một hoạt động truy vấn hoàn chỉnh. Trong LINQ, một hoạt động truy vấn hoàn chỉnh bao gồm: tạo nguồn dữ liệu, định nghĩa biểu thức truy vấn và thực hiện truy vấn trong câu lệnh foreach.

```
class LINQQueryExpressions
{
    static void Main()
    {
        // Xác định nguồn dữ liệu
        int[] scores = new int[] { 97, 92, 81, 60 };

        // Định nghĩa biểu thức truy vấn
        IEnumerable<int> scoreQuery =
            from score in scores
            where score > 80
            select score;

        // Thực thi truy vấn
        foreach (int i in scoreQuery)
```

```

    {
        Console.Write(i + " ");
    }
}

```



Kết quả:

Output: 97 92 81

1.1. Truy vấn là gì và truy vấn làm gì?

Truy vấn là một tập hợp các chỉ dẫn mô tả dữ liệu cần truy xuất từ một nguồn dữ liệu nhất định, định dạng và cách tổ chức dữ liệu được trả về.

Nhìn chung, dữ liệu nguồn được tổ chức như một chuỗi các phần tử cùng loại. Ví dụ, một bảng cơ sở dữ liệu SQL chứa một chuỗi các dòng. Trong file XML, có một "chuỗi" các phần tử XML (mặc dù các phần tử này được tổ chức theo thứ bậc trong cấu trúc cây). Một tập hợp chứa một chuỗi các đối tượng trong bộ nhớ.

Từ quan điểm của ứng dụng, loại và cấu trúc cụ thể của dữ liệu nguồn ban đầu không quan trọng. Ứng dụng luôn xem dữ liệu nguồn là một tập hợp `IEnumerable<T>` hoặc `IQueryable<T>`. Ví dụ: trong LINQ to XML, dữ liệu nguồn được hiển thị dưới dạng `IEnumerable<XElement>`.

Với chuỗi nguồn này, một truy vấn có thể thực hiện một trong ba điều sau:

- Truy xuất một tập hợp con của các phần tử để tạo ra một chuỗi mới mà không sửa đổi các phần tử riêng biệt. Sau đó, truy vấn có thể sắp xếp hoặc nhóm chuỗi được trả về theo nhiều cách khác nhau, như trong ví dụ sau (giả sử `scores` là mảng 1 chiều, các phần tử có kiểu `int`):

```

IEnumerable<int> highScoresQuery =
    from score in scores
    where score > 80
    orderby score descending
    select score;

```

- Lấy một chuỗi các phần tử như trong ví dụ trên nhưng chuyển đổi chúng thành một kiểu đối tượng mới. Ví dụ: một truy vấn có thể chỉ truy xuất last name từ các bản ghi khách hàng nhất định trong nguồn dữ liệu. Hoặc nó có thể truy xuất toàn bộ bản ghi và sau đó sử dụng nó để tạo một loại đối tượng khác trong bộ nhớ hoặc thậm chí dữ liệu XML trước khi sinh chuỗi kết quả cuối cùng. Ví dụ sau đây cho thấy một phép chiếu từ một số nguyên đến một chuỗi. Chú ý kiểu mới của `highScoresQuery`.

```

IEnumerable<string> highScoresQuery2 =
    from score in scores
    where score > 80
    orderby score descending
    select $"The score is {score}";

```

- Lấy một giá trị đơn lẻ về dữ liệu nguồn, chẳng hạn như:
 - + Số lượng các phần tử phù hợp với một điều kiện nhất định.
 - + Phần tử có giá trị lớn nhất hoặc nhỏ nhất.
 - + Phần tử đầu tiên thỏa mãn điều kiện hoặc tổng các giá trị cụ thể trong một bộ phần tử được chỉ định.

Ví dụ: truy vấn sau đây trả về số điểm lớn hơn 80 từ mảng số nguyên Scores:

```
int highScoreCount =
    (from score in scores
     where score > 80
     select score)
    .Count();
```

Trong ví dụ trên, lưu ý việc sử dụng dấu ngoặc đơn xung quanh biểu thức truy vấn trước khi gọi phương thức Count. Bạn cũng có thể biểu diễn điều này bằng cách sử dụng một biến mới để lưu trữ kết quả cụ thể. Kỹ thuật này dễ đọc hơn vì nó giữ biến lưu trữ truy vấn tách biệt với truy vấn lưu kết quả.

```
IEnumerable<int> highScoresQuery3 =
    from score in scores
    where score > 80
    select score;

int scoreCount = highScoresQuery3.Count();
```

1.2. Biểu thức truy vấn (query expression) là gì?

Một biểu thức truy vấn là một truy vấn được biểu diễn theo cú pháp truy vấn. Nó cũng giống như bất kỳ biểu thức nào và có thể được sử dụng trong bất kỳ ngữ cảnh nào ở đó biểu thức C# là hợp lệ.

Một biểu thức truy vấn bao gồm một tập các mệnh đề được viết theo cú pháp khai báo tương tự như SQL hoặc XQuery. Mỗi mệnh đề lại chứa một hoặc nhiều biểu thức C# và chính các biểu thức này có thể là biểu thức truy vấn hoặc chứa biểu thức truy vấn.

Một biểu thức truy vấn phải bắt đầu bằng mệnh đề from và phải kết thúc bằng mệnh đề select hoặc group. Giữa mệnh đề from đầu tiên và mệnh đề select hoặc group cuối cùng, có thể chứa một hoặc nhiều mệnh đề tùy chọn sau: where, orderby, join, let và thậm chí thêm các mệnh đề from. Bạn cũng có thể sử dụng từ khóa into để cho phép kết quả của mệnh đề join hoặc group trở thành nguồn cho các mệnh đề truy vấn bổ sung trong cùng một biểu thức truy vấn.

1.3. Biến truy vấn (Query variable)

Trong LINQ, biến truy vấn là bất kỳ biến nào lưu trữ truy vấn thay vì kết quả của truy vấn. Cụ thể hơn, một biến truy vấn luôn là một kiểu enumerable, kiểu sẽ tạo ra một chuỗi các phần tử khi nó được lặp trong một câu lệnh foreach hoặc một lời gọi trực tiếp đến phương thức IEnumerator.MoveNext.

Ví dụ sau đây cho thấy một biểu thức truy vấn đơn giản với một nguồn dữ liệu, một mệnh đề lọc dữ liệu, một mệnh đề sắp xếp và không chuyển đổi các phần tử nguồn. Mệnh đề select kết thúc truy vấn.

```
static void Main()
{
    // Nguồn dữ liệu.
    int[] scores = { 90, 71, 82, 93, 75, 82 };

    // Biểu thức truy vấn.
    IEnumerable<int> scoreQuery = //biến truy vấn
        from score in scores //mệnh đề from là bắt buộc
        where score > 80 // mệnh đề where là tùy chọn
        orderby score descending // mệnh đề orderby là tùy chọn
        select score; //biểu thức truy vấn phải kết thúc bằng select|group

    // Thực thi truy vấn để sinh kết quả
    foreach (int testScore in scoreQuery)
    {
        Console.WriteLine(testScore);
    }
}
```



Kết quả:

Output: 93 90 82 82

Trong ví dụ trên, scoreQuery là một biến truy vấn, đôi khi chỉ được gọi là truy vấn. Biến truy vấn không lưu trữ dữ liệu kết quả thực sự, dữ liệu sẽ được sinh ra trong vòng lặp foreach. Và khi câu lệnh foreach thực thi, kết quả truy vấn không được trả về thông qua biến truy vấn scoreQuery. Thay vào đó, chúng được trả về thông qua biến lặp testScore. Biến scoreQuery có thể được lặp lại trong vòng lặp foreach thứ hai. Nó sẽ sinh ra kết quả tương tự miễn là nó và nguồn dữ liệu không bị sửa đổi.

Một biến truy vấn có thể lưu trữ một truy vấn được biểu diễn bằng cú pháp truy vấn hoặc cú pháp phương thức hoặc kết hợp cả hai. Trong các ví dụ sau, cả queryMajorCities và queryMajorCities2 đều là các biến truy vấn:

```
//Cú pháp truy vấn
IEnumerable<City> queryMajorCities =
    from city in cities
    where city.Population > 100000
    select city;

// Cú pháp phương thức
IEnumerable<City> queryMajorCities2 =
    cities.Where(c => c.Population > 100000);
```

Mặt khác, hai ví dụ sau đây cho thấy các biến không phải là biến truy vấn mặc dù mỗi biến được khởi tạo bằng một truy vấn. Chúng không phải là biến truy vấn vì chúng lưu trữ kết quả:

```
int highestScore =
    (from score in scores
     select score)
     .Max();

// hoặc tách biểu thức
IEnumerable<int> scoreQuery =
    from score in scores
    select score;
int highScore = scoreQuery.Max();

// đoạn code sau trả lại cùng kết quả
int highScore = scores.Max();

List<City> largeCitiesList =
    (from country in countries
     from city in country.Cities
     where city.Population > 10000
     select city)
     .ToList();

// hoặc tách biểu thức
IEnumerable<City> largeCitiesQuery =
    from country in countries
    from city in country.Cities
    where city.Population > 10000
    select city;

List<City> largeCitiesList2 = largeCitiesQuery.ToList();
```

Kiểu của các biến truy vấn tường minh và ngầm định

Ta sử dụng biến truy vấn có kiểu tường minh để thể hiện mối quan hệ kiểu giữa biến truy vấn và mệnh đề select. Tuy nhiên, bạn cũng có thể sử dụng từ khóa var để hướng dẫn trình biên dịch suy luận ra kiểu của biến truy vấn (hoặc bất kỳ biến cục bộ nào khác) tại thời điểm biên dịch.

Ví dụ: truy vấn đã được hiển thị trước đó trong chủ đề này cũng có thể được thể hiện bằng cách sử dụng kiểu ngầm định:

```
// Sử dụng var là tùy chọn ở đây và trong tất cả các truy vấn
// queryCities là IEnumerable<City> ngay khi nó được xác định kiểu
var queryCities =
    from city in cities
    where city.Population > 100000
```

```
select city;
```

1.4. Xây dựng biểu thức truy vấn

1.4.1. Bắt đầu biểu thức truy vấn bằng mệnh đề from

Một biểu thức truy vấn phải bắt đầu bằng mệnh đề from. Nó chỉ ra nguồn dữ liệu cùng với một biến phạm vi (range variable). Biến phạm vi đại diện cho từng phần tử liên tiếp trong chuỗi nguồn khi chuỗi nguồn được duyệt qua. Biến phạm vi được xác định kiểu dựa trên kiểu của phần tử trong nguồn dữ liệu.

Trong ví dụ sau, vì countries là một mảng các đối tượng Country, biến phạm vi country cũng được xác định kiểu là Country. Vì biến phạm vi được xác định kiểu, bạn có thể sử dụng dấu chấm (.) để truy cập bất kỳ thành viên có sẵn nào của kiểu đó.

```
IEnumerable<Country> countryAreaQuery =  
    from country in countries  
    where country.Area > 500000 // km2  
    select country;
```

Biến phạm vi nằm trong phạm vi sử dụng cho đến khi truy vấn được thoát bằng dấu chấm phẩy hoặc với mệnh đề *continuation*.

Một biểu thức truy vấn có thể chứa nhiều mệnh đề from. Sử dụng thêm các mệnh đề from khi bản thân mỗi phần tử trong chuỗi nguồn là tập hợp hoặc chứa một tập hợp.

Ví dụ: giả sử rằng bạn có một tập hợp các đối tượng country, mỗi đối tượng chứa một tập hợp các đối tượng City có tên là Cities. Để truy vấn các đối tượng City trong mỗi Country, hãy sử dụng hai mệnh đề from như sau:

```
IEnumerable<City> cityQuery =  
    from country in countries  
    from city in country.Cities  
    where city.Population > 10000  
    select city;
```

1.4.2. Kết thúc biểu thức truy vấn với mệnh đề select hoặc group

Một biểu thức truy vấn phải kết thúc bằng mệnh đề group hoặc mệnh đề select.

Mệnh đề group

Sử dụng mệnh đề group để tạo một chuỗi các nhóm được tổ chức bởi một khóa (key) mà bạn chỉ định. Khóa có thể là bất kỳ loại dữ liệu nào.

Ví dụ: truy vấn sau đây tạo một chuỗi các nhóm có chứa một hoặc nhiều đối tượng Country và khóa của nó là 1 giá trị kiểu char có giá trị là ký tự đầu tiên của thuộc tính Name

```
var queryCountryGroups =  
    from country in countries  
    group country by country.Name[0];
```

Mệnh đề select

Sử dụng mệnh đề select để tạo ra tất cả các loại chuỗi phân tử khác.

Một mệnh đề select đơn giản chỉ tạo ra một chuỗi các đối tượng cùng kiểu như các đối tượng được chứa trong nguồn dữ liệu.

Trong ví dụ sau, nguồn dữ liệu chứa các đối tượng Country. Mệnh đề orderby chỉ sắp xếp các phân tử theo một thứ tự mới và mệnh đề select tạo ra một chuỗi các đối tượng country đã được sắp xếp lại.

```
IEnumerable<Country> sortedQuery =
    from country in countries
    orderby country.Area
    select country;
```

Mệnh đề select có thể được sử dụng để chuyển đổi dữ liệu nguồn thành chuỗi các kiểu mới. Sự chuyển đổi này cũng được đặt tên là một projection (phép chiếu).

Trong ví dụ sau, mệnh đề select chiếu một chuỗi các đối tượng kiểu vô danh, kiểu này chỉ chứa một tập hợp con của các trường trong phân tử gốc.

```
// ở đây var là bắt buộc bởi truy vấn sinh ra kiểu vô danh
var queryNameAndPop =
    from country in countries
    select new { Name = country.Name, Pop = country.Population };
```

1.4.3. Từ khóa into

Bạn có thể sử dụng từ khóa into trong mệnh đề select hoặc group để tạo định danh tạm thời lưu trữ truy vấn. Thực hiện việc này khi bạn phải thực hiện các thao tác truy vấn bổ sung trên một truy vấn sau thao tác nhóm hoặc chọn.

Trong ví dụ sau, các quốc gia được phân nhóm theo dân số trong khoảng 10 triệu. Sau khi các nhóm này được tạo, các mệnh đề bổ sung lọc ra một số nhóm và sau đó sắp xếp các nhóm theo thứ tự tăng dần. Để thực hiện các hoạt động bổ sung đó, cần có phần tiếp theo được biểu diễn bởi countryGroup.

```
// percentileQuery là một IEnumerable<IGrouping<int, Country>>
var percentileQuery =
    from country in countries
    let percentile = (int) country.Population / 10000000
    group country by percentile into countryGroup
    where countryGroup.Key >= 20
    orderby countryGroup.Key
    select countryGroup;
// Nhóm là một IGrouping<int, Country>
foreach (var grouping in percentileQuery)
{
    Console.WriteLine(grouping.Key);
    foreach (var country in grouping)
        Console.WriteLine(country.Name + ":" + country.Population);
}
```


1.4.4. Lọc, sắp xếp và kết nối

Giữa mệnh đề bắt đầu from và mệnh đề kết thúc select hoặc group, tất cả các mệnh đề khác (where, join, orderby, from, let) là tùy chọn. Bất kỳ mệnh đề tùy chọn nào cũng có thể được sử dụng 0 hoặc nhiều lần trong thân truy vấn.

Mệnh đề where

Sử dụng mệnh đề where để lọc các phần tử từ dữ liệu nguồn dựa trên một hoặc nhiều điều kiện.

Ví dụ mệnh đề where có 2 điều kiện

```
IEnumerable<City> queryCityPop =  
    from city in cities  
    where city.Population < 200000 && city.Population > 100000  
    select city;
```

Mệnh đề orderby

Sử dụng mệnh đề orderby để sắp xếp các kết quả theo thứ tự tăng dần hoặc giảm dần. Bạn cũng có thể chỉ định các thứ tự sắp xếp tiếp theo. Ví dụ sau đây thực hiện sắp xếp trên các đối tượng country bằng cách sử dụng thuộc tính Area. Sau đó, nó thực hiện sắp xếp bằng cách sử dụng thuộc tính Population.

```
IEnumerable<Country> querySortedCountries =  
    from country in countries  
    orderby country.Area, country.Population descending  
    select country;
```

Từ khóa ascending là tùy chọn; nó là thứ tự sắp xếp mặc định

Mệnh đề join

Sử dụng mệnh đề join để liên kết và/hoặc kết hợp các phần tử từ một nguồn dữ liệu với các phần tử từ một nguồn dữ liệu khác dựa trên so sánh bằng giữa các khóa được chỉ định trong mỗi phần tử. Trong LINQ, phép nối được thực hiện trên chuỗi các đối tượng mà các phần tử của chúng có kiểu khác nhau. Sau khi bạn đã kết nối hai chuỗi, bạn phải sử dụng câu lệnh select hoặc group để chỉ định phần tử nào sẽ lưu trong chuỗi kết quả. Bạn cũng có thể sử dụng một kiểu vô danh để kết hợp các thuộc tính từ mỗi bộ phần tử được liên kết thành một loại mới cho chuỗi kết quả.

Ví dụ sau đây liên kết các đối tượng prod có thuộc tính Category khớp với một trong các category trong mảng chuỗi categories. Các product có Category không khớp với bất kỳ chuỗi nào trong categories được loại ra. Câu lệnh select chiếu một kiểu mới có các thuộc tính được lấy từ cả cat và prod.

```
var categoryQuery =  
    from cat in categories  
    join prod in products on cat equals prod.Category  
    select new { Category = cat, Name = prod.Name };
```

Bạn cũng có thể thực hiện kết nối nhóm bằng cách lưu trữ kết quả của hoạt động kết nối vào một biến tạm bằng cách sử dụng từ khóa `into`.

1.4.5. Mệnh đề `let`

Sử dụng mệnh đề `let` để lưu trữ kết quả của một biểu thức, như một lời gọi phương thức, trong một biến phạm vi mới.

Trong ví dụ sau, biến `firstName` lưu trữ phần tử đầu tiên của mảng chuỗi được trả về bởi `Split`.

```
string[] names = { "Svetlana Omelchenko", "Claire O'Donnell", "Sven Mortensen",
"Cesar Garcia" };
IEnumerable<string> queryFirstNames =
    from name in names
    let firstName = name.Split(' ')[0]
    select firstName;

foreach (string s in queryFirstNames)
    Console.WriteLine(s + " ");
```

 Kết quả:

Output: Svetlana Claire Sven Cesar

1.5. Cú pháp truy vấn và cú pháp phương thức

Có hai cách để viết truy vấn LINQ là cú pháp truy vấn (query syntax) và cú pháp phương thức (method syntax).

Cú pháp truy vấn: cú pháp tương tự như truy vấn SQL, có các mệnh đề `select`, `from`, `where`, `join`. . . truy vấn được viết trong code C# hoặc VB.NET theo cú pháp sau:

```
from <biến-phạm-vi> in <tập-hợp-<IEnumerable<T>-hoặc-IQueryable<T>>
<Các-mệnh-đề-truy-vấn-chuẩn> <biểu-thức-lambda>
<toán-tử select hoặc groupBy > <dữ-liệu-kết-quả>
```

Cú pháp phương thức: cú pháp phương thức LINQ sử dụng các phương thức mở rộng có trong 2 lớp `Enumerable` hoặc `Queryable`

Ví dụ sau minh họa một truy vấn sử dụng cú pháp phương thức để trả về những sản phẩm có giá = 400

```
var ketqua = products.Where(product => product.Price == 400);
```

Phương thức mở rộng
Biểu thức lambda

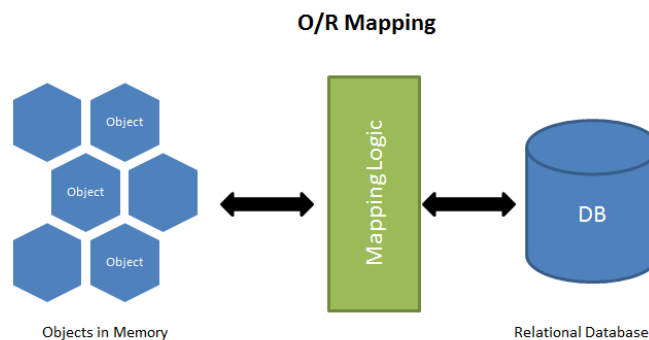
Cú pháp pha trộn: thực tế là một số phương thức LINQ không được hỗ trợ cú pháp truy vấn, khi đó ta có thể sử dụng cách viết pha trộn của cả hai loại cú pháp

2. ORM và EF

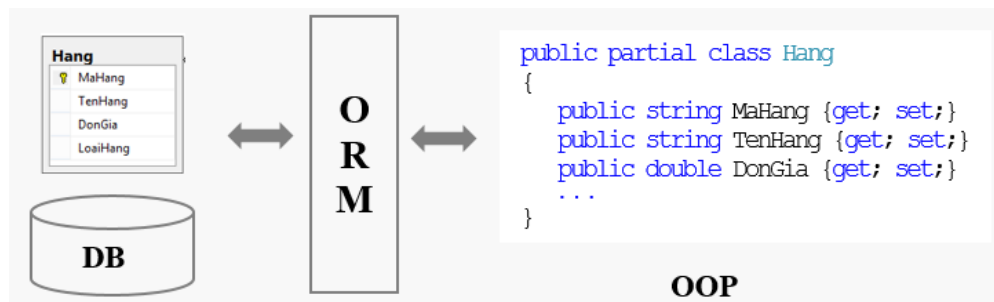
2.1. ORM (Object Relational Mapping)

Khi chúng ta làm việc với một hệ thống hướng đối tượng, có một sự không khớp giữa mô hình đối tượng và cơ sở dữ liệu quan hệ. RDBMS thể hiện dữ liệu trong định dạng bảng, trong khi các ngôn ngữ hướng đối tượng đại diện cho dữ liệu là một đồ thị kết nối của các đối tượng. (ORM) ra đời nhằm giải quyết bài toán về ánh xạ giữa table-to-object và object-to-table.

ORM là một kỹ thuật thực hiện ánh xạ CSDL sang các đối tượng trong các ngôn ngữ lập trình hướng đối tượng



Trong ORM các bảng tương ứng các class, mối ràng buộc giữa các bảng tương ứng quan hệ giữa các class



ORM Là khái niệm phổ biến, được cài đặt trong tất cả các ngôn ngữ hiện đại như: c#, java, php, node.js, ...

Tại sao nên cài đặt ORM để truy xuất dữ liệu?

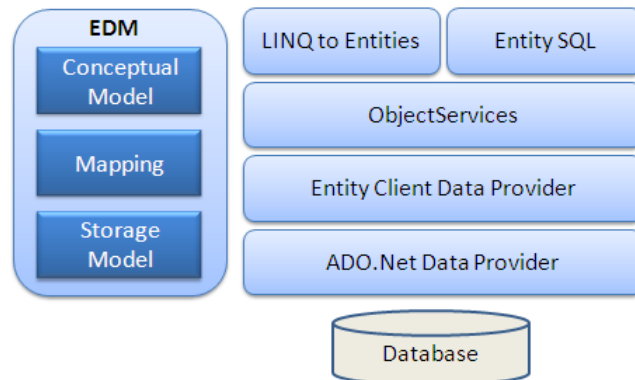
- **Portable – tính năng động:** ORM được sử dụng để viết cấu trúc một lần và lớp ORM sẽ xử lý câu lệnh cuối cùng phù hợp với DBMS được cấu hình. Đây là một lợi thế tuyệt vời khi thao tác truy xuất dữ liệu đơn giản như giới hạn được thêm vào dưới dạng 'limit 0,100' ở cuối câu lệnh Select trong MySQL, trong khi đó với cách truy xuất thông thường phải viết là 'Select Top 100 From Table' trong MS SQL.
- **Nesting Of Data – truy xuất lồng dữ liệu:** trong trường hợp database có nhiều bảng và các bảng này liên hệ phức tạp về dữ liệu thì ORM sẽ tự động lấy dữ liệu một cách đơn giản (ở đây không bàn về vấn đề tối ưu truy xuất)

- Single Language – không cần biết SQL: thật vậy với nguyên lý thiết kế là ánh xạ toàn bộ dữ liệu lấy được từ DBMS sang bộ nhớ nên việc thao tác truy xuất bây giờ chỉ phụ thuộc vào ngôn ngữ lập trình bạn đang sử dụng, bạn chẳng cần quan tâm phía đằng sau của ORM sẽ làm gì sinh ra mã SQL như thế nào khi truy xuất SQL, và kết quả là chúng ta chỉ cần thuần thục ngôn ngữ lập trình đang dùng.
- Adding is like modifying – thêm sửa dữ liệu là như nhau: đối với ORM, nó không phân biệt giữa thêm mới và cập nhật mọi tác vụ có liên quan đến sửa đổi hay chèn dữ liệu đều được xem là định nghĩa thêm mới, hai tác vụ này được xem như là một

2.2. EF – ENTITY FRAMEWORK

- EF là cách thực hiện của ORM trong .NET. Đây là một cải tiến của ADO.NET, cung cấp cho các nhà phát triển một cơ chế tự động để truy cập và lưu trữ dữ liệu trong cơ sở dữ liệu.
- Các tính năng:
 - + Cung cấp cơ chế để truy cập và lưu trữ dữ liệu vào CSDL
 - + Cung cấp các dịch vụ như theo dõi sự thay đổi của dữ liệu, dịch truy vấn . . .
 - + Sử dụng LINQ to Entities để truy vấn, thêm, sửa, xóa dữ liệu
 - + Tăng khả năng bảo trì và mở rộng cho ứng dụng

Kiến trúc của EF



- EDM (Entity Data Model - mô hình dữ liệu thực thể): EDM chứa 3 thành phần chính là - Conceptual model, Mapping và Storage model.
- Conceptual model - mô hình khái niệm: là mô hình các class và mối quan hệ giữa chúng. Mô hình này độc lập với thiết kế bảng trong csdl
- Storage Model – mô hình lưu trữ: là mô hình thiết kế csdl, nó bao gồm tables, views, stored procedures, mối quan hệ giữa chúng và các khóa
- Mapping – mô hình ánh xạ: chứa các thông tin về cách mô hình khái niệm được ánh xạ đến mô hình lưu trữ
- LINQ to Entities: là một ngôn ngữ truy vấn được sử dụng để truy vấn trên mô hình đối tượng. Nó trả lại các thực thể đã được định nghĩa trong mô hình khái niệm.

- Entity SQL: cũng là một ngôn ngữ truy vấn giống như LINQ to Entities. Tuy nhiên nó phức tạp hơn LINQ to Entity một chút và người phát triển cần học về nó riêng
- Object Service: tự động sinh ra các class tương ứng với mô hình dữ liệu
- Entity Client Data Provider: trách nhiệm chính của tầng này là chuyển đổi các truy vấn của LINQ to Entities hoặc Entity SQL thành câu lệnh truy vấn SQL. Nó giao tiếp với CSDL thông qua ADO.Net để gửi và nhận dữ liệu về từ csdl
- ADO.Net Data Provider: đây là tầng giao tiếp với CSDL sử dụng ADO.NET tiêu chuẩn

3. Entity Framework Core (EF Core)

EF Core là phiên bản mới của EF. Nó là phiên bản nhẹ, có thể mở rộng, mã nguồn mở và đa nền tảng của EF.

EF Core làm việc như một ánh xạ ORM, nó cho phép :

- + Các nhà phát triển .NET làm việc với cơ sở dữ liệu sử dụng các đối tượng .NET
- + Loại bỏ hầu hết các lệnh truy cập dữ liệu thường phải viết.

Có thể truy cập nhiều csdl khác nhau như SQL Server, Sqlite, MySQL ... thông qua các thư viện plug-in được gọi là [Database Providers](#)

3.1.1. Model (mô hình dữ liệu)

Với EF Core, việc truy cập dữ liệu được thực hiện sử dụng một **Model**. Một model được tạo thành từ các lớp thực thể (**Entities**) và một đối tượng context đại diện cho một phiên làm việc với cơ sở dữ liệu (**DbContext**). Đối tượng context cho phép truy vấn và lưu dữ liệu.

EF hỗ trợ các cách tiếp cận phát triển model sau:

- Tạo một model từ cơ sở dữ liệu hiện có (Database First)
- Viết code model để tạo cơ sở dữ liệu (Code First)
- Ngay khi model được tạo, sử dụng EF Migrations để tạo cơ sở dữ liệu từ mô hình

3.1.2. Truy vấn dữ liệu

Các instance của các lớp thực thể được truy xuất từ cơ sở dữ liệu bằng cách sử dụng LINQ

```
using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}
```

3.1.3. Cập nhật dữ liệu

dữ liệu được tạo, xóa và sửa đổi trong cơ sở dữ liệu bằng cách sử dụng các instances của các lớp thực thể.

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

4. Truy vấn dữ liệu sử dụng EF Core

Theo cách tiếp cận Database First, EF Core API tạo các lớp thực thể và Context dựa trên cơ sở dữ liệu hiện có bằng cách sử dụng các lệnh của EF Core

Các bước để truy vấn dữ liệu sử dụng EF Core

B1. Thêm các thư viện gồm EF Core và một số package khác tùy mục đích sử dụng

B2. Tạo model

B3. Truy vấn dữ liệu sử dụng LINQ

4.1. Thêm các package cần thiết

EF Core không phải là một thành phần của .NET tiêu chuẩn. Ta cần cài đặt các NuGet package cho hai thành phần sau đây để sử dụng EF Core trong ứng dụng:

1. Data Provider của EF Core
2. Công cụ EF Core

4.1.1. Cài đặt Data Provider

EF Core cho phép truy cập cơ sở dữ liệu qua Data Provider. Data Provider là tập hợp các lớp cho phép ta giao tiếp hiệu quả với cơ sở dữ liệu. EF Core có các Data Provider cho các cơ sở dữ liệu khác nhau. Các Data Provider này có sẵn dưới dạng các gói NuGet., như trong bảng sau:

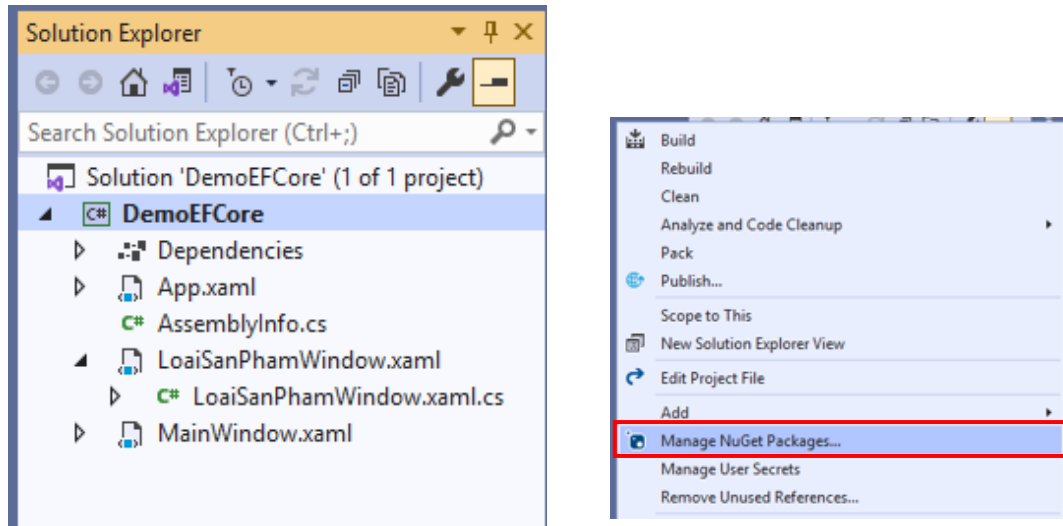
Cơ sở dữ liệu	NuGet packate
Microsoft SQL Server 2012 trở lên	Microsoft.EntityFrameworkCore.SqlServer
SQLite 3.7 trở lên	Microsoft.EntityFrameworkCore.Sqlite
MySQL	MySql.EntityFrameworkCore
Oracle DB 11.2	Oracle.EntityFrameworkCore
Azure Cosmos DB SQL API	Microsoft.EntityFrameworkCore.Cosmos

Xem danh sách đầy đủ các EF Core Data Provider hiện có theo link sau:

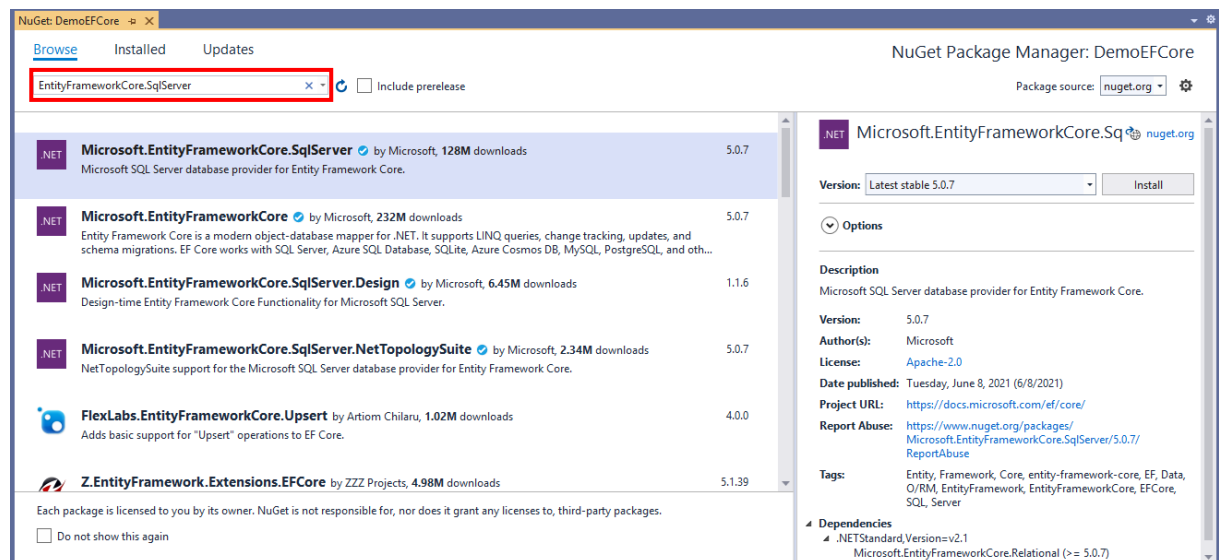
<https://docs.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli>

Ta có thể cài đặt NuGet package theo hai cách sau:

Cách 1: Để cài đặt gói NuGet của DataProvider, ta nhấp phải chuột vào project trong cửa sổ Solution Explorer và chọn → **Manage NuGet Packages for Solution . . .** (hoặc chọn trên menu Tools → NuGet Package Manager → Manage NuGet Packages for Solution . . .)

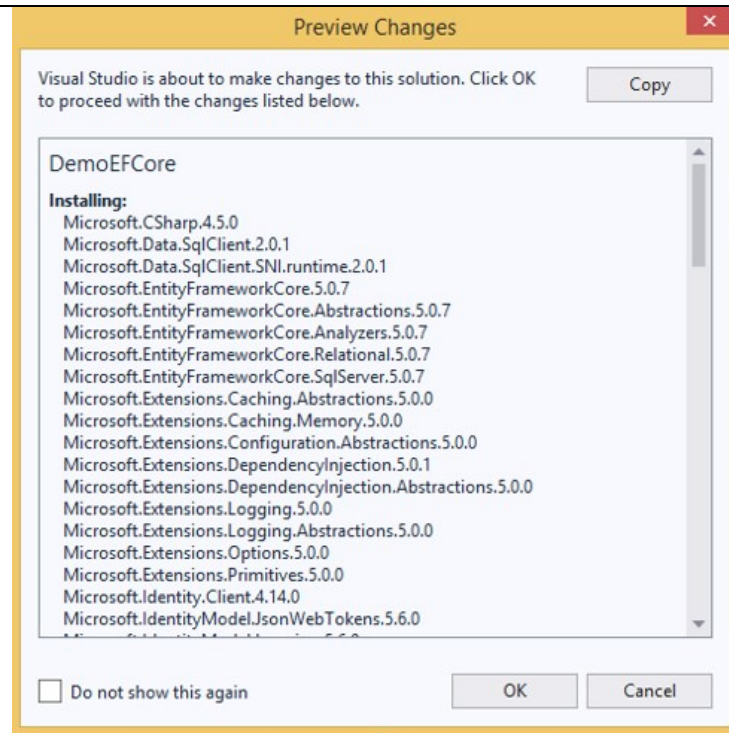


Giao diện trình quản lý gói NuGet được mở. Trong cửa sổ NuGet –Solution → chọn tab Browse và tìm kiếm gói muốn cài đặt như hình sau:

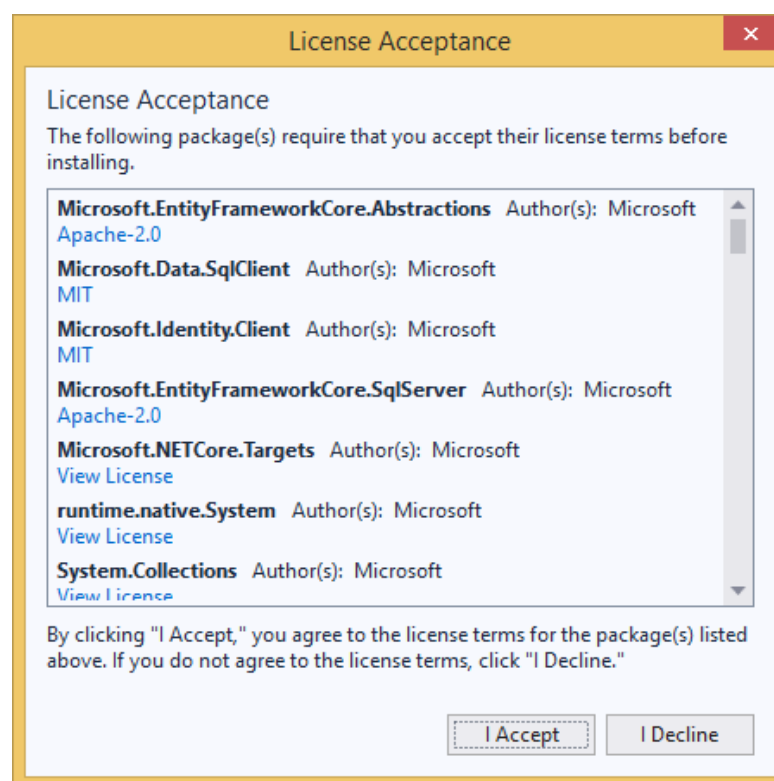


Chọn DataProvider muốn cài. Trong trường hợp này ta muốn truy cập cơ sở dữ liệu SQL Server vì vậy ta cần cài đặt **Microsoft.EntityFrameworkCore.SqlServer**, đảm bảo là nó có logo .NET và tác giả là Microsoft → nhấn Install để cài đặt.

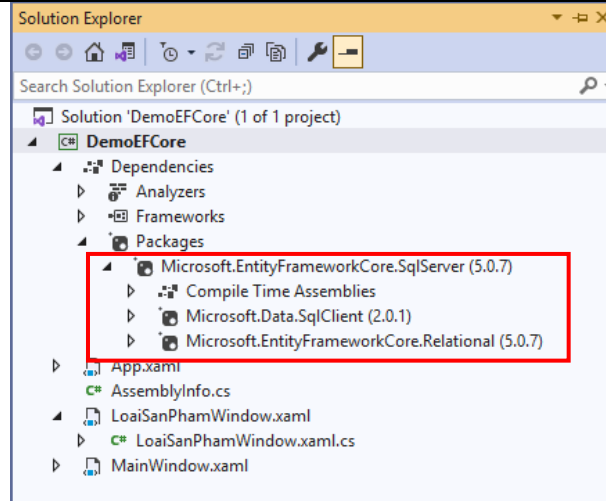
Cửa sổ xem trước hiển thị danh sách các gói sẽ được cài đặt trong ứng dụng, xem lại các thay đổi và nhấn OK



Cuối cùng chấp nhận các điều khoản cấp phép liên quan đến các gói được cài đặt



Sau đó kiểm tra xem gói **Microsoft.EntityFrameworkCore.SqlServer** đã được cài đặt trong Dependencies → NuGet như sau



Cách 2: Cài đặt package bằng Package Manager Console

- Chọn menu Tools → NuGet Package Manager → Package Manager Console
- Nhập lệnh theo cú pháp sau vào cửa sổ Package Manager Console

PM> Install-Package tên-package –Version tên-phiên-bản

+ Nếu không có tùy chọn –Version, mặc định cài phiên bản mới nhất

Ví dụ:

PM> Install-Package Microsoft.EntityFrameworkCore.SqlServer –Version 5.0.6

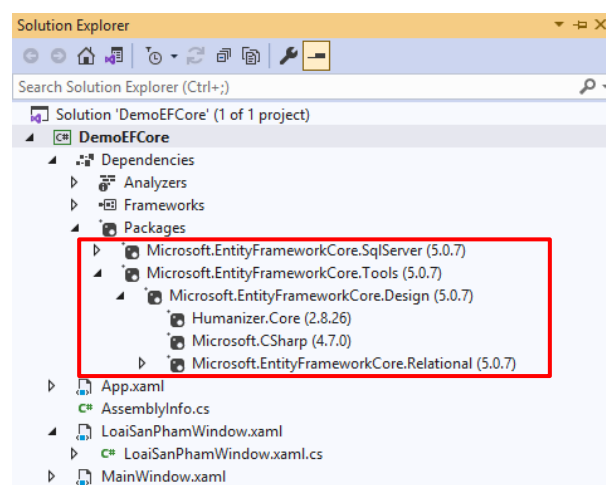
4.1.2. Cài đặt EF Core Tool

Để thực thi các lệnh EF Core ta cần cài đặt NuGet package

Microsoft.EntityFrameworkCore.Tools, điều này giúp ta thực hiện các nhiệm vụ liên quan đến EF Core trong dự án tại thời điểm thiết kế như scaffolding ...

Cài đặt EFCore Tool cũng thực hiện tương tự như cài đặt DataProvider.

Kiểm tra xem gói **Microsoft.EntityFrameworkCore.Tools** đã được cài đặt thành công trong Dependencies → NuGet:

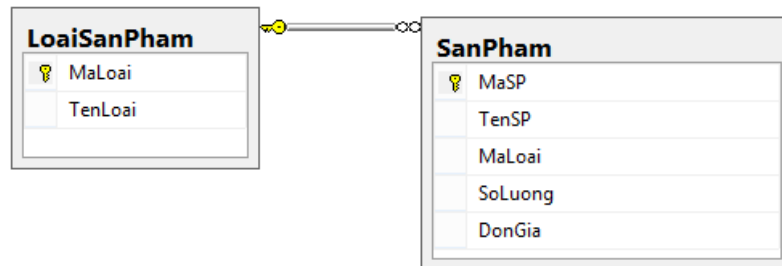


4.2. Tạo model

Trong phần này ta sẽ sử dụng công cụ để tạo các lớp biểu diễn mô hình dữ liệu cho một cơ sở dữ liệu có sẵn, sử dụng kỹ thuật đảo ngược.

4.2.1. Chuẩn bị cơ sở dữ liệu

Tạo cơ sở dữ liệu **QLBanHang** gồm 2 bảng **LoaiSanPham** và **SanPham** có lược đồ cơ sở dữ liệu như sau:



Thiết kế của 2 bảng:

NHUNGNT.QLBanH...dbo.LoaiSanPham X			
	Column Name	Data Type	Allow Nulls
	MaLoai	char(3)	<input type="checkbox"/>
	TenLoai	nvarchar(50)	<input type="checkbox"/>

NHUNGNT.QLBanHang - dbo.SanPham X			
	Column Name	Data Type	Allow Nulls
	MaSP	char(4)	<input type="checkbox"/>
	TenSP	nvarchar(50)	<input type="checkbox"/>
	MaLoai	char(3)	<input checked="" type="checkbox"/>
	SoLuong	int	<input checked="" type="checkbox"/>
	DonGia	int	<input checked="" type="checkbox"/>

4.2.2. Tạo model

EF Core không hỗ trợ thiết kế trực quan cho mô hình cơ sở dữ liệu, tạo các lớp thực thể và lớp Context như các phiên bản trước. Thay vào đó ta sử dụng lệnh **Scaffold-DbContext**. Lệnh này tạo các lớp thực thể và Context (bằng cách dẫn xuất từ lớp DbContext) dựa trên lược đồ của cơ sở dữ liệu hiện có.

Trong cửa sổ Package Manager Console thực hiện lệnh Scaffold-DbContext như sau:

Scaffold-DbContext “chuỗi-kết-nối” tên-DataProvider **-OutputDir** tên-thư-mục

Ví dụ:

```
Scaffold-DbContext "Data Source=MyPC;Initial Catalog=QLBanHang;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

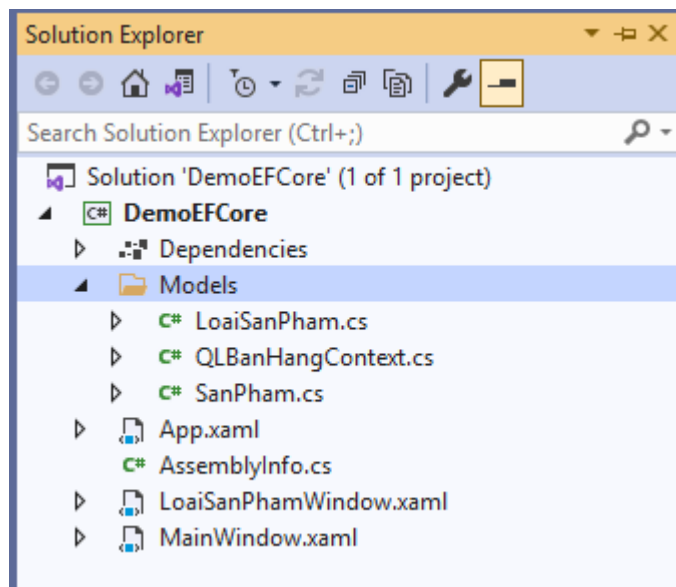
- Tham số chuỗi kết nối gồm ba phần: máy chủ cơ sở dữ liệu, tên cơ sở dữ liệu và thông tin bảo mật.

Ví dụ, trong chuỗi kết nối:

"Data Source=PC;Initial Catalog=QLBanHang;Integrated Security=True"

- + Máy chủ cơ sở dữ liệu là : MyPC
- + Tên cơ sở dữ liệu là: QLBanHang
- + Kết nối với SQL Server sử dụng xác thực Windows Authentication
- Tham số thứ hai là tên DataProvider, vì ta sử dụng cơ sở dữ liệu SQL Server nên trong ví dụ tên DataProvider là Microsoft.EntityFrameworkCore.SqlServer
- Tham số thứ ba là tên thư mục, nơi mà ta muốn lưu tất cả các lớp thực thể, trong ví dụ là thư mục Models của project

Lệnh Scaffold-DbContext tạo các lớp thực thể cho mỗi bảng trong cơ sở dữ liệu QLBanHang và lớp Context với cấu hình cho tất cả các thực thể trong thư mục Models



Sau đây là lớp thực thể LoaiSanPham được tạo từ bảng LoaiSanPham và lớp thực thể SanPham được tạo từ bảng SanPham

```
using System;
using System.Collections.Generic;

#nullable disable

namespace DemoEFCore.Models
{
    public partial class LoaiSanPham
    {
        public LoaiSanPham()
        {
            SanPhams = new HashSet<SanPham>();
        }
    }
}
```

```

    public string MaLoai { get; set; }
    public string TenLoai { get; set; }

    public virtual ICollection<SanPham> SanPhams { get; set; }
  }
}

```

```

using System;
using System.Collections.Generic;

#nullable disable

namespace DemoEFCore.Models
{
    public partial class SanPham
    {
        public string MaSp { get; set; }
        public string TenSp { get; set; }
        public string MaLoai { get; set; }
        public int? SoLuong { get; set; }
        public int? DonGia { get; set; }

        public virtual LoaiSanPham MaLoaiNavigation { get; set; }
    }
}

```

Lớp thực thể (Entity class):

Các lớp thực thể ánh xạ vào các bảng bên trong một cơ sở dữ liệu. Các thuộc tính của lớp ánh xạ vào các cột trong bảng. Mỗi thể hiện của một lớp thực thể biểu diễn 1 dòng trong bảng.

Trong ví dụ trên, lớp thực thể SanPham ánh xạ vào bảng SanPham trong cơ sở dữ liệu. Các thuộc tính như MaSP, TenSP ... ánh xạ các cột MaSP, TenSP... trong bảng sản phẩm. Mỗi một thể hiện của lớp SanPham biểu diễn 1 dòng trong bảng SanPham. Lớp thực thể LoaiSanPham ánh xạ vào bảng LoaiSanPham, mỗi một thể hiện của lớp LoaiSanPham biểu diễn 1 dòng trong bảng LoaiSanPham

Quan hệ giữa các lớp thực thể:

Được tự động tạo ra dựa trên các mối quan hệ primary key/foreign key trong CSDL, khi đó các thuộc tính tương ứng sẽ được thêm vào các lớp thực thể.

Ví dụ: quan hệ giữa lớp thực thể LoaiSanPham và SanPham được tạo dựa trên quan hệ 1–nhiều giữa bảng LoaiSanPham và bảng SanPham. Mỗi quan hệ này làm lớp thực thể SanPham có thêm thuộc tính “LoaiSanPham”, dùng để truy cập vào thực thể LoaiSanPham của một SanPham. Lớp LoaiSanPham cũng có thêm thuộc tính “SanPhams”, đây là một tập hợp cho phép ta lấy ra tất cả các SanPham có trong LoaiSanPham đó.

Sau đây là lớp `QLBanHangContext` được sử dụng để lưu hoặc truy xuất dữ liệu

```
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata;

#nullable disable

namespace DemoEFCore.Models
{
    public partial class QLBanHangContext : DbContext
    {
        public QLBanHangContext()
        {
        }

        public QLBanHangContext(DbContextOptions<QLBanHangContext>
options)
        : base(options)
        {
        }

        public virtual DbSet<LoaiSanPham> LoaiSanPhams { get; set; }
        public virtual DbSet<SanPham> SanPhams { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {
            if (!optionsBuilder.IsConfigured)
            {
                #warning To protect potentially sensitive information in your connection
string, you should move it out of source code. You can avoid scaffolding
the connection string by using the Name= syntax to read it from
configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For
more guidance on storing connection strings, see
http://go.microsoft.com/fwlink/?LinkId=723263.
                optionsBuilder.UseSqlServer("Data Source=NhungNT;Initial
Catalog=QLBanHang;Integrated Security=True");
            }
        }

        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            modelBuilder.HasAnnotation("Relational:Collation",
"SQL_Latin1_General_CP1_CI_AS");

            modelBuilder.Entity<LoaiSanPham>(entity =>
            {
                entity.HasKey(e => e.MaLoai)
                    .HasName("PK__LoaiSanP__730A575920049E9B");
            });
        }
    }
}
```

```

        entity.ToTable("LoaiSanPham");

        entity.Property(e => e.MaLoai)
            .HasMaxLength(3)
            .IsUnicode(false)
            .IsFixedLength(true);

        entity.Property(e => e.TenLoai)
            .IsRequired()
            .HasMaxLength(50);
    });

    modelBuilder.Entity<SanPham>(entity =>
    {
        entity.HasKey(e => e.MaSp)
            .HasName("PK__SanPham__2725081CACF60F4F");

        entity.ToTable("SanPham");

        entity.Property(e => e.MaSp)
            .HasMaxLength(4)
            .IsUnicode(false)
            .HasColumnName("MaSP")
            .IsFixedLength(true);

        entity.Property(e => e.MaLoai)
            .HasMaxLength(3)
            .IsUnicode(false)
            .IsFixedLength(true);

        entity.Property(e => e.TenSp)
            .IsRequired()
            .HasMaxLength(50)
            .HasColumnName("TenSP");

        entity.HasOne(d => d.MaLoaiNavigation)
            .WithMany(p => p.SanPhams)
            .HasForeignKey(d => d.MaLoai)
            .HasConstraintName("FK__SanPham__MaLoai__1273C1CD");
    });

    OnModelCreatingPartial(modelBuilder);
}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}
}

```

Lớp DbContext là một phần không thể thiếu của Entity Framework. Một thể hiện của DbContext đại diện cho một phiên làm việc với cơ sở dữ liệu, có thể được sử dụng để truy vấn và lưu các thể hiện của các thực thể của bạn vào cơ sở dữ liệu

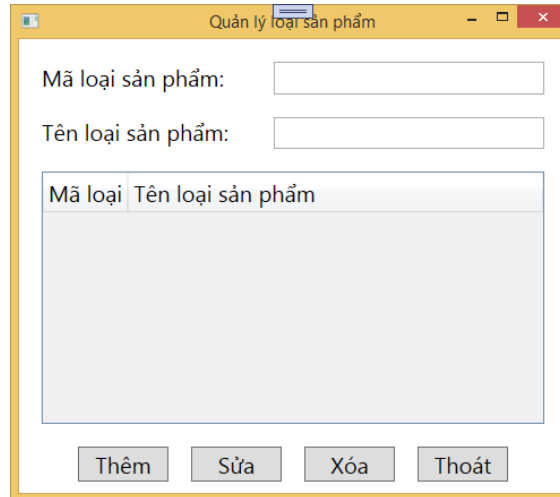
Để sử dụng DbContext trong ứng dụng, chúng ta cần tạo lớp kế thừa từ lớp DbContext, còn được gọi là lớp Context.

Lớp Context này có các thuộc tính **Dbset<TEntity>** cho mỗi thực thể trong mô hình.

4.2.3. Truy vấn dữ liệu sử dụng LINQ

Thiết kế giao diện:

Tạo window để xem và cập nhật dữ liệu bảng LoaiSanPham như sau:



XAML để tạo giao diện:

```
<Window x:Class="DemoEFCore.LoaiSanPhamWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:local="clr-namespace:DemoEFCore"
  mc:Ignorable="d"
  Title="Quản lý loại sản phẩm" Height="440" Width="500"
  FontSize="20">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="auto"/>
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="auto"/>
      <RowDefinition Height="auto"/>
      <RowDefinition />
      <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>

    <TextBlock Grid.Row="0" Grid.Column="0" Margin="20,20,20,10">Mã
loại sản phẩm:</TextBlock>
    <TextBox x:Name="txtMa"
      Grid.Row="0" Grid.Column="1"
      Margin="20,20,20,10"></TextBox>
```

```

    <TextBlock Grid.Row="1" Grid.Column="0" Margin="20,10,20,10">Tên
    loại sản phẩm:</TextBlock>
    <TextBox x:Name="txtTen"
              Grid.Row="1" Grid.Column="1"
    Margin="20,10,20,10"></TextBox>

    <Grid Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="2"
    Margin="20,10,20,10">
        <DataGrid x:Name="dgvLoaiSanPham">
        </DataGrid>
    </Grid>

    <WrapPanel Grid.Row="3" Grid.Column="0" Grid.ColumnSpan="2"
    HorizontalAlignment="Center">
        <Button x:Name="btnThem" Width="80" Margin="10">Thêm</Button>
        <Button x:Name="btnSua" Width="80" Margin="10">Sửa</Button>
        <Button x:Name="btnXoa" Width="80" Margin="10">Xóa</Button>
        <Button x:Name="btnThoat" Width="80"
    Margin="10">Thoát</Button>
    </WrapPanel>
  </Grid>
</Window>

```

Truy vấn dữ liệu:

Sau khi đã định nghĩa mô hình dữ liệu, chúng ta có thể dễ dàng truy vấn và lấy dữ liệu từ cơ sở dữ liệu. EF Core cho phép bạn làm điều này bằng cách viết các câu truy vấn dùng cú pháp LINQ với lớp Context đã được tạo.

Ví dụ, để lấy và duyệt qua một tập các đối tượng LoaiSanPham, ta viết:

```

//Khởi tạo thể hiện của lớp Context
QLBanHangContext db = new QLBanHangContext();
. . .
//truy vấn dữ liệu sử dụng lớp Context
var query = from lsp in db.LoaiSanPhams
             select lsp;
//Hiển thị dữ liệu lên data grid
dgvLoaiSanPham.ItemsSource = query.ToList();

```

Để lấy về một tập các đối tượng thỏa mãn điều kiện, sử dụng mệnh đề where trong truy vấn.

Ví dụ, lấy các loại sản phẩm có tên loại bắt đầu bằng Đ

```

. . .
var query = from lsp in db.LoaiSanPhams
             where lsp.TenLoai.StartsWith("Đ")
             select new {
                 lsp.MaLoai,
                 lsp.TenLoai
             };

```

Trong câu truy vấn trên, ta đã dùng mệnh đề where trong cú pháp LINQ để chỉ trả về các loại sản phẩm mà tên bắt đầu bằng ký tự Đ. Ta có thể tùy biến câu lệnh và nắm bắt

ưu điểm của mối quan hệ giữa các thực thể đã được tạo ra khi mô hình hóa các lớp làm cho câu lệnh phong phú và tự nhiên hơn.

Ví dụ, lấy ra loại sản phẩm có từ 2 sản phẩm trở lên

```
var query = from lsp in db.LoaiSanPhams
             where lsp.SanPhams.Count >= 2
             select lsp;
```

Chú ý cách chúng ta đã dùng tập hợp SanPhams mà EF đã tạo trên mỗi lớp LoaiSanPham nhờ vào mối quan hệ một – nhiều đã được mô hình hóa. Tập hợp SanPhams chứa các sản phẩm của loại sản phẩm đang xét đến.

5. Cập nhật cơ sở dữ liệu sử dụng EF core

Như trong các ví dụ trước, chúng ta dễ dàng dùng các truy vấn LINQ để lấy dữ liệu từ cơ sở dữ liệu bằng cách dùng lớp Context.

Ví dụ: ta viết biểu thức LINQ như sau để lấy về các đối tượng loại sản phẩm có nhiều hơn 2 sản phẩm

```
var loaiSanPhamQuery = from lsp in db.LoaiSanPhams
                        where lsp.SanPhams.Count>=2
                        select lsp;
```

Khi ta thực hiện truy vấn và lấy về các đối tượng như đối tượng loại sản phẩm trong ví dụ trên, LINQ sẽ lưu lại vết của tất cả các thay đổi hay cập nhật mà ta thực hiện trên các đối tượng đó (gọi là change tracking). Chúng ta có thể thực hiện bao nhiêu câu truy vấn và thay đổi mà chúng ta muốn bằng cách dùng LINQ, và tất cả các thay đổi đó sẽ được lưu vết lại

Sau khi đã cập nhật các đối tượng lấy từ LINQ, gọi phương thức "SaveChanges()" của lớp Context để lưu các thay đổi lên CSDL.

5.1. Thêm mới dữ liệu

Thực hiện các bước sau để thêm bản ghi mới vào CSDL:

- Tạo (các) đối tượng thuộc lớp thực thể muốn thêm
- Gán giá trị cho thuộc tính của (các) đối tượng
- Thêm đối tượng vào tập hợp đối tượng tương ứng sử dụng phương thức **Add()** hoặc **AddRange()** của lớp Context
- Thực thi phương thức **SaveChanges()** của lớp Context để cập nhật các thay đổi vào csdl

Ví dụ: thêm một loại sản phẩm mới vào csdl

```
//1. tạo đối tượng loại sản phẩm có thuộc tính do user nhập
LoaiSanPham lspMoi = new LoaiSanPham();
lspMoi.MaLoai = txtMa.Text;
lspMoi.TenLoai = txtTen.Text;
```

```
//2. Thêm vào tập hợp LoaiSanPhams
db.LoaiSanPhams.Add(lspMoi);
//3. Lưu thay đổi vào csdl
db.SaveChanges();
```

5.2. Sửa dữ liệu

Thực hiện các bước sau để sửa dữ liệu:

- Sử dụng LINQ lấy ra các đối tượng muốn sửa dữ liệu
- Sửa thông tin của các đối tượng
- Thực thi phương thức **SaveChanges()** của lớp Context để cập nhật các thay đổi vào csdl

Ví dụ: sửa thông tin của loại sản phẩm có mã trong text box txtMa

```
//lấy ra sản phẩm muốn sửa
var spSua = db.LoaiSanPhams.SingleOrDefault(
    lsp => lsp.MaLoai == txtMa.Text);
//cập nhật thông tin của sản phẩm
spSua.TenLoai = txtTen.Text;
//Lưu thay đổi vào csdl
db.SaveChanges();
```

5.3. Xóa dữ liệu

Thực hiện các bước sau để xóa dữ liệu:

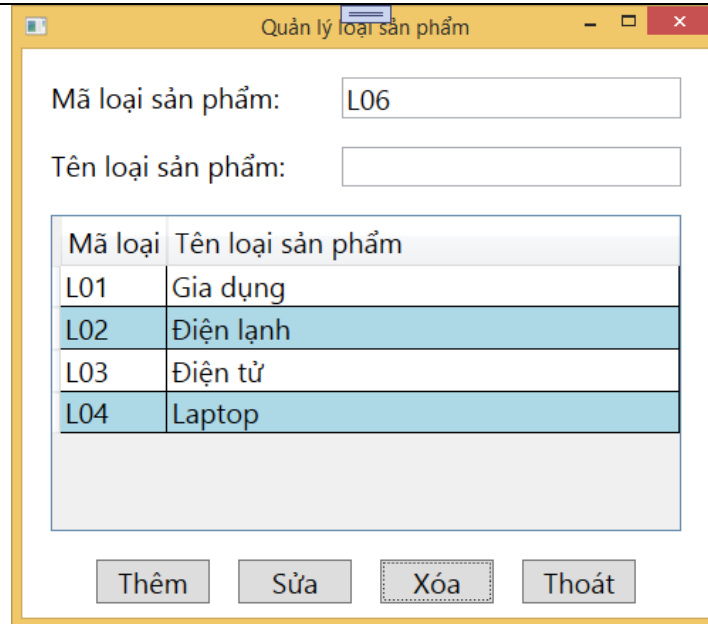
- Sử dụng LINQ lấy ra (các) đối tượng muốn xóa
- Xóa (các) đối tượng khỏi tập hợp tương ứng của lớp Context sử dụng phương thức **Remove()** hoặc **RemoveRange()**
- Thực thi phương thức **SubmitChanges()** của DataContext để cập nhật các thay đổi vào csdl

Ví dụ: xóa loại sản phẩm có mã trong

```
//1. lấy ra sản phẩm muốn xóa
var spSua = db.LoaiSanPhams.SingleOrDefault(lsp => lsp.MaLoai ==
txtMa.Text);
//2. xóa đối tượng khỏi tập hợp
db.LoaiSanPhams.Remove(spSua);
//3. lưu thay đổi vào csdl
db.SaveChanges();
```

Ví dụ 11.1: Truy vấn, cập nhật dữ liệu trên 1 bảng – bảng Loại sản phẩm

Giao diện



Mã loại	Tên loại sản phẩm
L01	Gia dụng
L02	Điện lạnh
L03	Điện tử
L04	Laptop

Yêu cầu:

- Ngay khi hiển thị cửa sổ, dữ liệu trong bảng Loại sản phẩm đã hiển thị trong data grid
- Người dùng nhập mã loại, tên loại sản phẩm sau đó nhấn nút Thêm, loại sản phẩm mới được thêm vào bảng. Có kiểm tra không cho thêm loại sản phẩm trùng mã. Hiển thị lại thông tin sau khi thêm.
- Người dùng nhập mã và tên mới của loại sản phẩm muốn sửa, sau đó nhấn nút Sửa, thông tin của loại sản phẩm được lưu vào bảng. Có thông báo nếu không tìm thấy loại sản phẩm muốn sửa. Hiển thị lại thông tin sau khi sửa
- Người dùng nhập mã của loại sản phẩm muốn xóa, sau đó nhấn nút Xóa, loại sản phẩm bị xóa khỏi bảng. Có thông báo nếu không tìm thấy loại sản phẩm muốn xóa. Hiển thị lại thông tin sau khi xóa.

Code cửa sổ Loại sản phẩm

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using DemoEFCore.Models;
namespace DemoEFCore
{
    /// <summary>
    /// Interaction logic for WindowLoaiSanPham.xaml
    /// </summary>
    public partial class WindowLoaiSanPham : Window
    {
        public WindowLoaiSanPham()
        {
            InitializeComponent();

            //Tạo thể hiện của lớp Context
        }
    }
}
```

```
QLBanHangContext db = new QLBanHangContext();

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    HienThiDuLieu();
}

private void HienThiDuLieu()
{
    //Truy vấn dữ liệu sử dụng LINQ
    var query = from lsp in db.LoaiSanPhams
                select lsp;
    //Hiển thị dữ liệu lên data grid
    dgvLoaiSanPham.ItemsSource = query.ToList();
}

private void btnThem_Click(object sender, RoutedEventArgs e)
{
    //Tạo đối tượng mới của lớp Loại sản phẩm
    LoaiSanPham lspMoi = new LoaiSanPham();
    //Thuộc tính của đối tượng loại sản phẩm mới do user nhập vào
    lspMoi.MaLoai = txtMa.Text;
    lspMoi.TenLoai = txtTen.Text;
    //nếu chưa có loại sản phẩm thì mới thêm, ngược lại thông báo lỗi
    if (!db.LoaiSanPhams.Contains(lspMoi))
    {
        //Thêm đối tượng vào tập hợp loại sản phẩm
        db.LoaiSanPhams.Add(lspMoi);
        //Lưu thay đổi vào cơ sở dữ liệu
        db.SaveChanges();
        //Hiển thị lại dữ liệu sau khi thêm
        HienThiDuLieu();
    }
    else
    {
        MessageBox.Show("Đã có sản phẩm " + txtMa.Text, "THÊM DỮ LIỆU",
            MessageBoxButton.OK, MessageBoxImage.Error);
    }
}

private void btnSua_Click(object sender, RoutedEventArgs e)
{
    //Lấy ra đối tượng loại sản muốn sửa
    var lspSua = (from lsp in db.LoaiSanPhams
                  where lsp.MaLoai == txtMa.Text
                  select lsp).SingleOrDefault();
    //Nếu có sản phẩm muốn sửa thì thực hiện cập nhật thông tin,
    //ngược lại hiển thị thông báo
    if (lspSua != null)
    {
        //sửa thông tin sản phẩm -- không sửa mã
        lspSua.TenLoai = txtTen.Text;
        //lưu thay đổi vào cơ sở dữ liệu
        db.SaveChanges();
        //Hiển thị lại dữ liệu sau khi sửa
        HienThiDuLieu();
    }
}
```

```

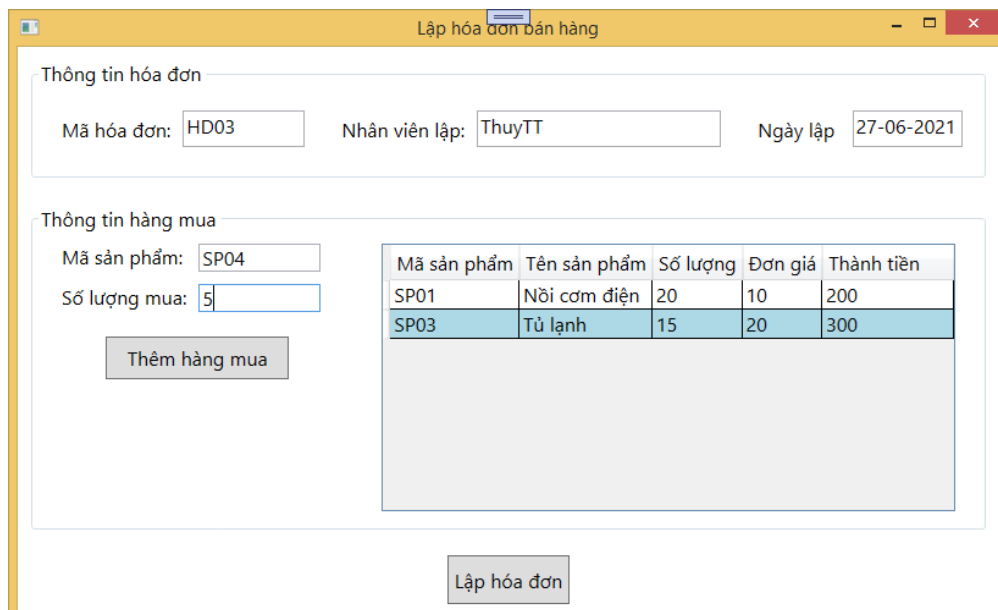
else
{
    MessageBox.Show("Không có sản phẩm mã " + txtMa.Text, "SỬA DỮ LIỆU",
        MessageBoxButton.OK, MessageBoxImage.Information);
}
}

private void btnXoa_Click(object sender, RoutedEventArgs e)
{
    //Lấy ra đối tượng loại sản phẩm muốn xóa
    var lspXoa = (from lsp in db.LoaiSanPhams
        where lsp.MaLoai == txtMa.Text
        select lsp).SingleOrDefault();
    //Nếu có sản phẩm muốn xóa thì thực hiện xóa, ngược lại hiển thị thông báo
    if (lspXoa != null)
    {
        //xóa sản phẩm
        db.LoaiSanPhams.Remove(lspXoa);
        //lưu thay đổi vào cơ sở dữ liệu
        db.SaveChanges();
        //Hiển thị lại dữ liệu sau khi xóa
        HienThiDuLieu();
    }
    else
    {
        MessageBox.Show("Không có sản phẩm mã " + txtMa.Text, "XÓA DỮ LIỆU",
            MessageBoxButton.OK, MessageBoxImage.Information);
    }
}
}
}

```

Ví dụ 11.2: Cập nhật dữ liệu trên 2 bảng – bảng Hóa đơn và Hóa đơn chi tiết

Giao diện:



Mã sản phẩm	Tên sản phẩm	Số lượng	Đơn giá	Thành tiền
SP01	Nồi cơm điện	20	10	200
SP03	Tủ lạnh	15	20	300

Yêu cầu:

- Khi hiển thị cửa sổ lập hóa đơn, ngày lập lấy là ngày hiện tại của hệ thống
- Người dùng nhập mã sản phẩm, số lượng mua vào text box sau đó nhấn nút Thêm hàng mua, thông tin mặt hàng vừa nhập được hiển thị lên data grid. Trong đó tên sản phẩm, đơn giá lấy từ bảng Sản phẩm, Thành tiền = số lượng mua * đơn giá
- Sau khi đã nhập thông tin hóa đơn, thông tin hàng mua. Người dùng nhấn nút Lập hóa đơn thì thông tin hóa đơn được lưu vào bảng Hóa đơn, thông tin hàng mua được lưu vào bảng Hóa đơn chi tiết

Code cửa sổ Lập hóa đơn:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Reflection;
using System.Windows;
using DemoEFCore.Models;
namespace DemoEFCore
{
    /// <summary>
    /// Interaction logic for WindowLapHoaDon.xaml
    /// </summary>
    public partial class WindowLapHoaDon : Window
    {
        public WindowLapHoaDon()
        {
            InitializeComponent();

            QLBanHangContext db = new QLBanHangContext();

            private void Window_Loaded(object sender, RoutedEventArgs e)
            {
                txtNgay.Text = DateTime.Now.ToString("dd-MM-yyyy");
            }

            private void btnThem_Click(object sender, RoutedEventArgs e)
            {
                //Lấy sản phẩm có mã nhập
                SanPham spMua = new SanPham();
                spMua = db.SanPhams.Where(sp => sp.MaSp ==
                                                                    txtMaSP.Text).SingleOrDefault();
                if (spMua!=null)
                {
                    //Thêm sản phẩm mua vào data grid
                    dgvSanPham.Items.Add(new
                    {
                        MaSP = spMua.MaSp,
                        TenSP = spMua.TenSp,
                        SoLuong = txtSoLuong.Text,
                        spMua.DonGia,
                        ThanhTien = int.Parse(txtSoLuong.Text) * spMua.DonGia
                    });
                    //Xóa các điều khiển
                }
            }
        }
    }
}
```

```

        txtMaSP.Text = "";
        txtSoLuong.Text = "";
        txtMaSP.Focus();
    }
    else
    {
        MessageBox.Show("Không có sản phẩm mã " + txtMaSP.Text,
            "LẬP HÓA ĐƠN", MessageBoxButtons.OK, MessageBoxImage.Error);
        //Xóa điều khiển text box Mã sản phẩm
        txtMaSP.Text = "";
        txtMaSP.Focus();
    }
}

private void btnLapHoaDon_Click(object sender, RoutedEventArgs e)
{
    //tạo đối tượng Hóa đơn
    HoaDon hdMoi = new HoaDon();
    hdMoi.MaHd = txtMaHD.Text;
    hdMoi.NguoiLap = txtNguoiLap.Text;
    //Thêm vào tập hợp Hóa đơn
    db.HoaDons.Add(hdMoi);

    //Thêm các hóa đơn chi tiết
    foreach (var item in dgvSanPham.Items)
    {
        Type t = item.GetType();
        PropertyInfo[] sanPhamInfo = t.GetProperties();
        HoaDonChiTiet hdctMoi = new HoaDonChiTiet();
        hdctMoi.Mahd = hdMoi.MaHd;
        hdctMoi.MaSp = sanPhamInfo[0].GetValue(item).ToString();
        hdctMoi.SoLuongMua =
            Convert.ToInt32(sanPhamInfo[2].GetValue(item));
        db.HoaDonChiTiets.Add(hdctMoi);
    }
    //Lưu vào cơ sở dữ liệu
    db.SaveChanges();
    MessageBox.Show("Đã lưu hóa đơn vào cơ sở dữ liệu", "LẬP HÓA ĐƠN",
        MessageBoxButtons.OK, MessageBoxImage.Information);
}
}
}

```