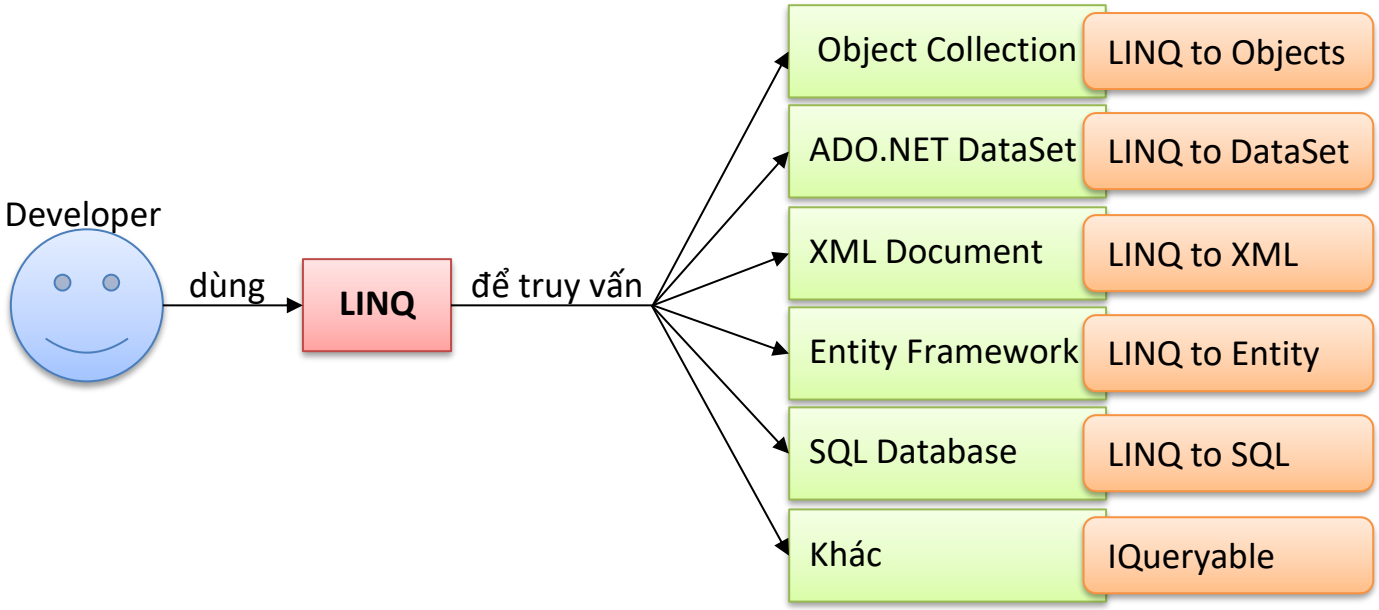
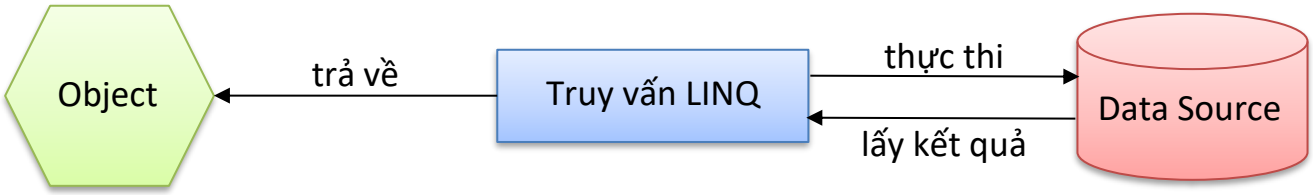
Buổi 6
Truy vấn với LINQ

I. LINQ là gì?

LINQ (**L**anguage **In**tegrated **Q**uery) là 1 cú pháp truy vấn được tích hợp trong C#, cho phép truy vấn dữ liệu từ nhiều nguồn, nhiều định dạng khác nhau như CSDL, DataSet, file XML… với cú pháp thống nhất:



Các câu truy vấn LINQ trả về kết quả dưới dạng object, do đó lập trình viên có thể thao tác trực tiếp với các kết quả này theo các phương pháp hướng đối tượng thay vì phải chuyển đổi kiểu dữ liệu nếu như sử dụng truy vấn SQL, XML… theo cách bình thường.



Ví dụ: Câu truy vấn LINQ trả về tất cả tài khoản trong `_context.Accounts`:

```
var lstAccount =  from account in _context.Accounts
                  select account;
```

→ `lstAccount` sẽ thuộc kiểu `IQueryable<Account>`, do đó có thể duyệt từng phần tử một cách dễ dàng bằng các phương pháp hướng đối tượng.

Một số ưu điểm của việc dùng LINQ so với cách truy vấn dữ liệu truyền thống:

- Cú pháp của LINQ quen thuộc với cú pháp SQL nên không mất nhiều thời gian làm quen.
- LINQ sử dụng 1 cú phép thống nhất cho tất cả các loại Data Source khác nhau, dù là CSDL, file XML, DataSet hay Entity Framework. Do đó, lập trình viên không cần phải học cú pháp truy vấn với từng loại Data Source khác nhau.
- Cú pháp của LINQ dễ đọc và có hỗ trợ IntelliSense.
- LINQ trả về object, lập trình viên có thể thao tác ngay với object này mà không cần chuyển đổi kiểu dữ liệu.

II. Cú pháp của LINQ

Để có thể sử dụng được LINQ, cần thêm 2 namespace sau:

```
System.Collections.Generic
System.Linq
```

1. LINQ Query

Câu truy vấn LINQ thường bắt đầu bằng mệnh đề `from`, theo sau đó có thể là các mệnh đề như `where`, `orderby`, `join`… hoặc các toán tử, cuối cùng kết thúc bằng mệnh đề `select` hoặc `group`, với cú pháp khá giống ngôn ngữ SQL:

```
from <variable> in <collection>
[query operator] [lambda expression]
select/group <result>
```

Trong đó:

- `<variable>` là tên biến do lập trình viên tự đặt, dùng để duyệt từng phần tử trong `<collection>`. Có thể xem như mệnh đề `from` này tương tự vòng lặp sau:
  `foreach (var <variable> in <collection>).`
- `[query operator]` là các toán tử chuẩn[1] của truy vấn LINQ.
- `[lambda expression]` là biểu thức lambda[2].
- `<result>` là biểu thức kết quả

Ví dụ: Sử dụng LINQ Query:

- Lấy ra tên tài khoản có `Id` cho trước:

```
var accountName = (from acc in _context.Accounts
                   where acc.Id == id
                   select acc.Username).FirstOrDefault();
```

- Lấy ra danh sách tài khoản có địa chỉ email thuộc Gmail:

```
var lstAccount = from acc in _context.Accounts
                 where acc.Email.Contains("@gmail.com")
                 select acc;
```

- Lấy ra danh sách các hóa đơn có tổng tiền từ 500.000 đến 1.000.000, sắp xếp theo ngày lập hóa đơn giảm dần:

```
var lstInvoice = from inv in _context.Invoices
                 where inv.Total >= 500000 && inv.Total <= 1000000
                 orderby inv.IssuedDate descending
                 select inv;
```

2. LINQ Method

Thay vì sử dụng câu truy vấn LINQ, chúng ta có thể sử dụng các phương thức LINQ tương đương như `Where()`, `OrderBy()`… để thực hiện truy vấn. Các phương thức này thường nhận vào tham số là các biểu thức lambda, và có thể được gọi liên tiếp nhau.

Các câu truy vấn LINQ cũng đều được trình biên dịch chuyển thành các phương thức LINQ tại thời điểm biên dịch.

Ví dụ: Sử dụng LINQ Method cho các ví dụ ở phần II.1 ở trên:

---

[1] Sẽ được trình bày ở phần IV
[2] Sẽ được trình bày ở phần III

- Lấy ra tên tài khoản có `Id` cho trước:

```
var accountName = _context.Accounts.Where(acc => acc.Id == id)
      .Select(acc => acc.Username).First();
```

hoặc

```
var accountName = _context.Accounts
      .First(acc => acc.Id == id).Username;
```

- Lấy ra danh sách tài khoản có địa chỉ email thuộc Gmail:

```
var lstAccount = _context.Accounts
      .Where(acc => acc.Email.Contains("@gmail")).ToList<Account>();
```

- Lấy ra danh sách các hóa đơn có tổng tiền từ 500.000 đến 1.000.000, sắp xếp theo ngày lập hóa đơn giảm dần:

```
var lstInvoice = _context.Invoices
      .Where(inv => inv.Total >= 500000 && inv.Total <= 1000000)
      .OrderByDescending(inv => inv.IssuedDate).ToList<Invoice>();
```

III. Biểu thức Lambda

1. Khái niệm và cú pháp

Biểu thức lambda[3] là 1 cách để cài đặt các *phương thức vô danh* (*anonymous method*)[4] bằng một số cú pháp đặc biệt.

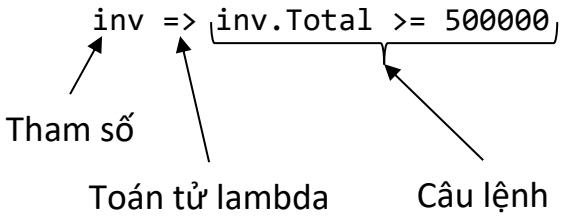
Giả sử cần lấy ra những hóa đơn có tổng tiền từ 500.000 trở lên. Chúng ta cần có 1 phương thức để kiểm tra tổng tiền 1 hóa đơn có ≥ 500.000 hay không. Phương thức này có thể được cài đặt theo cách bình thường, hoặc được cài như 1 anonymous method như sau:

```
delegate(Invoice inv) { return inv.Total >= 500000; }
```

Phương thức trên có thể được viết thành 1 biểu thức lambda bằng cách loại bỏ từ khóa `delegate` và kiểu dữ liệu của tham số, chèn thêm toán tử lambda `=>` như sau:

```
(inv) => { return inv.Total >= 500000; }
```

Đối với những phương thức chỉ chứa 1 câu lệnh, có thể loại bỏ cặp dấu ngoặc `{ }`, từ khóa `return` và dấu `;` cuối câu lệnh. Tương tự, đối với những phương thức chỉ nhận vào 1 tham số, có thể loại bỏ cặp dấu ngoặc `()`. Cuối cùng chúng ta có được biểu thức lambda như sau:



Lưu ý:

- Nếu có nhiều câu lệnh, vẫn phải giữ nguyên cặp dấu ngoặc `{ }` và cài đặt các câu lệnh như bình thường.

```
inv =>
{
      float tax = 0.1;
      return inv.Total * (1 + tax) >= 500000;
}
```

[3] Xem thêm về biểu thức lambda tại đây: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-expressions

[4] Xem thêm về anonymous method và delegate tại đây: https://www.tutorialsteacher.com/csharp/csharp-anonymous-method

- Nếu có nhiều tham số hoặc không có tham số, vẫn phải giữ nguyên cặp dấu ngoặc ( ). Có thể khai báo kiểu dữ liệu kèm theo tham số nếu cần:

```
(Invoice inv, float tax) => return inv.Total * (1 + tax) >= 500000
() => Console.WriteLine("Hello World")
```

- Biểu thức lambda không cần quy định kiểu trả về (*return type*). Kiểu trả về sẽ được xác định dựa vào các câu lệnh trong biểu thức này[5].

2. Cách thức hoạt động

Xét câu lệnh ví dụ sau:

```
var lstInvoice = _context.Invoices.Where(inv => inv.Total >= 500000);
```

Câu lệnh trên sử dụng phương thức LINQ `Where()` cho phép quy định điều kiện truy vấn cho collection `_context.Invoices`. Phương thức này nhận vào tham số là 1 biểu thức lambda.

Cách thức hoạt động của câu lệnh trên:

- Với mỗi phần tử của collection chứa trong `_context.Invoices`, phần tử đó sẽ được đưa vào phương thức `Where()` với tên biến là `inv`.
- Biểu thức lambda sẽ kiểm tra thuộc tính `Total` của phần tử đó xem có phù hợp với điều kiện `inv.Total >= 500000` hay không. Nếu đúng sẽ trả về `true`, ngược lại sẽ trả về `false`.
- Phương thức `Where()` sẽ lọc ra những phần tử được biểu thức lambda trả về giá trị true và đưa những phần tử này vào 1 collection kết quả.
- Sau khi mọi phần tử của collection `_context.Invoices` đã được xét hết, phương thức `Where()` sẽ trả về collection kết quả cuối cùng.

IV. Các toán tử chuẩn và phương thức mở rộng của LINQ

Các toán tử chuẩn của LINQ về bản chất chính là các phương thức mở rộng (*extension method*) cho các đối tượng collection thuộc kiểu `IEnumerable<T>` và `IQueryable<T>`, được dùng trong LINQ Query nhằm cung cấp 1 cách viết truy vấn LINQ gần giống với ngôn ngữ SQL.

Các toán tử này sẽ được biên dịch thành các phương thức mở rộng tại thời điểm biên dịch, do đó sử dụng toán tử cho LINQ Query hay sử dụng phương thức cho LINQ Method là như nhau:

```
// LINQ Query
var lstInvoices = from inv in _context.Invoices
                  where inv.Total >= 500000
                  select inv;

// LINQ Method
var lstInvoices = _context.Invoices
      .Where(inv => inv.Total >= 500000).ToList<Invoice>();
```

Toán tử LINQ

Phương thức LINQ

Có trên 50 toán tử LINQ được chia thành nhiều nhóm khác nhau dựa vào mục đích và chức năng của chúng, ví dụ như lọc dữ liệu (*filtering*), sắp xếp (*sorting*), gom nhóm (*grouping*), tập hợp (*aggregation*)... Trong bảng dưới đây, những nhóm toán tử được tô vàng là những nhóm thường dùng nhất.

[5] Tính chất này được biểu thức lambda kế thừa từ anonymous method

Loại	Toán tử
Filtering	Where
Sorting	OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
Grouping	GroupBy, ToLookUp
Projection	Select, SelectMany
Join	GroupJoin, Join
Aggregation	Aggregate, Average, Count, LongCount, Max, Min, Sum
Quantifiers	All, Any, Contains
Element	ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
Set	Distinct, Except, Intersect, Union
Partitioning	Skip, SkipWhile, Take, TakeWhile
Conversion	AsEnumerable, AsQueryable, ToArray, ToDictionary, ToList, Cast
Relation	Include

Các ví dụ sau đây đều được viết theo 2 cách tương đương: LINQ Query và LINQ Method.

1. Lọc dữ liệu theo điều kiện (Filtering)

Toán tử `Where` giúp lọc dữ liệu theo 1 điều kiện cho trước.

Ví dụ: Lọc ra những sản phẩm còn hàng và còn được kinh doanh:

```
var lstProducts = from prd in _context.Products
                  where prd.Stock > 0 && prd.Status == true
                  select prd;
var lstProducts = _context.Products
      .Where(prd => prd.Stock > 0 && prd.Status).ToList<Product>();
```

Có thể có nhiều toán tử `Where` trong cùng 1 câu truy vấn.

Ví dụ: Lọc ra những sản phẩm có giá từ 50.000 đến 100.000:

```
var lstProducts = from prd in _context.Products
                  where prd.Price >= 50000
                  where prd.Price <= 100000
                  select prd;
var lstProducts = _context.Products
      .Where(prd => prd.Price >= 50000)
      .Where(prd => prd.Price <= 100000).ToList<Product>();
```

2. Sắp xếp (Sorting)

Với LINQ Query, toán tử `OrderBy` cho phép sắp xếp danh sách kết quả theo 1 hoặc nhiều thuộc tính. Kiểu sắp xếp là tăng dần (`ascending`) hoặc giảm dần (`descending`).

Với LINQ Method, phương thức `OrderBy()` chỉ có thể sắp xếp theo 1 thuộc tính. Để sắp xếp theo nhiều thuộc tính, chúng ta phải dùng thêm phương thức `ThenBy()`. Ngoài ra, 2 phương thức này chỉ sắp xếp tăng dần. Nếu muốn sắp xếp giảm dần, chúng ta cần dùng 2 phương thức `OrderByDescending()` và `ThenByDescending()`.

Ngoài ra, LINQ Method hỗ trợ phương thức `Reverse()` để đảo ngược danh sách kết quả.

Ví dụ: Lấy ra danh sách hóa đơn, sắp xếp theo ngày lập hóa đơn giảm dần. Nếu hóa đơn được lập trong cùng 1 ngày, sắp xếp theo tổng tiền tăng dần:

```
var lstInvoices = from inv in _context.Invoices
                  orderby inv.IssuedDate descending, inv.Total ascending
                  select inv;
var lstInvoices = _context.Invoices
     .OrderByDescending(inv => inv.IssuedDate)
     .ThenBy(inv => inv.Total).ToList<Invoice>();
```

3. Gom nhóm (Grouping)

Toán tử GroupBy hoạt động tương tự như mệnh đề GROUP BY của ngôn ngữ SQL. Toán tử này sẽ trả về 1 danh sách kết quả được gom nhóm theo 1 thuộc tính nào đó. Danh sách kết quả trả về là 1 collection gồm nhiều phần tử, mỗi phần tử có Key chính là giá trị của thuộc tính được gom nhóm, và phần tử này lại là 1 collection chứa các đối tượng có cùng Key đó.

Ví dụ: Liệt kê danh sách các hóa đơn, gom nhóm theo ngày lập hóa đơn:

```
var lstInvoices = from inv in _context.Invoices
                  group inv by inv.IssuedDate;
var lstInvoices = _context.Invoices.GroupBy(inv => inv.IssuedDate);
```

Lưu ý: LINQ Query có thể kết thúc bằng Select hoặc GroupBy.

4. Phép chiếu (Projection)

Toán tử Select hoạt động tương tự như mệnh đề SELECT của ngôn ngữ SQL, dùng để chọn ra 1 thuộc tính cần truy vấn của đối tượng, hoặc chọn cả đối tượng đó.

Ví dụ:

- Liệt kê danh sách các sản phẩm:

```
var lstProducts = from prd in _context.Products
                  select prd;
var lstProducts = _context.Products.ToList<Product>();
```

- Liệt kê danh sách tên tài khoản:

```
var lstUsernames = from acc in _context.Accounts
                   select acc.Username;
var lstUsernames = _context.Accounts.Select(acc => acc.Username);
```

Mặc định toán tử Select chỉ có thể lấy ra 1 thuộc tính của đối tượng, hoặc cả đối tượng đó. Nếu muốn lấy ra nhiều thuộc tính, chúng ta có thể gộp những thuộc tính này thành 1 đối tượng mới như sau:

Ví dụ: Liệt kê danh sách tên tài khoản kèm theo SĐT:

```
var lstAccounts = from acc in _context.Accounts
                  select new { TenTK = acc.Username, SDT = acc.Phone };
var lstAccounts = _context.Accounts
     .Select(acc => new { TenTK = acc.Username, SDT = acc.Phone });
```

5. Kết bảng (Join)

Toán tử Join hoạt động tương tự như phép kết bảng INNER JOIN trong ngôn ngữ SQL, dùng để kết 2 bảng lại với nhau dựa trên 1 cặp thuộc tính.

Ví dụ: Liệt kê danh sách tên sản phẩm kèm theo tên loại sản phẩm của nó:

```
var lstProducts = from prd in _context.Products
                  join prdT in _context.ProductTypes
                  on prd.ProductTypeId equals prdT.Id
                  select new { Name = prd.Name, TypeName = prdT.Name };
```

```
var lstProducts = _context.Products.Join(
      _context.ProductTypes,
      prd => prd.ProductTypeId, prdT => prdT.Id,
      (prd, prdT) => new { Name = prd.Name, TypeName = prdT.Name });
```

Tuy nhiên, việc kết bảng chỉ cần thiết nếu khóa ngoại được lưu trữ theo cách 1 hoặc cách 2, tức là chỉ được lưu ở 1 phía[6]. Nếu khóa ngoại được lưu theo cách 3, tức là ở cả 2 phía, ở class tham chiếu sẽ có navigation reference property là 1 đối tượng đang được tham chiếu tới, do đó không cần phải sử dụng phép kết bảng nữa. Khi đó, câu truy vấn LINQ cho ví dụ ở trên như sau:

```
var lstProducts =  from prd in _context.Products
                   select new {
                       Name = prd.Name,
                       TypeName = prd.ProductType.Name
                   };
var lstProducts = _context.Products
      .Select(prd => new {
            Name = prd.Name, TypeName = prd.ProductType.Name
      });
```

6. Tính toán tổng hợp (Aggregation)

Các toán tử `Sum`, `Max`, `Min`, `Count`, `Average` giúp tính tổng, tìm giá trị lớn nhất, nhỏ nhất, đếm, tính trung bình cộng dựa trên các thuộc tính dạng số, hoạt động tương tự như các hàm tổng hợp cùng tên trong ngôn ngữ SQL. Các toán tử này chỉ có thể được dùng trong LINQ Method.

Ví dụ:

- Tính trung bình cộng tổng tiền của 1 hóa đơn:

```
double avg = _context.Invoices.Average(inv => inv.Total);
```

- Tính tổng tiền hàng còn tồn kho:

```
int sum = _context.Products.Sum(prd => prd.Price * prd.Stock);
```

7. Định lượng (Quantifier)

Toán tử `All` kiểm tra tất cả phần tử có thỏa mãn 1 điều kiện hay không.

Toán tử `Any` kiểm tra có phần tử nào thỏa mãn 1 điều kiện hay không.

Toán tử `Contains` kiểm tra có tồn tại phần tử có giá trị cụ thể hay không.

Các toán tử định lượng chỉ có thể được dùng trong LINQ Method.

Ví dụ:

- Kiểm tra tất cả sản phẩm đều còn hàng:

```
bool allInStock = _context.Products.All(prd => prd.Stock > 0);
```

- Kiểm tra có sản phẩm hết hàng hay không:

```
bool anyOutStock = _context.Products.Any(prd => prd.Stock == 0);
```

- Kiểm tra có tài khoản nào có tên tài khoản là *dhphuoc* hay không:

```
bool checkAcc = _context.Accounts.Select(acc => acc.Username)
      .Contains("dhphuoc");
```

8. Chọn lọc phần tử (Element)

Toán tử `ElementAt` trả về phần tử tại 1 vị trí trong danh sách.

Toán tử `First`, `Last` trả về phần tử đầu tiên hoặc cuối cùng của danh sách.

---

[6] Xem lại tài liệu buổi 5, phần II

Các toán tử `ElementAtOrDefault`, `FirstOrDefault`, `LastOrDefault` có công dụng tương tự như trên, nhưng nếu không tìm ra phần tử thì sẽ trả về giá trị mặc định của kiểu dữ liệu tương ứng (thay vì sinh ra 1 exception). Do đó, nên sử dụng các toán tử `OrDefault` này thay cho các toán tử ở trên.

Các toán tử chọn lọc phần tử chỉ có thể được dùng trong LINQ Method.

Ví dụ:

- Tìm hóa đơn mới nhất:

```
var latestInvoice = _context.Invoices
    .OrderByDescending(inv => inv.IssuedDate).FirstOrDefault();
```

- Tìm sản phẩm có giá cao nhất:

```
var maxPricePrd = _context.Products
    .OrderBy(prd => prd.Price).LastOrDefault();
```

Ngoài ra, các toán tử `First`, `FirstOrDefault`, `Last` và `LastOrDefault` cũng có thể tìm ra phần tử đầu tiên hoặc cuối cùng thỏa mãn 1 điều kiện nào đó trong danh sách.

Ví dụ: Tìm hóa đơn đầu tiên có giá trên 500.000:

```
var invoice = _context.Invoices.FirstOrDefault(inv => inv.Total > 500000);
```

9. Phân vùng (Partitioning)

Toán tử `Skip` cho phép bỏ qua 1 số lượng phần tử cụ thể, tính từ đầu danh sách, hoạt động tương tự như từ khóa `OFFSET` trong ngôn ngữ SQL.

Toán tử `Take` cho phép lấy 1 số lượng phần tử cụ thể, tính từ đầu danh sách, hoạt động tương tự như từ khóa `TOP` trong ngôn ngữ SQL.

Kết hợp cả `Skip` và `Take` cho phép lấy 1 đoạn phần tử cụ thể trong danh sách kết quả, hoạt động tương tự như cụm `OFFSET … FETCH …` trong ngôn ngữ SQL.

Các toán tử phân vùng chỉ có thể được dùng trong LINQ Method.

Ví dụ:

- Liệt kê danh sách sản phẩm, kể từ sản phẩm thứ 4 trở đi:

```
var lstProducts = _context.Products.Skip(3);
```

- Liệt kê danh sách 5 hóa đơn mới nhất:

```
var lstInvoices = _context.Invoices
    .OrderByDescending(inv => inv.IssuedDate).Take(5);
```

- Liệt kê danh sách các sản phẩm từ 31-40 (thường áp dụng trong kỹ thuật phân trang):

```
var lstProducts = _context.Products.Skip(30).Take(10);
```

Toán tử `SkipWhile` và `TakeWhile` hoạt động tương tự như `Skip` và `Take`, nhưng cho phép quy định điều kiện bỏ qua/lấy phần tử thay vì số lượng.

Ví dụ:

- Liệt kê danh sách hóa đơn, bỏ qua các hóa đơn dưới 500.000 đầu tiên:

```
var lstInvoices = _context.Invoices
    .SkipWhile(inv => inv.Total < 500000);
```

- Liệt kê danh sách sản phẩm, tính từ sản phẩm còn hàng đầu tiên:

```
var lstProducts = _context.Products.TakeWhile(prd => prd.Stock > 0);
```

10.Chuyển đổi kiểu dữ liệu (Conversion)

Có 3 loại toán tử chuyển đổi kiểu dữ liệu là As, To và Cast. Tất cả các toán tử chuyển đổi kiểu dữ liệu đều chỉ có thể được dùng trong LINQ Method.

a) As

Toán tử `AsEnumerable` và `AsQueryable` lần lượt chuyển đổi đối tượng thành kiểu `IEnumerable` và `IQueryable`. 2 kiểu này có điểm giống và khác nhau như sau:

- `IQueryable` kế thừa từ `IEnumerable`, do đó `IQueryable` có toàn bộ đặc tính của `IEnumerable` và có thêm các đặc tính riêng của nó.
- `IEnumerable` thuộc namespace `System.Collections` còn `IQueryable` thuộc namespace `System.Linq`.
- Cả 2 đều chỉ có thể duyệt phần tử theo 1 chiều.
- `IEnumerable` dùng tốt nhất với dữ liệu được lưu tạm thời trong RAM như List, Array… trong khi `IQueryable` dùng tốt nhất với dữ liệu được lưu ngoài RAM như CSDL.
- Khi truy vấn dữ liệu từ CSDL, `IEnumerable` sẽ lấy toàn bộ dữ liệu từ server về lưu ở client, sau đó mới thực hiện lọc dữ liệu còn `IQueryable` sẽ lọc dữ liệu trực tiếp ở server.

b) To

Toán tử `ToArray`, `ToList` và `ToDictionary` lần lượt chuyển đổi đối tượng thành kiểu `Array` (mảng), `List` (danh sách) và `Dictionary`.

Ví dụ:

```
IList<Account> lstAccounts = _context.Accounts.ToList<Account>();

// Chuyển đổi 1 List thành Array
Account[] arrAccounts = lstAccounts.ToArray<Account>();

// Chuyển đổi 1 Array thành List
IList<Account> lstAccounts = arrAccounts.ToList<Account>();

// Chuyển đổi 1 List thành Dictionary
// với key là Account.ID và value là object Account
IDictionary<int, Account> dicAccounts
     = lstAccounts.ToDictionary<Account, int>(acc => acc.Id);
```

c) Cast

Toán tử `Cast` chuyển đổi đối tượng thành kiểu `IEnumerable` tương tự toán tử `AsEnumerable`.

Ví dụ: Giả sử có object `arrAccounts` là 1 mảng (Array) chứa các tài khoản. Khi đó, 3 câu lệnh sau là như nhau:

```
IEnumerable<Account> lstAccounts = arrAccounts.AsEnumerable();
IEnumerable<Account> lstAccounts = (IEnumerable<Account>)arrAccounts;
IEnumerable<Account> lstAccounts = arrAccounts.Cast<Account>();
```

11.Lấy dữ liệu có liên quan (Relation)

Giả sử chúng ta cần lấy ra danh sách những hóa đơn của tài khoản *dhphuoc*:

```
var lstInvoices = _context.Invoices.Where(inv => inv.Username == "dhphuoc");
```

Cách viết ở trên sẽ tương đương với câu truy vấn SQL sau:

```
SELECT * FROM Invoices WHERE Username = 'dhphuoc'
```

Hiển nhiên, cả câu truy vấn LINQ và SQL ở trên đều báo lỗi cú pháp vì đối tượng Invoice/bảng Invoices không chứa thuộc tính Username. Bảng Invoices có khóa ngoại tham chiếu đến bảng Accounts, và bảng này mới có thuộc tính Username.

Khi truy vấn, nếu cần truy cập đến các thuộc tính của bảng khác, chúng ta có thể sử dụng toán tử `Include` của LINQ để load thêm dữ liệu của bảng đó vào:

```
var lstInvoices = _context.Invoices.Include(inv => inv.Account)
      .Where(inv => inv.Account.Username == "dhphuoc");
```

Khi dùng toán tử `Include`, ngay khi load dữ liệu cho collection `_context.Invoices`, LINQ cũng sẽ load dữ liệu của mỗi tài khoản tương ứng với hóa đơn đó. Cách load dữ liệu này được gọi là *Eager Loading*[7].

V. Deferred Execution và Immediate Execution trong LINQ

Các câu truy vấn trong LINQ có thể được thực thi tức thời (*immediate execution*) hoặc thực thi trì hoãn (*deferred execution*).

1. Thực thi trì hoãn (Deferred Execution)

Xét ví dụ sau trong môi trường console:

- Cho sẵn class `Employee` mô phỏng 1 nhân viên như sau:

```
class Employee
{
      public int ID { get; set; }
      public string Name { get; set; }
      public int Age { get; set; }
}
```

- Tạo 1 List các nhân viên:

```
List<Employee> lstEmployees = new List<Employee>();
lstEmployees.Add(new Employee { ID = 1, Name = "Jack", Age = 32 });
lstEmployees.Add(new Employee { ID = 2, Name = "Rahul", Age = 35 });
lstEmployees.Add(new Employee { ID = 3, Name = "Sheela", Age = 28 });
lstEmployees.Add(new Employee { ID = 4, Name = "Mary", Age = 25 });
```

- Yêu cầu: Dùng LINQ để lấy ra tên các nhân viên có tuổi > 28, sau đó xuất ra màn hình:

```
var result = from emp in lstEmployees
             where emp.Age > 28
             select emp.Name;
foreach (string name in result)
{
      Console.WriteLine(name);
}
```

➔ Kết quả xuất ra màn hình gồm có Jack và Rahul.

Với ví dụ trên, có vẻ như câu truy vấn LINQ được thực thi tại thời điểm nó được tạo ra (tại vị trí dấu mũi tên). Tuy nhiên thực tế không phải như vậy. Câu truy vấn này sẽ chỉ được thực thi khi biến `result` được duyệt (ở vòng lặp `foreach`). Cách thực thi này được gọi là thực thi trì hoãn (*deferred execution*).

Để chứng minh cho điều này, chúng ta sẽ bổ sung thêm 1 câu lệnh ngay trước vòng lặp `foreach` và thêm 1 nhân viên vào danh sách như sau:

---

[7] Xem thêm về Eager Loading tại đây: https://www.entityframeworktutorial.net/eager-loading-in-entity-framework.aspx

```
lstEmployees.Add(new Employee { ID = 5, Name = "Bill", Age = 39 });
foreach (string name in result) { Console.WriteLine(name); }
```

➔ Kết quả xuất ra màn hình gồm có Jack, Rahul và Bill.

Như vậy, mặc dù nhân viên *Bill* được thêm vào sau khi câu truy vấn được tạo ra, nhưng kết quả xuất ra màn hình vẫn có nhân viên này, chứng tỏ câu truy vấn được thực thi tại thời điểm vòng lặp `foreach` được chạy.

Deferred execution cho lập trình viên sự linh hoạt trong viết truy vấn, thậm chí có thể tách nhỏ 1 câu truy vấn phức tạp thành nhiều câu truy vấn con. Ngoài ra, thực thi trì hoãn theo cách này giúp dữ liệu lấy được luôn là dữ liệu mới nhất, rất hữu ích khi dữ liệu có thể bị thay đổi liên tục.

2. Thực thi tức thời (Immediate Execution)

Nhược điểm của thực thi trì hoãn là nếu dùng với dữ liệu không có nhiều thay đổi thì cứ mỗi lần cần lấy dữ liệu, câu truy vấn lại phải thực thi 1 lần. Điều này sẽ tăng tần suất thao tác với CSDL và tốn tài nguyên của máy chủ. Chúng ta cũng có thể ép 1 câu truy vấn phải thực thi ngay lúc nó được tạo ra. Cách thực thi này được gọi là thực thi tức thời (*immediate execution*).

Xét ví dụ trên, nhưng câu truy vấn sẽ được thay đổi như sau:

```
var result = (from emp in lstEmployees
              where emp.Age > 28
              select emp.Name).ToList();
```

Như chúng ta đã tìm hiểu ở phần IV.10, phương thức `ToList()` cho phép chuyển 1 object thành kiểu `List`. Do đó, câu truy vấn bắt buộc phải thực thi ngay tại thời điểm này để cho ra kết quả và chuyển thành danh sách. Kết quả xuất ra màn hình trong ví dụ này gồm có Jack và Rahul.

Ngoài phương thức `ToList()`, các phương thức LINQ có thao tác với danh sách kết quả truy vấn ngay lập tức như `ToArray()`, `ToDictionary()`, `Count()`, `Sum()`… đều ép câu truy vấn phải thực thi tức thời.

VI. Bài tập

Tạo các View hiển thị dữ liệu theo yêu cầu sau:

Controller	View (.cshtml)	Dữ liệu hiển thị
Accounts	ByAddress	Tài khoản có địa chỉ ở Tp.Hồ Chí Minh
	ByEmail	Tài khoản có email thuộc Gmail
Products	ByPriceRange	Sản phẩm có giá từ 40.000 đến 60.000
	ByProductType	Sản phẩm thuộc loại *Sách*
	ByStock	Sản phẩm sắp hết hàng (tồn kho < 10)
Invoices	ByTotalRange	Hóa đơn có tổng tiền từ 400.000 đến 600.000
	ByAccountName	Hóa đơn được mua bởi tài khoản có username là *john*
	ByAccountAddress	Hóa đơn được mua bởi tài khoản ở Tp.Hồ Chí Minh
	ByProduct	Hóa đơn có sản phẩm có SKU là *VIAZXR3Y24IY*
	ByProductType	Hóa đơn có sản phẩm thuộc loại Tiểu thuyết &Tự truyện