Mục lục

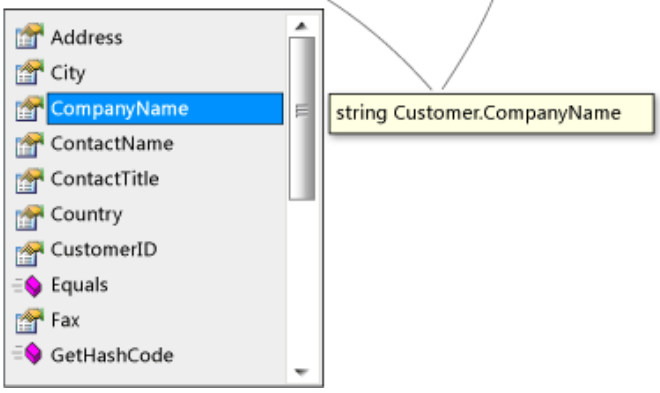
I. Giới thiệu về LINQ.

LINQ là viết tắt của từ Language – Integrated Query tạm dịch là ngôn ngữ tích hợp truy vấn là một sự đổi mới trong Visual Studio 2008 và .NET Framework 3.5 là cầu nối khoảng cách giữa thế giới của các đối tượng với thế giới của dữ liệu.

Theo truyền thống các câu truy vấn trên dữ liệu được thể hiện một cách dễ dàng giống như các chuỗi kí tự đơn giản mà không cần đến kiểu kiểm tra tại thời điểm biên dịch hoặc sự hỗ trợ của trình hỗ trợ trực quan. Hơn nữa bạn cần phải tìm hiểu một ngôn ngữ truy vấn khác nhau cho mỗi loại dữ liệu liệu nguồn khác nhau như: Cở sở dữ liệu SQL, tài liệu XML, các dịch vụ Web. LINQ làm cho một truy vấn một lớp đầu tiên xây dựng trong ngôn ngữ C# và Visual Basic. Bạn viết một câu truy vấn dựa trên tập hợp các đối tượng bằng cách sử dụng ngôn ngữ, các từ khóa các toán tử quen thuộc. Ví dụ minh họa sau đây cho thấy một phần câu truy vấn được hoàn thành dựa trên cơ sở dữ liệu SQL Server trong C# với đầy đủ loại kiểm tra và sự hỗ trợ của trình hỗ trợ trực quan.

```
Northwnd nw = new
Northwnd(@"northwnd.mdf");

var companyNameQuery =
   from cust in nw.Customers
   where cust.
```

```
Dim nw As New _
Northwnd("c:\northwnd.mdf")

Dim companyNameQuery = _
   From cust In nw.Customers _
   Where cust.
```



Trong Visual Studio 2008 bạn có thể viết các câu truy vấn LINQ trong Visual Basic hoặc C# với cơ sở dữ liệu SQL Server, các tài liệu XML, ADO.NET Datasets và bất kỳ tập đối tượng được hỗ trợ IEnumerable hoặc có đặc điểm chung giống giao diện IEnumerable<T>. LINQ hỗ trợ cho các thực thể ADO.NET Framework và LINQ đang được các nhà cung cấp hiện nay viết bởi bên thứ ba cho nhiều dịch vụ Web và các triển khai dữ liệu khác. Bạn có thể sử dụng các truy vấn LINQ trong các dự án mới hoặc trong các dự án hiện có. Một yêu cầu duy nhất là các dự án đó được xây dựng trên .NET Framework 3.5.

II. Giới thiệu về các truy vấn LINQ.

Một câu truy vấn là một biêu thức gọi ra dữ liệu từ dữ liệu nguồn. Câu truy vấn thường nói rõ trong ngôn ngữ truy vấn đữ được thiết kế cho mục dích riêng. Các ngôn ngữ khác nhau đã được phát triển theo thời gian cho các loại dữ liệu ngồn, ví dụ như SQL dành cho cơ sở dữ liệu quan hệ và XQuery dành cho XML. Vì vậy các nhà phát triển đã tìm hiểu một ngôn ngữ truy vấn mới cho các loại dữ liệu nguồn hoặc các định dạng mà họ phải hỗ trợ. LINQ đơn giản tình trạng này bằng cách cung cấp một mô hình nhất quán để làm việc với các loại dữ liệu nguồn khác nhau và các định dạng. Trong một truy vấn LINQ bạn phải luôn luôn làm việc với các đối tượng. Bạn sử dụng giống như truy vấn mẫu cơ bản mã hóa và chuyển đổi dữ liệu trong các tài liệu XML, cơ sở dữ liệu SQL, ADO.NET DataSet và cho bất kì một định đạng nào mà một nhà cung cấp LINQ có sẵn.

II.1 Ba phần của một biểu thức LINQ.

Tất cả các biểu thức LINQ làm việc theo ba thao tác.

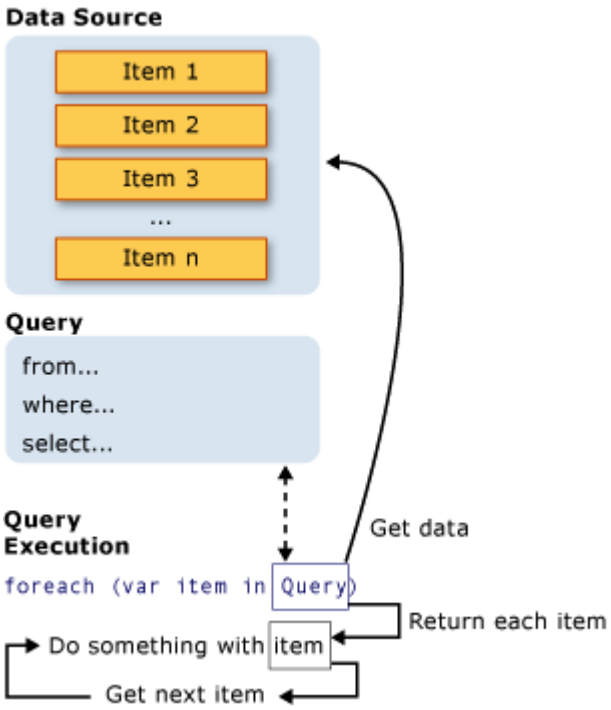
1. Có được các dữ liệu nguồn.
2. Tạo các truy vấn.
3. Thực hiện các truy vấn.

Ví dụ trong mã nguồn sau đây cho thấy ba phần của một truy vấn hoạt động như thế nào. Ví dụ sử dụng một mảng số nguyên như là một sự thay thế cho nguồn dữ liệu; tuy

nhiên, trong cùng một khái niệm áp dụng cho các nguồn dữ liệu khác cũng có. Ví dụ này sẽ được giới thiệu đến trong suốt phần còn lại của chủ đề này.

```
class IntroToLINQ
{
    static void Main()
    {
        // The Three Parts of a LINQ Query:
        //  1. Data source.
        int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

        // 2. Query creation.
        // numQuery is an IEnumerable<int>
        var numQuery =
            from num in numbers
            where (num % 2) == 0
            select num;

        // 3. Query execution.
        foreach (int num in numQuery)
        {
            Console.Write("{0,1} ", num);
        }
    }
}
```

Minh họa sau đây cho thấy các hoạt động truy vấn tìm kiếm được hoàn tất. Trong LINQ việc thực hiện các truy vấn riêng biệt từ bản thân câu truy vấn. Nói cách khác bạn không lấy ra bất kỳ dữ liệu nào bằng cách tạo ra một biến truy vấn.



II.2 Các dữ liệu nguồn.

Trong ví dụ trước vì dữ liệu là một mảng, nó hoàn toàn hỗ trợ đặc điểm chung giao diện IEnumerable <T>. Điều này có nghĩa thực tế nó có thể được truy vấn với LINQ. Một truy vấn được thực hiện trong một câu lệnh foreach và foreach yêu cầu IEnumerable hay IEnumerable(T). Loại có hỗ trợ IEnumerable(T) hoặc một giao diện như IQueryable(T) được gọi là các loại queryable. Một loại queryable không yêu cầu phải sửa đổi hay xử lý đặc biệt để phục vụ một LINQ nguồn dữ liệu. Nếu các nguồn dữ liệu không phải là đã có trong bộ nhớ như là một loại queryable, một nhà cung cấp LINQ phải đại diện cho nó như vậy. Ví dụ, LINQ to XML một tài liệu XML vào một queryable XElement:

```
//  Query execution.
foreach (int num in numQuery)
{
   Console.Write("{0,1} ", num);
}
```

Với LINQ to SQL trước tiên bạn tạo một đối tượng quan hệ được ánh xạ vào lúc thiết kế cái này được làm thủ công hoặc bằng cách sử dụng trình thiết kế đối tượng quan hệ(O/R Designer). Bạn viết các câu truy vấn của bạn dựa trên đối tượng và thi hành

LINQ to SQL để sử lý các giao tiếp với cơ sở dữ liệu. Trong ví dụ sau, Customer đại diện cho một bảng trong cơ sở dữ liệu, và Table<Customer> hỗ trợ các đặc tính chung IQueryable<T> mà được bắt đầu từ IEnumerable<T>.

```
// Create a data source from a SQL Server database.
// using System.Data.Linq;
DataContext db = new DataContext(@"c:\northwind\northwnd.mdf");
```

II.3 Truy vấn.

Truy vấn trong ví dụ trước trả về tất cả các số từ mảng số nguyên. Các biểu thức truy vấn chứa ba mệnh đề: from, where, select.(Nếu bạn đang quen với SQL sắp đặt của các mệnh đề là sai vị trí trong SQL). Mệnh đề from dùng để xác định nguồn dữ liệu, mệnh đề where dùng để lọc dữ liệu, mệnh đề select dùng để chọn ra những phần tử được trả về. các mệnh đề này và các mệnh đề truy vấn khác sẽ được thảo luận chi tiết trong phần LINQ Query Expresstions(Hướng dẫn lập trình C#). Lúc này một điểm quan trọng là trong LINQ, các biến truy vấn tựu nó không hành động và trả về không có dữ liệu. Nó chỉ chứa đựng thông tin đó là yêu cầu từ kết quả trả về khi câu truy vấn được thực hiện tại một số điểm sau.

II.3 Thực thi truy vấn.

Hoãn thực thi.

Cũng giống như trạng thái trước đây, biến truy vấn tự nó chỉ chứa các lệnh truy vấn. Hiện nay sự thực thi của các truy vấn là hoãn lại đến tận khi bạn nhắc lại đối với biến truy vấn trong câu lệnh foreach. Cái này làm cơ sở để quy cho hoãn thực thi và là cái điển hình trong ví dụ sau:

```
//  Query execution.
foreach (int num in numQuery)
{
   Console.Write("{0,1} ", num);
}
```

Câu lệnh foreach là nơi các kết quả truy vấn được trả về ví dụ các truy vấn trước đó biến *num* được lặp và nắm dữ mỗi giá trị trong trình tự trả về. Bởi các biến truy vấn tự

nó không bao giờ chứa kết quả truy vấn, bạn có thể thực hiện nó thường xuyên như bạn muốn. Ví dụ bạn dang có một cơ sở dữ liệu mà đang được cập nhập liên tục bởi một ứng dụng riêng biệt. Trong ứng dụng của bạn, bạn có thể tạo một truy vấn để lấy ra dữ liệu mới nhất và bạn có thể thi hành nó một cách liên tục tại một khoảng thời gian để lấy kết quả mỗi lần.

II.4 Thực thi bắt buộc tức thời.

Truy vấn mà sự kết hợp thực hiện các chức năng trên một loạt các phần tử nguồn đầu tiên phải lặp đi lặp lại trên những nhần tử. Ví dụ như các truy vấn Count, Max, Average, và First. Những thực thi mà không có một câu lệnh foreach nào rõ ràng bởi vì các truy vấn tự nó phải sử dụng foreach để trả về là một kết quả. Cũng lưu ý rằng các loại truy vấn trả lại một giá trị, không phải là một tập IEnumerable. Các truy vấn sau đây sẽ trả về một số lượng các số trong mảng nguồn:

```
var evenNumQuery =
   from num in numbers
   where (num % 2) == 0
   select num;
int evenNumCount = evenNumQuery.Count();

List<int> numQuery2 =
   (from num in numbers
    where (num % 2) == 0
    select num).ToList();
// or like this:
// numQuery3 is still an int[]
var numQuery3 =
   (from num in numbers
    where (num % 2) == 0
    select num).ToArray();
```

III. Tổng quan về thiết kế O / R.

Thiết kế đối tượng quan hệ (O / R Designer) cung cấp một hình ảnh thiết kế trực quan cho việc tạo LINQ to SQL để tổ chức các thực thể và sự kết hợp (các mối quan hệ) dựa trên các đối tượng trong một cơ sở dữ liệu. Nói cách khác, các O / R được thiết kế sử dụng để tạo ra một mô hình đối tượng trong một ứng dụng để ánh xạ các đối tượng trong một cơ sở dữ liệu. Nó cũng tạo ra một mạnh kiểu rõ ràng DataContext được sử dụng để

gửi và nhận dữ liệu giữa các lớp thực thể và cơ sở dữ liệu. O / R Designer cũng cung cấp tính năng cho việc ánh xạ các thủ tục lưu trữ và các hàm để thục hiện các phương thức trong DataContext trả về các dữ liệu và populating tổ chức các lớp thực thể. Cuối cùng, O / R Designer cung cấp khả năng thiết kế thừa kế các mối quan hệ giữa các lớp thực thể.

O / R Designer tạo ra những file có phần mở rông là .dbml cung cấp cho việc ánh xạ giữa các lớp LINQ to SQL và các đối tượng dữ liệu. O / R Designer cũng tạo ra những kiểu DataContext và tổ chức các lớp thực thể.

O / R Designer có hai khu vực riêng biệt trên bề mặt thiết kế: các thực thể trong của sổ bên trái, và các phương thức trong cửa sổ bên phải. Cửa sổ các thực thể chính là bề mặt thiết kế lớp thực thể, các sự kết hợp, và các bậc kế thừa. Các phương thức trong của sổ bên phảilà bề mặt thiết kế có hiển thị DataContext các phương thức ánh xạ để lưu trữ các thủ tục và hàm.

III.1 Mở các O / R Designer.

Bạn có thể mở các O / R bằng cách thêm một class mới vào dự án

O / R Designer là một cách dễ dàng đẻ ánh xạ các đối tượng bởi vì nó chỉ hỗ trợ ánh xạ 1:1các mối quan hệ. Nói cách khác, một lớp thực thể có thể chỉ có một ánh xạ 1:1trong mối quan hệ với một cơ sở dữ liệu hoặc view. Ánh xạ phức tạp, chẳng hạn như ánh xạ một lớp thực thể tham gia vào một bảng, hiện chưa hỗ trợ. Ngoài ra, các nhà thiết kế là một sinh mã tự động một chiều. Điều này có nghĩa là chỉ thay đổi mà bạn thực hiện để các nhà thiết kế bề mặt được phản ánh trong các tập tin code. Hướng dẫn để thay đổi các tập tin code không được phản ánh trong O / R Designer. Bất kỳ thay đổi nào mà bạn làm thủ công trong các tập tin mã được ghi đè khi thiết kế được lưu và code là tự phục hồi.

III.2 Cấu hình và tạo ra DataContext

Sau khi bạn thêm một lớp LINQ cho SQL cho một mục dự án và mở O / R Designer thiết kế, các thiết kế bề mặt trống rỗng đại diện một DataContext sẵn sàng để được cấu hình. các DataContext được cấu hình kết nối với các thông tin được cung cấp

bởi các phần tử đầu tiên được kéo vào cho việc thiết kế .. Vì vậy, các DataContext được cấu hình bằng cách sử dụng kết nối thông tin từ các phần tử dầu tiên được kéo vào thiết kế bề mặt thiết kế.

III.3 Tạo tổ chức các lớp mà cơ sở dữ liệu bản đồ để bàn và xem.

Bạn có thể tạo các lớp thực thể được ánh xạ từ các bảng và các view bằng cách kéo thả các cơ sở dữ liệu và các view Server Explorer / Explorer Database lên các O / R Designer. Như chỉ định trong phần trước của DataContext được cấu hình kết nối với các thông tin được cung cấp bởi các phần tử đầu tiên được kéo thả vào bề mặt thiết kế. Nếu một mục sau mà sử dụng một kết nối khác sẽ được thêm vào O / R Designer, bạn có thể thay đổi kết nối cho các DataContext.

III.4 DataContext tạo ra phương pháp gọi thủ tục lưu trữ và các hàm.

Bạn có thể tạo DataContext chưa các phương thức mà gọi (được ánh xạ tới) các thủ tục và các hàm lưu trữ bằng cách kéo chúng từ Server Explorer / Explorer Database lên các O / R Designer. Các thủ tục lưu trữ và các hàm được đưa vào các O / R Designer như phương thức của DataContext.

III.5 Cấu hình một DataContext để sử dụng các thủ tục lưu trữ dữ liệu lưu dữ liệu giữa các lớp thực thể và cơ sở dữ liệu.

Như đã nêu trên, bạn có thể tạo DataContext chứa các phương thức gọi các thủ tục lưu trữ và các hàm. Ngoài ra, bạn cũng có thể chỉ định các thủ tục lưu trữ được sử dụng mặc định cho LINQ to SQL để thực hiện hành động insert, update, và delete.

III.6 Thừa kế và các O / R Designer

Giống như các đối tượng khác, các lớp LINQ to SQL có thể sử dụng được kế thừa và thu được từ các lớp. Trong một cơ sở dữ liệu, các mối quan hệ thừa kế được tạo ra trong một số cách. O / R Designer hỗ trợ các khái niệm về đơn-bảng kế thừa như nó thường triển khai thực hiện trong các hệ thống.

IV. Các truy vấn LINQ to SQL.

IV.1 Tách rời DataContext đã tạo ra và các lớp thực thể vào các namespaces khác nhau

O / R Designer cung cấp cho các thuộc tính Context Namespace và Entity Namespace trên DataContext. Những thuộc tính xác định tên DataContext và các lớp thực thể đã được tạo ra. Theo mặc định, các thuộc tính là trống rỗng và các DataContext và các lớp thực thể được tạo ra vào ứng dụng của namesapce. Để tạo ra các mã vào một namespace khác các ứng dụng của namespace, nhập một giá trị vào trong thuộc tính Context Namespace và / hoặc Entity Namespace.

IV.2 Làm thế nào để: Chỉ định lưu trữ Thực hiện thủ tục Update, Insert, và delete

Thủ tục lưu trữ có thể được đưa vào các O / R Designer và thực hiện như các phương thức điển hình trong DataContext. Chúng cũng có thể được sử dụng để phủ quyết các phương thức mặc định trong LINQ to SQLđể thực hiện hành vi thêm, cập nhật, và xóa khi các thay đổi đều được lưu từ các thực thể để tổ chức một cơ sở dữ liệu (ví dụ, khi gọi các phương thức SubmitChanges).

Nếu thủ tục lưu trữ của bạn trả về giá trị mà cần phải được gửi lại cho client (ví dụ, giá trị tính toán trong thủ tục lưu trữ), tạo ra tham số của bạn được lưu trữ trong các thủ tục. Nếu bạn không thể sử dụng tham số, viết một phần phương thức một phần của việc triển khai thực hiện thay vì dựa vào các phủ quyết được tạo ra bởi các O / R Designer. Các thành viên được ánh xạ để tạo ra các giá trị cho cơ sở dữ liệu cần phải được thiết lập thích hợp cho các giá trị sau khi hoàn tất thành công của qua trình INSERT hoặc UPDATE.

V. LINQ và các kiểu có chung đặc điểm

Các câu truy vấn LINQ được dựa trên các loại có chung đặc điểm, đã được giới thiệu trong phiên bản 2.0 của. NET Framework. Bạn không cần phải có kiến thức đi vào tìm hiểu sâu các đặc điểm chung trước khi bạn có thể bắt đầu viết truy vấn. Tuy nhiên, bạn có thể muốn hiểu rõ hai khái niệm cơ bản:

1. Khi bạn tạo một ví dụ của một tập hợp có chung đặc điểm như List(T), bạn thay thế "T" với các loại đối tượng trong danh sách đó sẽ chứa. Ví dụ, một danh sách các xâu kí tự được thể hiện như *List<string>* , và một danh sách *Customer* các đối tượng khách hàng được thể hiện như *List<Customer>*. Một danh sách chung có kiểu sinh động và cung cấp nhiều lợi ích hơn một tập hợp nó cất giữ các phần tử của chúng như đối tượng. Nếu bạn cố gắng để thêm một Customer vào trong mộ *List<string>* , bạn sẽ nhận được một lỗi tại thời gian biên soạn. Nó là một cách dễ dàng để sử dụng chung các tập hợp vì bạn không có thể thực hiện các hoạt động đã được phân loại.
2. ***IEnumerable (T)*** là giao diện cho phép tập hợp các lớp để liệt kê bằng cách sử dụng câu lệnh foreach. Tập hợp chung các lớp hỗ trợ ***IEnumerable (T)*** cũng giống như tập hợp các lớp không chung chẳng hạn như ***IEnumerable*** hỗ trợ ArrayList.

V.1 IEnumerable các biến trong các câu truy vấn LINQ

Các biến trong câu truy vấn LINQ có kiểu như ***IEnumerable (T)*** hoặc có kiểu bắt nguồn từ một nguồn như ***IQueryable (T).*** Khi bạn xem một câu truy vấn có biến là kiểu ***IEnumerable<Customer>,*** nó đơn giản là các thức truy vấn, khi nó được thực hiện, sẽ tạo ra một trình tự không có gì hoặc nhiều đối tượng ***Customer***.

```
IEnumerable<Customer> customerQuery =from cust in customers
  where cust.City == "London"   select cust;
foreach (Customer customer in customerQuery)
{
 Console.WriteLine(customer.LastName + ", " + customer.FirstName);
}
```

V.2 Cho phép chương trình biên dịch xử lý các loại khai báo chung

Nếu bạn thích, bạn có thể tránh cú pháp chung chung bằng cách sử dụng từ khóa **var**. Các từ khóa **var** để hướng dẫn trình biên dịch nhận ra loại biến một truy vấn tìm kiếm tại các nguồn dữ liệu được xác định trong mệnh đề **from**. Ví dụ sau cho cùng một kết quả như đoạn mã được xây dựng phía trên.

```
var customerQuery2 =
    from cust in customers
    where cust.City == "London"
    select cust;
```

```
foreach(var customer in customerQuery2)
{
    Console.WriteLine(customer.LastName + ", " + customer.FirstName);
}
```

Các từ khóa **var** rất hữu ích khi các loại của biến là rõ ràng hoặc khi nó không phải là điều quan trọng để xác định rõ ràng các loại chung như là cái đó được tạo ra bằng cách nhóm các truy vấn. Nói chung, chúng tôi đề nghị rằng nếu bạn sử dụng var, nhận thấy rằng nó có thể làm cho mã của bạn khó khăn hơn cho những người khác đọc.

V.3 Hoạt động truy vấn cơ bản.

Chủ đề này cho một giới thiệu tóm tắt về truy vấn LINQ và một số biểu hiện của các loại hình hoạt động điển hình mà bạn thực hiện trong một truy vấn.

Chú ý: Nếu bạn đã là quen thuộc với một truy vấn ngôn ngữ như SQL hay XQuery, bạn có thể bỏ qua hầu hết các chủ đề này. Đọc về "mệnh đề **from**" trong phần kế tiếp để tìm hiểu về trật tự của các mệnh đề trong biểu thức truy vấn LINQ.

V.3.1 Obtaining a Data Source

Lấy vật là một nguồn dữ liệu

Trong một truy vấn LINQ, bước đầu tiên là xác định nguồn dữ liệu. Trong C # cũng như trong hầu hết các ngôn ngữ lập trình một biến phải được khai báo trước khi nó có thể được sử dụng. Trong một truy vấn LINQ, mệnh đề from đứng đầu tiên để giới thiệu các nguồn dữ liệu (***customer***) và nhiều biến (***cust***).

```
//queryAllCustomers is an IEnumerable<Customer>
var queryAllCustomers = from cust in customers
                        select cust;
```

Phạm vi biến giống như các biến lặp trong một vòng lặp foreach, ngoại trừ không có thực sự một vòng lặp xảy ra trong một biểu thức truy vấn. Khi truy vấn được thi hành, trong phạm vi biến sẽ phục vụ như là một tham chiếu lần lượt đến các các phần tử trong mỗi **customers**. Bởi vì trình biên dịch có thể nhận ra các kiểu khác nhau của **cuts**, bạn không thể xác định nó rõ ràng. Bổ sung phạm vi các biến có thể được giới thiệu bởi một mệnh đề **let.**

V.3.2 Filtering(Lọc)

Có lẽ các hành đọng truy vấn phổ biến nhất là một bộ lọc để áp dụng trong các mẫu của một biểu thức logic Boolean. Các bộ lọc giúp các truy vấn trả về duy nhất các phần tử cho các biểu thức là đúng. Kết quả là kết quả được sử dụng mệnh đề **where**. Các bộ lọc có hiệu lực xác định các yếu tố đó để loại trừ từ các nguồn liên tục. Trong ví dụ sau, chỉ những khách hàng có địa chỉ ở London sẽ được trả về.

```
var queryLondonCustomers = from cust in customers
                           where cust.City == "London"
                           select cust;
```

Bạn có thể sử dụng quen ngôn ngữ C# với các biểu thức logic AND và OR để vận hành áp dụng như nhiều bộ lọc trong mệnh đề **where**. Ví dụ, chỉ trả về các khách hàng có địa chỉ tại"London" và có tên là "Devon" bạn sẽ viết đoạn mã sau đây:

```
where cust.City=="London" && cust.Name == "Devon"
```

Để trả về khách hàng có địa chỉ ở London hay Paris, bạn sẽ viết mã sau:

```
where cust.City == "London" || cust.City == "Paris"
```

V.3.3 Ordering (Thứ tự)

Thường nó là thuận tiện để phân loại dữ liệu trả về. Mệnh đề **orderby** sẽ gây ra các phần tử trong chuỗi trả về để được sắp xếp theo mặc định so sánh cho các loại đang được sắp xếp. Ví dụ, sau đây truy vấn có thể được mở rộng để phân loại các kết quả dựa trên thuộc tính Name. Bởi vì thuộc tính ***Name*** là một chuỗi, mặc định nó sẽ so sánh và thực hiện sắp xếp theo thứ tự chữ cái từ A đến Z.

```
var queryLondonCustomers3 =
    from cust in customers
    where cust.City == "London"
    orderby cust.Name ascending
    select cust;
```

Để các kết quả trong thứ tự đảo ngược trật tự, từ A to Z, sử dụng mệnh đề **orderby** ...**descending**.

V.3.4 Grouping

Mệnh đề **group** cho phép bạn nhóm các kết quả của bạn dựa trên một khóa mà bạn chỉ định. Ví dụ, bạn có thể xác định rằng các kết quả cần được nhóm lại theo thuộc tính

City để tất cả các khách hàng từ London, Paris hay cá nhân đang có trong nhóm. Trong trường hợp này, ***cust.City*** chính là khóa.

Chú ý: Các kiểu rõ ràng đang có trong các ví dụ để minh họa các khái niệm. Bạn cũng có thể sử dụng chính cho ***custQuery***, ***group***,cad ***customer*** để cho trình biên dịch xác định chính xác loại.

```
// queryCustomersByCity is an IEnumerable<IGrouping<string, Customer>>
  var queryCustomersByCity =
      from cust in customers
      group cust by cust.City;

  // customerGroup is an IGrouping<string, Customer>
  foreach (var customerGroup in queryCustomersByCity)
  {
      Console.WriteLine(customerGroup.Key);
      foreach (Customer customer in customerGroup)
      {
          Console.WriteLine("    {0}", customer.Name);
      }
  }
```

Khi bạn kết thúc một truy vấn với mệnh đề **group**, các kết quả của bạn sẽ sẽ được trả về một danh sách từ các danh sách. Mỗi phần tử trong danh sách là một đối tượng có một *Key* thành viên và danh sách các phần tử đó là nhóm chứa khóa này. Khi bạn lặp qua một truy vấn mà kết quả là một nhóm có trình tự, bạn cần phải sử dụng một vòng lặp foreach. Nếu bạn cần phải tham khảo các kết quả thi hành của một nhóm, bạn có thể sử dụng từ khóa **into** để tạo ra một định danh có thể được thêm vào câu truy vấn. Dưới đây là những truy vấn trả lại chỉ những nhóm có chứa nhiều hơn hai khách hàng:

```
var custQuery =
    from cust in customers
    group cust by cust.City into custGroup
    where custGroup.Count() > 2
    orderby custGroup.Key
    select custGroup;
```

V.3.5 Joining

Thao tác **Join** tạo ra sự kết hợp giữa nhiều sự kiện không được rõ ràng mô trong nguồn dữ liệu. Ví dụ, bạn có thể thực hiện một thao tác để tìm tất cả các khách hàng ở London những người đặt hàng các sản phẩm từ nhà cung cấp, họ đang ở Paris. Trong LINQ mệnh đề **Join** luôn luôn tham gia các hoạt động dựa trên tập đối tượng thay vì các bảng cơ sở dữ liệu. Trong LINQ bạn không sử dụng mệnh đề **Join** thường xuyên như

bạn làm trong SQL bởi vì các khóa ngoại LINQ miêu tả trong mô hình như là thuộc tính nắm giữ một tập các mục. Ví dụ, một đối tượng ***Customer*** có chứa một tập **Order** của các đối tượng. Đúng hơn là biểu diễn một thao tác, bạn truy cập các thứ tự bằng cách sử dụng dấu chấm:

```
from order in Customer.Orders...
```

V.3.6 Selecting (Projections)

Mệnh đề **Select** đưa ra các kết quả trả về của một câu truy vấn và xác định "hình dạng" hoặc kiểu của mỗi kết quả trả về. Ví dụ, bạn có thể chỉ định cho dù kết quả của bạn sẽ bao gồm tất cả các đối tượng ***Customer***, chỉ cần một thành viên, một nhóm của các thành viên, hoặc một số kết quả loại hoàn toàn khác nhau dựa trên tính toán hay một đối tượng mới tạo ra. Khi mệnh đề **Select** đưa ra một cái gì đó khác là một bản sao của các phần tử nguồn, thao tác được gọi là bản dự thảo. Việc sử dụng các bản dự thảo để chuyển đổi dữ liệu là một khả năng của biểu thức truy vấn LINQ.

V.4 Chuyển đổi dữ liệu với LINQ

Ngôn ngữ-Integrated Query (LINQ) không phải là chỉ có truy lại dữ liệu. Nó cũng là một công cụ mạnh mẽ cho việc chuyển dữ liệu. Bằng cách sử dụng một truy vấn LINQ, bạn có thể sử dụng một chuỗi nguồn dữ liệu vào và sửa đổi nó trong nhiều cách để tạo ra một chuỗi ra mới. Bạn có thể sửa đổi trình tự bản thân nó mà không sửa đổi các phần tử bằng cách phân loại và gom nhóm. Nhưng có lẽ trong hầu hết các tính năng mạnh mẽ của các câu truy vấn LINQ là khả năng tạo loại mới. Đây là hoàn hảo trong mệnh đề **select**. Ví dụ, bạn có thể thực hiện các nhiệm vụ sau:

- Hợp nhất nhiều dãy đầu vào thành một dãy đầu ra đơn lẻ để có một loại mới.
- Tạo ra dãy các phần tử bao gồm chỉ một hoặc một vài thuộc tính của mỗi phần tử.
- Tạo ra dãy các phần tử bao gồm các kết quả của sự thi hành trên các nguồn dữ liệu.
- Tạo ra dãy trong một định dạng khác nhau. Ví dụ, bạn có thể chuyển đổi những hàng dữ liệu từ SQL hoặc văn bản vào file XML.

Đây chỉ là một vài ví dụ. Tất nhiên, những sự chuyển đổi có thể được kết hợp theo cách khác nhau trong cùng một truy vấn. Hơn nữa, trình tự ra của một chuỗi truy vấn này có thể được sử dụng như là yếu tố đầu vào cho một chuỗi truy vấn mới.

V.4.1 Tham gia vào nhiều yếu tố đầu vào xuất ra một trình tự.

Bạn có thể sử dụng một truy vấn LINQ để tạo ra một trình tự đầu ra có chứa các phần tử từ nhiều hơn một trình tự đầu vào. Ví dụ sau cho thấy làm thế nào để kết hợp hai cấu trúc dữ liệu trong bộ nhớ, nhưng cùng một nguyên tắc có thể được áp dụng để kết hợp các nguồn dữ liệu từ XML hoặc SQL hoặc DataSet. Ví dự sau cho thấy điều đó:

```
class Student
{
    public string First { get; set; }
    public string Last {get; set;}
    public int ID { get; set; }
    public string Street { get; set; }
    public string City { get; set; }
    public List<int> Scores;
}

class Teacher
{
    public string First { get; set; }
    public string Last { get; set; }
    public int ID { get; set; }
    public string City { get; set; }
}
```

Ví dụ sau cho thấy một câu truy vấn:

```
class DataTransformations
{
    static void Main()
    {
        // Create the first data source.
        List<Student> students = new List<Student>()
        {
            new Student {First="Svetlana",
                Last="Omelchenko",
                ID=111,
                Street="123 Main Street",
                City="Seattle",
                Scores= new List<int> {97, 92, 81, 60}},
            new Student {First="Claire",
                Last="O'Donnell",
                ID=112,
                Street="124 Main Street",
                City="Redmond",
                Scores= new List<int> {75, 84, 91, 39}},
            new Student {First="Sven",
                Last="Mortensen",
                ID=113,
                Street="125 Main Street",
```

```
                City="Lake City",
                Scores= new List<int> {88, 94, 65, 91}},
        };

        // Create the second data source.
        List<Teacher> teachers = new List<Teacher>()
        {
            new Teacher {First="Ann", Last="Beebe", ID=945, City = "Seattle"},
            new Teacher {First="Alex", Last="Robinson", ID=956, City = "Redmond"},
            new Teacher {First="Michiyo", Last="Sato", ID=972, City = "Tacoma"}
        };

        // Create the query.
        var peopleInSeattle = (from student in students
                    where student.City == "Seattle"
                    select student.Last)
                    .Concat(from teacher in teachers
                            where teacher.City == "Seattle"
                            select teacher.Last);

        Console.WriteLine("The following students and teachers live in Seattle:");
        // Execute the query.
        foreach (var person in peopleInSeattle)
        {
            Console.WriteLine(person);
        }

        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
/* Output:
    The following students and teachers live in Seattle:
    Omelchenko
    Beebe
 */
```

V.4.2 Lựa chọn một tập hợp con của mỗi phần tử nguồn

Có hai cách chính để lựa chọn một nhóm của mỗi phần tử trong chuỗi nguồn:

1. Để chọn chỉ cần một thành viên của các phần tử nguồn nguyên tố, sử dụng thao tác chấm. Trong ví dụ sau, giả định rằng một đối tượng ***Customer*** có chứa một số thuộc tính public bao gồm một chuỗi có tên ***City***. Khi thực hiện, truy vấn này sẽ cho ra một trình tự các chuỗi đầu ra.

```
var query = from cust in Customers      select cust.City;
```

3. Để tạo các phần tử có chứa nhiều hơn một thuộc tính các phần tử nguồn, bạn có thể sử dụng một đối tượng với một đối tượng có tên hoặc một loại vô danh. Ví dụ sau cho thấy việc sử dụng một ẩn danh để đóng gói hai loại thuộc tính từ mỗi phần tử C***ustomer***:

```
        var query = from cust in Customer
                    select new {Name = cust.Name, City = cust.City};
```

V.4.3 Chuyển đổi các đối tượng trong bộ nhớ vào XML

Các câu truy vấn LINQ làm cho nó dễ dàng chuyển hóa dữ liệu giữa câu trúc dữ liệu trong bộ nhớ, cơ sở dữ liệu SQL, ADO.NET Datasets và luồng XML, hoặc các tài liệu. Ví dụ sau cho thấy việc chuyển đổi dữ liệu trong bộ nhớ vào các phần tử trong XML:

```
class XMLTransform
{
    static void Main()
    {
        // Create the data source by using a collection initializer.
        List<Student> students = new List<Student>()
        {
            new Student {First="Svetlana", Last="Omelchenko", ID=111, Scores = new
List<int>{97, 92, 81, 60}},
            new Student {First="Claire", Last="O'Donnell", ID=112, Scores = new
List<int>{75, 84, 91, 39}},
            new Student {First="Sven", Last="Mortensen", ID=113, Scores = new
List<int>{88, 94, 65, 91}},
        };

        // Create the query.
        var studentsToXML = new XElement("Root",
            from student in students
            let x = String.Format("{0},{1},{2},{3}", student.Scores[0],
                    student.Scores[1], student.Scores[2], student.Scores[3])
            select new XElement("student",
                       new XElement("First", student.First),
                       new XElement("Last", student.Last),
                       new XElement("Scores", x)
                    ) // end "student"
                ); // end "Root"

        // Execute the query.
        Console.WriteLine(studentsToXML);

        // Keep the console open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
```

Ta có kết quả là:

```
< Root>
  <student>
    <First>Svetlana</First>
    <Last>Omelchenko</Last>
    <Scores>97,92,81,60</Scores>
  </student>
  <student>
    <First>Claire</First>
    <Last>O'Donnell</Last>
```

```
    <Scores>75,84,91,39</Scores>
  </student>
  <student>
    <First>Sven</First>
    <Last>Mortensen</Last>
    <Scores>88,94,65,91</Scores>
  </student>
</Root>
```

V.4.4 Thực hiện các hoạt động trên các phần tử nguồn.

Một trình tự đầu ra có thể không chứa bất kỳ những phần tử hoặc thuộc tính của phần tử từ trình tự nguồn. Đầu ra phải đượcthay thế được bởi chuỗi các giá trị được tính bằng cách sử dụng các phần tử nguồn như đối số đầu vào. Dưới đây là những truy vấn đơn giản, khi nó được thực hiện, kết quả đầu ra của một trình tự những chuỗi có giá trị đại diện cho một tính toán dựa trên các nguồn trình tự các phần tử thuộc kiểu **double**.

Chú ý: Việc gọi các phương thức trong các biểu thức truy vấn không được hỗ trợ nếu truy vấn sẽ được dịch sang một tên miền khác. Ví dụ, bạn không thể gọi phương thức C # thông thường trong LINQ to SQL vì SQL Server không có ngữ cảnh cho nó. Tuy nhiên, bạn có thể ánh xạ thành các thủ tục trong SQL và gọi các phương thức đó.

```
class FormatQuery
{
    static void Main()
    {
        // Data source.
        double[] radii = { 1, 2, 3 };

        // Query.
        IEnumerable<string> query =
            from rad in radii
            select String.Format("Area = {0}", (rad * rad) * 3.14);

        // Query execution.
        foreach (string s in query)
            Console.WriteLine(s);

        // Keep the console open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}
/* Output:
    Area = 3.14
    Area = 12.56
    Area = 28.26
*/
```

V.4.5 Loại các quan hệ trong thao tác truy vấn.

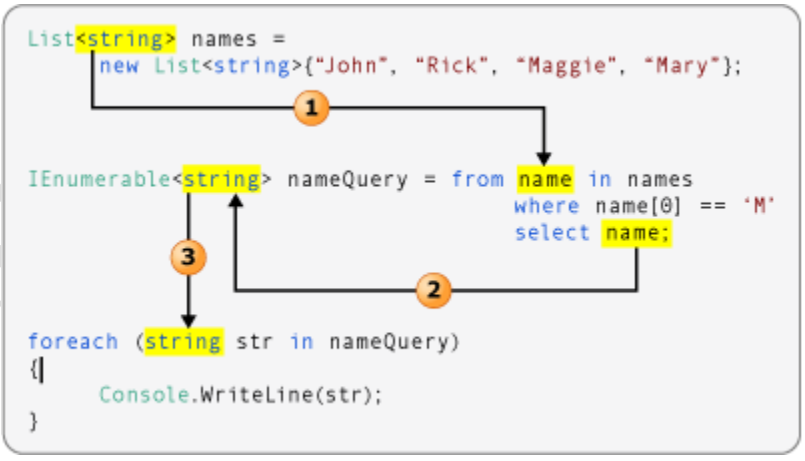
Để viết truy vấn có hiệu quả, bạn nên hiểu loại của các biến trong một truy vấn như thế nào để hoàn tất tất cả các thao tác liên quan đến nhau. Nếu bạn hiểu được những mối quan hệ, bạn sẽ dễ dàng hơn để lĩnh hội các ví dụ LINQ và đoạn code ví dụ trong tài liệu hướng dẫn. Hơn nữa, bạn sẽ hiểu những gì xảy ra đằng sau những hiện trường khi các biến được hoàn toàn phân loại cách sử dụng từ khóa **var**.

Thao tác truy vấn LINQ được phân loại rõ ràng trong nguồn dữ liệu, trong chính câu truy vấn, và trong thực thi truy vấn. Các loại của các biến trong truy vấn phải tương thích với các phần tử trong dữ liệu nguồn và với các loại của biến lặp trong câu lệnh **foreach**. Điều này đảm bảo rằng các loại lỗi đều bị bắt lại tại thời điểm biên dịch khi đó người ta có thể sửa lỗi đó trước khi nó được đưa vào làm ứng dụng.

Để giải thích các loại các mối quan hệ, hầu hết các ví dụ mà làm được sử dụng kiểu rõ ràng cho tất cả các biến. Cuối cùng ví dụ cho thấy như thế nào cùng áp dụng một nguyên tắc ngay cả khi bạn sử dụng từ khóa **var**.

V.5.6 Truy vấn mà không chuyển hóa các nguồn dữ liệu

Thí dụ minh họa sau đây cho thấy một câu truy vấn LINQ tới các đối tượng để hoạt động mà không thực hiện chuyển đổi trên dữ liệu. Nguồn chứa một trình tự của những chuỗi và giá trị đầu ra là một trình tự các chuỗi.

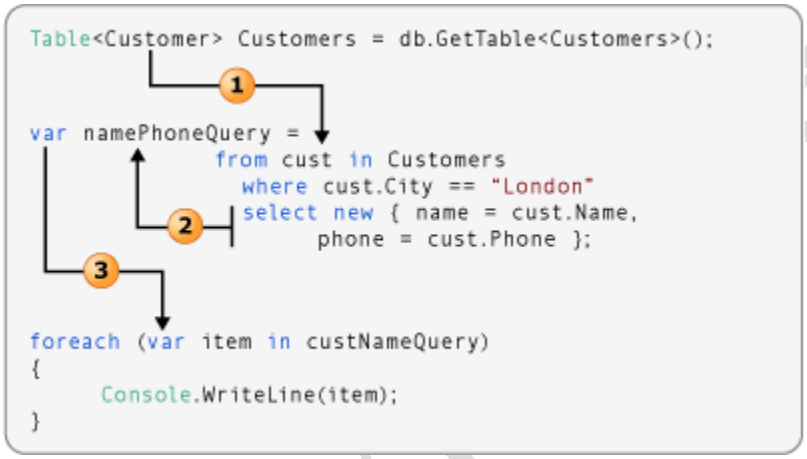


1. Các loại đối số của các nguồn dữ liệu xác định rõ loại miền của biến.

2. Câu lệnh **select** sẽ trả về thuộc tính ***Name*** thay vì hoàn thành đối tượng **Customer**. Bởi vì ***Name*** vì là một chuỗi, các kiểu đối số của ***custNameQuery*** là chuỗi, không phải **Customer**.

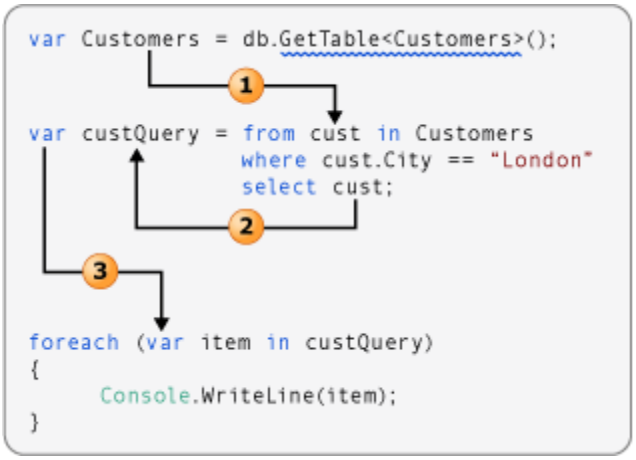
3. Bởi vì ***custNameQuery*** là một trình tự các chuỗi, biến của vòng lặp foreach cũng phải là một chuỗi.

Thí dụ sau đây cho thấy một sự chuyển đổi hơi phức tạp hơn. Câu lệnh **select** trả về một kiểu nặc danh loại mà chỉ cần lưu giữ được hai thành viên của các đối tượng **Customer** gốc.



V.5.7 Trình biên dịch phải suy luận ra các loại thông tin

Mặc dù bạn nên tìm hiểu những loại các mối quan hệ trong một hoạt động truy vấn, bạn không có tuỳ chọn để cho phép trình biên dịch làm tất cả các công việc cho bạn. Các từ khóa **var** có thể được sử dụng cho bất kỳ biến cục bộ nào trong một thao tác truy vấn. động. Thí dụ sau đây là ví dụ chính xác tương đương với ví dụ số 2 đã được thảo luận phía trên. Sự khác nhau duy nhất là trình biên dịch sẽ được cung cấp kiểu rõ ràng cho mỗi biến trong hoạt động truy vấn:



V.6 Cú pháp truy vấn vs cú pháp phương thức.

Nói chung, chúng tôi khuyên bạn nên sử dụng cú pháp truy vấn vì nó thường là đơn giản hơn và hay hơn; tuy nhiên ở đây không có sự khác biệt giữa cú pháp truy vấn và cú pháp phương thứ. Ngoài ra, một số truy vấn, chẳng hạn như việc truy lục các phần tử phù hợp với một điều kiện xác định, hầu hết các truy vấn trong LINQ giới thiệu trong tài liệu hướng dẫn được viết thành văn bản là biểu thức truy vấn bằng cách sử dụng cú pháp truy vấn có tính tuyên bố được giới thiệu trong C # 3.0. Tuy nhiên,.NET runtime ngôn ngữ chung(CRL_Common Language Runtime) không có ý niệm của cú pháp truy vấn trong chính nó. Vì vậy, tại thời điểm biên dịch, biểu thức truy vấn là thông dịch gì đó mà CRL, không hiểu: gọi các phương thức. Các phương thức này được gọi là toán tử truy vấn chuẩn, và chúng có các tên như **Where**, **Select**, **GroupBy**, **Join**, **Max**, **Average** và như vậy trên. Bạn có thể gọi chúng trực tiếp bằng cách sử dụng cú pháp phương thức các cú pháp truy vấn. Truy lục các phần tử có giá trị tối đa trong một mã nguồn trình tự, chỉ có thể được thể hiện như các lần gọi phương thức. Các tài liệu tham khảo cho biểu thức truy vấn chuẩn trong namespace System.Linq bởi hầu hết mọi người sử dụng cú pháp phương thức. Vì vậy, ngay cả khi bắt đầu viết truy vấn LINQ, nó rất hữu ích để làm quen với cách sử dụng cú pháp phương thức trong truy vấn và toán tử thức truy vấn.

V.6.1 Toán tử truy vấn chuẩn mở rộng các phương thức

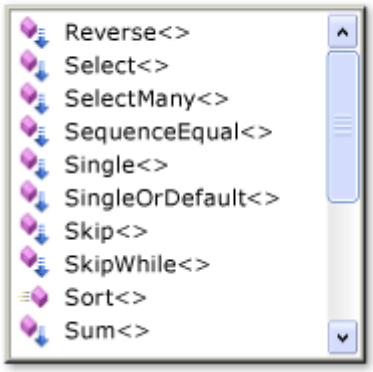
Ví dụ sau cho thấy một cách dễ dàng biểu thức truy vấn và các ngữ nghĩa tương đương truy vấn được viết như là một phương thức dựa trên truy vấn.

```
class QueryVMethodSyntax
{
    static void Main()
    {
        int[] numbers = { 5, 10, 8, 3, 6, 12};

        //Query syntax:
        IEnumerable<int> numQuery1 =
            from num in numbers
            where num % 2 == 0
            orderby num
            select num;

        //Method syntax:
        IEnumerable<int> numQuery2 = numbers.Where(num => num % 2 == 0).OrderBy(n =>
n);

        foreach (int i in numQuery1)
        {
            Console.Write(i + " ");
        }
        Console.WriteLine(System.Environment.NewLine);
        foreach (int i in numQuery2)
        {
            Console.Write(i + " ");
        }

        // Keep the console open in debug mode.
        Console.WriteLine(System.Environment.NewLine);
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}
/*
    Output:
    6 8 10 12
    6 8 10 12
 */
```

Đầu ta từ hai ví dụ là giống hệt nhau. Bạn có thể thấy rằng các loại của biến truy vấn biến là cùng một trong hai hình thức: IEnumerable (T).

Để tìm hiểu phương thức dựa trên câu truy vấn, hãy kiểm tra nó chặt chẽ hơn. Ở bên phải của biểu thức, chú ý rằng chú ý rằng mệnh đề where giờ đây không còn được thể hiện như là một phương thức dụ trên đối tượng numbers, mà là bạn sẽ gọi lại một loại kiểu IEnumerable<int> . Nếu bạn là quen thuộc với giao diện IEnumerable (T), bạn biết rằng nó không có một phương thức where. Tuy nhiên, nếu bạn gọi trình hỗ trợ thông minh hoàn thành danh sách trong Visual Studio IDE, bạn sẽ thấy không chỉ là một phương thức where, nhưng nhiều phương thức khác như: select, SelectMany, Join, và Orderby. Đây là tất cả các toán tử truy vấn chuẩn.

```
List<string> list = new List<string();
list.
```



Mặc dù nó có vẻ như nếu giao diện IEnumerable (T) đã được định nghĩa lại để bổ xung các phương thức này, trên thực tế cái này không phải là cách. Các toán tử truy vấn chuẩn được thực hiện như là một loại phương thức mới được gọi là các phương thức mở rộng. Các phương thức mở rộng " extend " một loại hiện có; chúng có thể được gọi là nếu chúng đã được thể hiện các phương thức dựa trên kiểu. Toán tử truy vấn chuẩn mở rộng interface IEnumerable (T) và đó là lý do tại sao, bạn có thể viết numbers.Where (...).

Để bắt đầu sử dụng LINQ, bạn phải chắc chắn rằng bạn thực sự hiểu tất cả về các phương thức mở rộng là làm thế nào để đem chúng vào trong phạm vi ứng dụng của bạn bằng cách sử dụng đúng hướng dẫn. Điều này được giải thích thêm trong phần làm thế nào để: Tạo một dự án LINQ.

V.6.2 Biểu thức Lambda

Trong ví dụ trước, chú ý rằng các biểu thức điều kiện (***num% 2 == 0***) là thông qua như là một trong những đối số của phương thức **where**: **Where (num => num% 2 == 0)**. Biểu thức trong ngoặc được được gọi là biểu thức lambda. Đó là một phương thức nặc danh có thể chứa đựng những biểu thức và các phát biểu và có thể sử dụng để tạo một ủy nhiệm chung hoặc một biểu thức cây. Trong C # => là toán tử lambda, được đọc như "goes to". Các ***num*** bên trái của các toán tử là yếu tố đầu vào biến đó tương ứng với ***num*** trong biểu thức truy vấn. Trình biên dịch có thể nhận kiểu ***num*** bởi vì nó biết rằng ***numbers*** là một kiểu chung ***IEnumerable (T)***. Toán tử lambda chỉ giống như các biểu thức trong cú pháp truy vấn hay trong biểu thức C # hoặc câu lệnh; nó có thể bao gồm cả

các lần gọi phương thức và các phương thức logic phức tạp. The "trả về giá trị" chỉ là các biểu thức kết quả.

Để bắt đầu sử dụng LINQ, bạn không sử dụng lambdas nhiều. Tuy nhiên, một số truy vấn chỉ có thể được thể hiện trong cú pháp phương thức và một số yêu cầu của những biểu thức lambda. Sau khi bạn trở nên quen thuộc với lambdas, bạn sẽ thấy rằng nó là một công cụ mạnh mẽ và linh hoạt trong LINQ của bạn.

Trong doạn mã ví dụ trước, không phải phương thức ***OrderBy*** là dẫn chứng bằng cách sử dụng dấu chấm để gọi tới **where**. **Where** đưa ra trình tự lọc, và sau đó **Orderby** có tác dụng sắp xếp trình tự đó. Bởi vì các truy vấn trở trả về một **IEnumerable**, bạn soạn chúng trong cú pháp phương thức xích các lần gọi phương thức lại với nhau. Đây là những gì trình biên dịch làm ở dăng sau hiện trường khi bạn viết truy vấn bằng cách sử dụng cú pháp truy vấn. Và bởi vì một truy vấn biến không lưu trữ các kết quả của câu truy vấn, bạn có thể thay đổi nó hay sử dụng nó như là cơ sở cho một truy vấn mới bất kỳ lúc nào, ngay cả sau khi nó đã được thực hiện.

V.7 Các đặc trưng được LINQ hỗ trợ trong C#3.0

Dưới đây là những phần giới thiệu những ngôn ngữ mới xây dựng trong C # 3.0. Mặc dù các tính năng mới tất cả đều được sử dụng đến một mức độ với các truy vấn LINQ, chúng không giới hạn đối với LINQ và có thể được sử dụng trong bối cảnh bất cứ nơi bạn tìm thấy chúng hữu ích.

V.7.1 Biểu thức truy vấn.

Biểu thức truy vấn sử dụng cú pháo khai báo tương tự với SQL hay XQuery để truy vấn trên tập hợp ***IEnumerable***. Tại thời điểm biên dịch cú pháp truy vấn là phương thức gọi đến một nhà cung cấp LINQ của việc triển khai thực hiện của toán tử truy vấn chuẩn các phương thức mở rộng. Ứng dụng kiểm soát các toán tử biểu thức truy vấn chuẩn trong namespace thích hợp với một chỉ dẫn **using**. Dưới đây là một biểu thức truy vấn chuẩn làm cho một mảng các chuỗi, nhóm chúng theo ký tự đầu tiên trong chuỗi, và phân loại chúng thành một nhóm.

```
var query = from str in stringArray
```

```
            group str by str[0] into stringGroup
            orderby stringGroup.Key
               select stringGroup;
```

V.7.2 Implicitly Typed Variables (var)

Thay vì xác định rõ ràng một loại khi bạn khai báo và khởi tạo một biến, bạn có thể sử dụng từ khóa **var** để chỉ thị cho trình biên dịch nhận ra và gán kiểu, như được hiển thị ở đây:

```
var number = 5;
var name = "Virginia";
var query = from str in stringArray
            where str[0] == 'm'
            select str;
```

Các biến được khai báo là **var** là chỉ cần kiểu rõ ràng như các biến có loại bạn chỉ định rõ ràng. Việc sử dụng các **var** làm cho nó có thể tạo ra loại vô danh, nhưng nó có thể được sử dụng cho bất kỳ biến cục bộ nào.

V.7.3 Đối tượng, và tập hợp các giá trị đầu vào

Đối tượng, và tập hợp các giá trị đầu vào làm cho nó có thể khởi tạo các đối tượng mà không rõ ràng gọi một constructor cho đối tượng. Các giá trị đầu vào thường được sử dụng trong truy vấn khi chúng biểu hiện cho nguồn dữ liệu của dự án vào một kiểu dữ liệu mới. Giả định rằng một lớp có tên là Customer với hai thuộc tính ***Name*** và ***Phone*** được khai báo với từ khóa **public**, các đối tượng giá trị đầu vào có thể được sử dụng như trong các mã sau đây:

```
Customer cust = new Customer {Name = "Mike" ; Phone ={ "555-1212 "};
```

V.7.4 Các loại chưa xác định

Kiểu ẩn danh là một kiểu xây dựng bởi trình biên dịch và loại tên là biến duy nhất cho trình biên dịch. Các loại chưa xác định cung cấp một sự tiện lợi để thiết lập một hợp nhóm các thuộc tính tạm thời trong một kết quả truy vấn mà không có để xác định một loại tên riêng. Các loại chưa xác định được khởi động với một biểu thức mới và một đối tượng giá trị đầu vào, như được hiển thị ở đây:

```
select new {name = cust.Name, phone = cust.Phone};
```

V.7.5 Các phương thức mở rộng

Các phương thức mở rộng là một phương thức tĩnh có thể được kết hợp với một loại, để nó có thể được gọi như một loại thể hiện của phương thức. Tính năng này cho phép bạn, có thể, "add" them một phương thức có sẵn mà bạn không phải chỉnh sửa nó. Các toán tử truy vấn chuẩn là một tạp các phương thức mở rộng cung cấp LINQ tính năng truy vấn cho bất kì kiểu nào thục thi interface IEnumerable (T).

V.7.6 Các thuộc tính tự động thi hành

Các tính tự động thi hành làm cho việc khai báo thuộc tính ngắn gọn hơn. Khi bạn khai báo một thuộc tính như được hiển thị trong ví dụ sau, trình biên dịch sẽ tạo ra một trường ẩn không cho phép truy cập, ngoại trừ thông qua các thuộc tính **get** và **set**.

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

V.8 Viết câu truy vấn trong C#

Phần này sẽ hướng dẫn bạn thông qua việc sử dụng các tính năng mới của C# 3.0 và hiển thị chúng như thế nào để viết biểu thức truy vấn LINQ. Sau khi hoàn tất phần này bạn sẽ sẵn sàng để chuyển sang các ví dụ mẫu và tài liệu hướng dẫn cụ thể cho các chủ đề mà LINQ cung cấp, chẳng hạn như LINQ to SQL, LINQ to Datasets, hoặc LINQ to XML.

Các nguồn dữ liệu cho các truy vấn đơn giản là một danh sách các đối tượng Student. Mỗi mẩu tin Student có các thuộc tính là firstName và lastname, và một mảng các số nguyên đại diện cho điểm kiểm tra trong một lớp. Sao chép đoạn mã này vào dự án của bạn.

V.8.1 Để thêm các dữ liệu nguồn

Thêm vào lớp ***Student*** và khởi tạo danh sách sinh viên tới lớp ***Program*** trong dự án của bạn.

```
public class Student
{
    public string First { get; set; }
    public string Last { get; set; }
    public int ID { get; set; }
    public List<int> Scores;
}
```

```
// Create a data source by using a collection initializer.
static List<Student> students = new List<Student>
{
   new Student {First="Svetlana", Last="Omelchenko", ID=111, Scores= new List<int>
{97, 92, 81, 60}},
   new Student {First="Claire", Last="O'Donnell", ID=112, Scores= new List<int> {75,
84, 91, 39}},
   new Student {First="Sven", Last="Mortensen", ID=113, Scores= new List<int> {88, 94,
65, 91}},
   new Student {First="Cesar", Last="Garcia", ID=114, Scores= new List<int> {97, 89,
85, 82}},
   new Student {First="Debra", Last="Garcia", ID=115, Scores= new List<int> {35, 72,
91, 70}},
   new Student {First="Fadi", Last="Fakhouri", ID=116, Scores= new List<int> {99, 86,
90, 94}},
   new Student {First="Hanying", Last="Feng", ID=117, Scores= new List<int> {93, 92,
80, 87}},
   new Student {First="Hugo", Last="Garcia", ID=118, Scores= new List<int> {92, 90,
83, 78}},
   new Student {First="Lance", Last="Tucker", ID=119, Scores= new List<int> {68, 79,
88, 92}},
   new Student {First="Terry", Last="Adams", ID=120, Scores= new List<int> {99, 82,
81, 79}},
   new Student {First="Eugene", Last="Zabokritski", ID=121, Scores= new List<int> {96,
85, 91, 60}},
   new Student {First="Michael", Last="Tucker", ID=122, Scores= new List<int> {94, 92,
91, 91} }
};
```

Thêm một Student mới vào danh sách Students và sử dụng một tên và điểm kiểm tra cho lựa chon của bạn. Hãy thử gõ tất cả các thông tin sinh viên mới để tìm hiểu các cú pháp tốt hơn cho các đối tượng.

V.9 Tạo các truy vấn

V.9.1 Để tạo một truy vấn đơn giản

Trong phương thúc Main của ứng dụng, tạo một truy vấn đơn giản, khi nó được thực hiện, sẽ xuất ra một danh sách của tất cả các sinh viên có điểm dầu tiên trong mảng điểm kiểm tra trên 90. Lưu ý rằng bởi vì toàn bộ đối tượng Student được chọn, các kiểu truy vấn là ***IEnumerable<Student>***. Mặc dù đoạn mã cũng có thể sử dụng đúng kiểu bằng cách sử dụng từ khóa **var**, kiểu rõ ràng được sử dụng để minh họa rõ ràng kết quả. Cũng lưu ý rằng các truy vấn nhiều biến, ***student***, phục vụ như một tham chiếu cho mỗi **Studnet** trong các nguồn, cung cấp cho các thành viên truy cập mỗi đối tượng.

```
// Create the query.
// studentQuery is an IEnumerable<Student>
var studentQuery =
    from student in students
    where student.Scores[0] > 90
    select student;
```

V.9.2 Để thực hiện các truy vấn

1. Bây giờ viết vòng lặp ***foreach*** sẽ thực hiện truy vấn. Lưu ý sau đây về các mã:

• Mỗi phần tử trong chuỗi được trả về là được truy cập thông qua các biến lặp trong vòng lặp ***foreach***.

• Các kiểu biến này là student, và các kiểu biến truy vấn biến là tương thích, IEnumerable<Student>.

2. Sau khi bạn đã được thêm vào mã này, xây dựng và chạy các ứng dụng bằng cách nhấn Ctrl + F5 để xem các kết quả trong cửa sổ Console.

```
// Execute the query.
// var could be used here also.
foreach (Student student in studentQuery)
{
    Console.WriteLine("{0}, {1}", student.Last, student.First);
}
```

V.9.3 Để thêm một điều kiện lọc

Bạn có thể kết hợp nhiều điều kiện logic trong mệnh đề **where** để tinh chỉnh thêm một truy vấn. Đoạn mã sau đây cho biết thêm một điều kiện đó, để truy vấn trả về những sinh viên có điểm số đầu tiên trên 90 và có điểm cuối cùng ít hơn 80.

```
where student.Scores[0] > 90 && student.Scores[3] < 80
```

V.9.4 Chỉnh sửa truy vấn

Nó sẽ được dễ dàng hơn để quét các kết quả nếu chúng có trong một số nhóm có đặc tính giống nhau. Bạn có thể sắp xếp theo trình tự các kết quả đã được trả về bằng cách sử dụng bất kỳ thuộc tính nào trong các phần tử nguồn. Ví dụ, sau đây mệnh đề **orderby** sắp xếp các kết quả trả về theo trật tự từ A tới Z theo tên của mỗi sinh viên. Thêm vào mệnh đề sau đây **orderby** vào câu truy vấn của bạn, ngay sau câu lệnh **where** và trước câu lệnh **select**:

```
orderby student.Last ascending
```

Bây giờ thay đổi mệnh đề **orderby** để nó sắp xếp các kết quả trả về tăng dần của điểm số dầu tiên trong mảng điểm kiểm tra cảu mỗi sinh viên.

```
orderby student.Scores[0] descending
```

Bạn thay đổi định dạng chuỗi được xuất ra màn hình dể xem kết quả:

```
Console.WriteLine("{0}, {1} {2}", s.Last, s.First, s.Scores[0]);
```

V.9.5 Để nhóm các kết quả

Nhóm là một khả năng mạnh mẽ có thể làm được trong biểu thức truy vấn. Một truy vấn với một nhóm mệnh đề dẫn đến việc nhóm chúng theo trình tự, và mỗi nhóm chứa một khóa và một trình tự bao gồm tất cả các thành viên của nhóm đó. Sau đây là một truy vấn mới nhóm các học sinh bằng cách sử dụng các chữ cái đầu tiên trong tên của họ làm khoá.

```
// studentQuery2 is an IEnumerable<IGrouping<char, Student>>
var studentQuery2 =
    from student in students
    group student by student.Last[0];
```

V.9.6 To order the groups by their key value

Khi bạn chạy các truy vấn trước đó, bạn thông báo rằng các nhóm không theo thứ tự chữ cái. Để thay đổi nó, bạn phải cung cấp một mệnh đề **orderby** sau mệnh đề **group**. Nhưng để sử dụng mệnh đề **orderby**, trước tiên bạn cần phải có định danh là phục vụ như một tham chiếu đến các nhóm được tạo ra bởi mệnh đề **group**. Bạn cung cấp các định danh bằng cách sử dụng từ khóa **into**, như sau:

```
var studentQuery4 =
    from student in students
    group student by student.Last[0] into studentGroup
    orderby studentGroup.Key
    select studentGroup;

foreach (var groupOfStudents in studentQuery4)
{
    Console.WriteLine(groupOfStudents.Key);
    foreach (var student in groupOfStudents)
    {
        Console.WriteLine("   {0}, {1}",
            student.Last, student.First);
    }
}
```

Khi bạn chạy truy vấn này, bạn sẽ thấy các nhóm và đang được sắp xếp theo thứ tự chữ cái.

V.9.7 Để giới thiệu một định danh bằng cách sử dụng let

Bạn có thể sử dụng từ khóa **let** để giới thiệu cho một định danh cho bất kỳ biểu thức trả về trong biểu thức truy vấn.Định danh này có thể là một sự tiện nghi, như trong ví dụ sau đây, hoặc nó có thể nâng cao hiệu quả sự thực thi bằng cách lưu trữ các kết quả của một biểu thức để nó không phải được tính toán nhiều lần.

```
var studentQuery5 =
    from student in students
    let totalScore = student.Scores[0] + student.Scores[1] +
        student.Scores[2] + student.Scores[3]
    where totalScore / 4 < student.Scores[0]
    select student.Last + " " + student.First;

foreach (string s in studentQuery5)
{
    Console.WriteLine(s);
}
```

V.9.8 Để sử dụng cú pháp phương thức trong một biểu thức truy vấn

Như đã nêu trong Cú pháp truy vấn và cú pháp phương (LINQ), một số hoạt động truy vấn chỉ có thể được thể hiện bằng cách sử dụng cú pháp phương thức. Dưới đây là đoạn mã tính toán tổng số điểm cho mỗi sinh viên trong trình tự nguồn, và sau đó gọi phương thức *Average ()* trên kết quả của câu truy vấn để tính toán các điểm trung bình của các các class. Lưu ý các vị trí của ngoặc xung quanh biểu thức truy vấn.

```
    var studentQuery6 =
        from student in students
        let totalScore = student.Scores[0] + student.Scores[1] +
            student.Scores[2] + student.Scores[3]
        select totalScore;

    double averageScore = studentQuery6.Average();
Console.WriteLine("Class average score = {0}", averageScore);
```

V.9.9 Để chuyển đổi hoặc dự án trong mệnh đề select

Nó rất phổ biến cho một truy vấn để tạo ra một trình tự các phần tử khác nhau từ các phần tử trong các trình tự nguồn. Xóa hay ghi chú cho truy vấn trước đó của bạn và thực hiện vòng lặp, và thay thế nó với đoạn mã sau đây. Lưu ý rằng các truy vấn sẽ trả về một trình tự của các chuỗi, và thực tế này được phản ánh trong vòng lặp **foreach**.

```
IEnumerable<string> studentQuery7 =
    from student in students
    where student.Last == "Garcia"
    select student.First;
Console.WriteLine("The Garcias in the class are:");
foreach (string s in studentQuery7)
{
Console.WriteLine(s);
}
```

Đoạn mã giới thiệu lúc đầu trong walkthrough này chỉ ra rằng điểm số trung bình là khoảng 334. Để tạo ra một trình tự của ***Students*** có tổng điểm là cao hơn mức trung bình, cùng với ***Student ID*** bạn có thể sử dụng một loại vô danh trong câu lệnh **select**:

```
var studentQuery8 =
    from student in students
    let x = student.Scores[0] + student.Scores[1] +
        student.Scores[2] + student.Scores[3]
    where x > averageScore
    select new { id = student.ID, score = x };

foreach (var item in studentQuery8)
{
    Console.WriteLine("Student ID: {0}, Score: {1}", item.id, item.score);
}
```

VI. LINQ to SQL

LINQ cho SQL là một thành phần của. NET Framework phiên bản 3,5 mà cung cấp một thời gian chạy-cơ sở hạ tầng để quản lý các dữ liệu như các đối tượng.

Chú ý: Các dữ liệu xuất hiện như là một bộ sưu tập của hai chiều-bảng (các mối quan hệ hoặc tập tin phẳng), nơi mà các cột bảng liên quan đến nhau. Để sử dụng LINQ cho SQL một cách có hiệu quả, bạn cần phải làm quen với một số khái niệm cơ bản về cơ sở dữ liệu quan hệ.

Trong LINQ cho SQL, mô hình dữ liệu của cơ sở dữ liệu quan hệ được ánh xạ tới mô hình đối tượng được mô tả trong ngôn ngữ lập trình của các chuyên viên phát triển ứng dụng. Khi chạy các ứng dụng, việc chuyển đổi LINQ cho SQL vào SQL, các truy vấn được tích hợp ngôn ngữ trong mô hình đối tượng và chuyển chúng vào cơ sở dữ liệu để xử lý. Khi cơ sở dữ liệu sẽ trả về kết quả, LINQ cho SQL chuyển chúng trở lại các đối tượng mà bạn đang lập trình bằng ngôn ngữ lập trình của bạn.

Các chuyên viên phát triển ứng dụng sử dụng Visual Studio sử dụng các chuyên viên thiết kế hướng đối tượng mà có thể cung cấp giao diện người dùng để thực thi nhiều tính năng của LINQ cho SQL.

Tài liệu hướng dẫn đi kèm trong bản phát hành này của LINQ SQL mô tả các khối xây dựng cơ bản, các quy trình, và kỹ thuật cần thiết để xây dựng các ứng dụng

LINQ cho SQL. Bạn cũng có thể tìm kiếm trên thư viện MSDN các vấn đề tương tự và tham gia, nơi bạn có thể thảo luận một cách chi tiết các chủ đề này với các chuyên gia. Cuối cùng, các truy vấn tích hợp ngôn ngữ .Net

LINQ to SQL là một phần của công nghệ ADO.NET. Nó được dựa trên các dịch vụ được cung cấp bởi mô hình nhà cung cấp ADO.NET. Do vậy, bạn có thể pha trộn mã LINQ to SQL với các ứng dụng ADO.NET sẵn có và chuyển các giải pháp ADO.NET cho LINQ to SQL. Ví dụ minh họa sau cung cấp cái nhìn cao hơn về các mối quan hệ

VI.1 Kết nối

Bạn có thể cung cấp một kết nối ADO.NET hiện có khi bạn tạo một DataContext cho LINQ to SQL. Tất cả các hoạt động chống lại các DataContext (bao gồm cả các truy vấn) sử dụng kết nối được cung cấp. Nếu kết nối đã mở, LINQ to SQL cho phép như là khi bạn đã kết thúc với nó.

```
string connString = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=c:\northwind.mdf;
   Integrated Security=True; Connect Timeout=30; User Instance=True";
SqlConnection nwindConn = new SqlConnection(connString);
nwindConn.Open();
Northwnd interop_db = new Northwnd(nwindConn);
SqlTransaction nwindTxn = nwindConn.BeginTransaction();
try
{
   SqlCommand cmd = new SqlCommand(
      "UPDATE Products SET QuantityPerUnit = 'single item' WHERE ProductID = 3");
   cmd.Connection = nwindConn;
   cmd.Transaction = nwindTxn;
   cmd.ExecuteNonQuery();

   interop_db.Transaction = nwindTxn;
```

```
    Product prod1 = interop_db.Products
        .First(p => p.ProductID == 4);
    Product prod2 = interop_db.Products
        .First(p => p.ProductID == 5);
    prod1.UnitsInStock -= 3;
    prod2.UnitsInStock -= 5;

    interop_db.SubmitChanges();

    nwindTxn.Commit();
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
    Console.WriteLine("Error submitting changes... all changes rolled back.");
}
nwindConn.Close();
```

Bạn luôn luôn có thể truy cập vào kết nối và đóng nó bằng cách sử dụng các đặc tính kết nối như mã sau đây:

```
db.Connection.Close();
```

VI.2 Giao dịch

Bạn có thể cung cấp DataContext với việc giao dịch cơ sở dữ liệu khi các ứng dụng của bắt đầu giao dịch và bạn muốn DataContext liên quan

Các phương pháp giao dịch thường dùng với .NET Framework đó là sử dụng những đối tượngTransactionScope. Bằng cách sử dụng phương pháp tiếp cận này, bạn có thể thực hiện các giao dịch được phân phối trên cơ sở dữ liệu và quản lý lưu trú bộ nhớ của nguồn. TransactionScope yêu cầu rất ít tài nguyên để khởi động. Chúng thúc đẩy các phương pháp giao dịch chỉ khi có nhiều kết nối trong phạm vi giao dịch.

```
using (TransactionScope ts = new TransactionScope())
{
    db.SubmitChanges();
    ts.Complete();
}
```

Bạn không thể sử dụng phương pháp tiếp cận này cho tất cả các cơ sở dữ liệu. Ví dụ, các kết nối cho SqlClient không thể thúc đẩy systemTransactions hệ thống khi nó hoạt động dựa trên một máy chủ SQL Server 2000. Thay vào đó, nó tự động vào một enlists đầy đủ, phân bố giao dịch bất cứ khi nào nó thấy một phạm vi giao dịch đang được sử dụng.

VI.3 Lệnh SQL trực tiếp

Đôi khi bạn có thể gặp tình huống mà khả năng của DataContext để truy vấn hoặc gửi đi các thay đổi không đủ cho các công việc chuyên môn mà bạn muốn thực hiện. Trong những trường hợp đó, bạn có thể sử dụng các phương pháp ExecuteQuery để xuất ra các lệnh SQL cho cơ sở dữ liệu và chuyển đổi kết quả truy vấn cho các đối tượng.

Ví dụ, giả định rằng dữ liệu của các khách hàng trải ra trên hai bảng (khách hàng 1 và khách hàng 2). Các truy vấn trả về sau đây là một kết quả của đối tượng khách hang.

```
IEnumerable<Customer> results = db.ExecuteQuery<Customer>(
  @"select c1.custid as CustomerID, c2.custName as ContactName
    from customer1 as c1, customer2 as c2
    where c1.custid = c2.custid"
);
```

Chỉ cần tên cột trong các kết quả nối với các thuộc tính cột của một lớp thực thể LINQ to SQL tạo ra các đối tượng ra khỏi bất kỳ truy vấn SQL.

Các tham số

Phương pháp ExecuteQuery chấp nhận tham số. Mã sau đây thực thi truy vấn bằng tham số:

```
IEnumerable<Customer> results = db.ExecuteQuery<Customer>(
  "select contactname from customers where city = {0}",
  "London"
);
```

VI.4 Cách kết nối một cơ sở dữ liệu (LINQ to SQL)

DataContext là đường dẫn chính mà bạn kết nối với một cơ sở dữ liệu,sau đó bạn truy lục dữ liệu từ đó và gửi trở lại các thay đổi. Bạn chỉ cần sử dụng DataContext tương

tự như khi bạn sử dụng một ADO.NET SqlConnection. Trong thực tế, các DataContext được khởi động với một kết nối hoặc kết nối chuỗi mà bạn cung cấp.

Mục đích của DataContext là để dịch các yêu cầu cho các đối tượng vào các truy vấn của SQL để dựa trên các cơ sở dữ liệu, và sau đó thu thập các đối tượng ra khỏi các kết quả. DataContext cho phép Language-Integrated Query (LINQ) bằng cách thực thi các mô hình tổ chức các toán tử truy vấn chuẩn chẳng hạn như mệnh đề "**Where**" và "**Select**".

Ví dụ

Trong ví dụ sau, các DataContext được sử dụng để kết nối với các cơ sở dữ liệu mẫu Northwind và để truy lục lại hàng của khách hàng là người sống ở London.

```
// DataContext takes a connection string.
DataContext db = new DataContext(@"c:\Northwnd.mdf");
// Get a typed table to run queries.
Table<Customer> Customers = db.GetTable<Customer>();
// Query for customers from London.
var query =
   from cust in Customers
   where cust.City == "London"
   select cust;
foreach (var cust in query)
   Console.WriteLine("id = {0}, City = {1}", cust.CustomerID, cust.City);
```

Tất cả các bảng cơ sở dữ liệu được trình bày trong bảng bằng phương pháp GetTable sử dụng lớp sẵn có để nhận dạng nó.

Phương pháp thực hành để trình bày một DataContext thay vì dựa trên lớp DataContext cơ bản và phương pháp GetTable. Loại DataContext trình bày tập hợp bảng như một thành phần của Context theo ví dụ sau:.

```
public partial class Northwind : DataContext
{
   public Table<Customer> Customers;
   public Table<Order> Orders;
   public Northwind(string connection) : base(connection) { }
}
```

Sau đó bạn có thể trình bày truy vấn cho khách hàng từ London đơn giản hơn bằng cách sau:

```
Northwnd db = new Northwnd(@"c:\Northwnd.mdf");
var query =
   from cust in db.Customers
   where cust.City == "London"
   select cust;
foreach (var cust in query)
   Console.WriteLine("id = {0}, City = {1}", cust.CustomerID, cust.City);
```

VI.5 Cách tạo cơ sở dữ liệu (LINQ to SQL)

Các lớp có các thuộc tính được mô tả về cấu trúc của bảng và cột dữ liệu liên quan. Bạn có thể sử dụng thông tin này để tạo mới các trường cơ sở dữ liệu. Khi bạn gọi phương thức CreateDatabase trên DataContext, LINQ to SQL để xây dựng một cơ sở dữ liệu mới với một cơ cấu xác định bởi các đối tượng

Bạn có thể sử dụng tính năng này trong bất kỳ kịch bản nào, đặc biệt là khi được biết đến như là một nhà cung cấp dịch vụ dữ liệu ví dụ như SQL Server 2005 Express:

- Bạn đang xây dựng một ứng dụng tự động cài đặt trên một hệ thống của khác hàng.
- Bạn đang xây dựng mộtứng dụng dành cho Client mà cần lưu cơ sở dữ liệu cục bộ lưu trong trạng thái ngoại tuyến.

Chú ý: Thuộc tính dữ liệu từ mô hình đối tượng không thể mã hóa tất cả cấu trúc của cơ sở dữ liệu hiện có. Thuộc tính không đại diện cho các nội dung của chức năng do người dùng quyết định, các thủ tục lưu giữ, triggers , kiểm tra các ràng buộc dữ liệu. Chức năng CreateDatabase tạo ra một bản sao của cơ sở dữ liệu chỉ trong phạm vi của những thông tin được mã hoá trong mô hình đối tượng. Hành động này là đủ cho một loạt các cơ sở dữ liệu.

Bạn cũng có thể sử dụng CreateDatabase với SQL Server bằng cách sử dụng một tập tin .mdf hay chỉ là định mục tùy thuộc vào chuỗi kết nối. LINQ to SQL sử dụng chuỗi kết nối để xác định các cơ sở dữ liệu được tạo ra và cách nó được tạo trên máy chủ

Ví dụ: Đoạn mã sau đây cung cấp một ví dụ về cách bạn sẽ tạo ra một cơ sở dữ liệu mới có tên MyDVDs.mdf.

```
public class MyDVDs : DataContext
{
   public Table<DVD> DVDs;
   public MyDVDs(string connection) : base(connection) { }
}

[Table(Name = "DVDTable")]
public class DVD
{
   [Column(IsPrimaryKey = true)]
   public string Title;
   [Column]
   public string Rating;
}
```

Bạn có thể sử dụng mô hình đối tượng để tạo ra một cơ sở dữ liệu như sau

```
public void CreateDatabase()
{
   MyDVDs db = new MyDVDs("c:\\mydvds.mdf");
   db.CreateDatabase();
}
```

LINQ to SQL cũng cung cấp một API để thả một cơ sở dữ liệu hiện có trước khi tạo cơ sở dữ liệu mới mới. Bạn có thể chỉnh sửa mã trong kịch bản 1để kiểm tra một phiên bản hiện có của cơ sở dữ liệu. Sử dụng DatabaseExists và DeleteDatabase phương pháp để thực hiện phương thức tiếp cận này. Sau khi bạn gọi CreateDatabase, các cơ sở dữ liệu mới tồn tại và chấp nhận các truy vấn và các lệnh .

Bạn có thể thực hiện các phương pháp tiếp cận này bằng cách sử dụng mã như sau:

```
public void CreateDatabase2()
{
   MyDVDs db = new MyDVDs(@"c:\mydvds.mdf");
   if (db.DatabaseExists())
   {
      Console.WriteLine("Deleting old database...");
      db.DeleteDatabase();
   }
   db.CreateDatabase();
```

VI.6 Bạn có thể làm gì với LINQ to SQL

LINQ SQL để hỗ trợ tối đa các khả năng quan trọng đáp ứng được mong muốn của bạn giống như một chuyên viên phát triển SQL. Bạn có thể truy vấn các thông tin, chèn, cập nhật, và xóa thông tin từ bảng.

VI.6.1 Lựa chọn(Select)

Lựa chọn là đạt được bằng cách chỉ viết một truy vấn LINQ trong ngôn ngữ lập trình của bạn và sau đó xử lý truy vấn để lấy kết quả. LINQ to SQL tự dịch tất cả các hoạt động cần thiết vào các hoạt động SQL cần thiết mà bạn đang làm quen.

Ví dụ sau, công ty, tên công ty của khách hàng từ London được truy lục và hiển thị trong cửa sổ console.

```
// Northwnd inherits from System.Data.Linq.DataContext.
Northwnd nw = new Northwnd(@"northwnd.mdf");

var companyNameQuery =
   from cust in nw.Customers
   where cust.City == "London"
   select cust.CompanyName;

foreach (var customer in companyNameQuery)
{
   Console.WriteLine(customer);
}
```

VI.6.2 Cách chèn hàng vào trong cơ sở dữ liệu (LINQ to SQL)

Bạn chèn hàng vào một cơ sở dữ liệu bằng cách thêm các đối tượng vào bảng LINQ to SQL (TEntity) và sau đó gửi các thay đổi đối tới cơ sở dữ liệu. LINQ cho SQL dịch vào những thay đổi của bạn thích hợp lệnh INSERT SQL thích hợp. Các bước sau tóm tắt một Dưới đây là những bước giả định rằng một hợp lệ DataContext kết nối bạn vào cơ sở dữ liệu Northwind.

```
// Create a new Order object.
Order ord = new Order
{
```

```
    OrderID = 12000,
    ShipCity = "Seattle",
    OrderDate = DateTime.Now
    // …
};
// Add the new object to the Orders collection.
db.Orders.InsertOnSubmit(ord);

// Submit the change to the database.
try
{
    db.SubmitChanges();
}
catch (Exception e)
{
    Console.WriteLine(e);
    // Make some adjustments.
    // ...
    // Try again.
    db.SubmitChanges();
}
```

VI.6.3 Chèn một hàng vào cơ sở dữ liệu

1. Tạo mới một đối tượng trong đó bao gồm các dữ liệu cột được gửi đến.
2. Thêm vào các đối tượng mới cho tập hợp các bảng LINQ to SQL với bảng target trong cơ sở dữ liệu.
3. Thay đổi để gửi đi các cơ sở dữ liệu.

Chèn(Insert)

Để chèn một SQL, chỉ cần thêm các đối tượng vào mô hình đối tượng bạn đã tạo, và gọi các SubmitChanges trên DataContext. Trong ví dụ sau, một khách hàng mới và các thông tin về các khách hàng sẽ được thêm vào bảng Khách hàng bằng cách sử dụng InsertOnSubmit.

```
// Northwnd inherits from System.Data.Linq.DataContext.
Northwnd nw = new Northwnd(@"northwnd.mdf");
```

```
Customer cust = new Customer();
cust.CompanyName = "SomeCompany";
cust.City = "London";
cust.CustomerID = "98128";
cust.PostalCode = "55555";
cust.Phone = "555-555-5555";
nw.Customers.InsertOnSubmit(cust);

// At this point, the new Customer object is added in the object model.
// In LINQ to SQL, the change is not sent to the database until
// SubmitChanges is called.
nw.SubmitChanges();
```

VI.6.4 Cách cập nhật hàng trong cơ sở dữ (LINQ to SQL)

Bạn có thể cập nhật hàng trong một cơ sở dữ liệu bằng cách thay đổi giá trị thành viên của các đối tượng kết hợp với bảng LINQ to SQL (TEntity) và sau đó gửi các thay đổi đối vào cơ sở dữ liệu. LINQ cho SQL chuyển đổi sự thay đổi vào trong lệnh SQL **Cập nhật(update**)

Để cập nhật một hàng trong cơ sở dữ liệu

1. Truy vấn cơ sở dữ liệu cho các hàng được cập nhật.
2. Thay đổi giá trị thành viên trong đối tượng LINQ to SQL
3. Gửi các thay đổi đối với cơ sở dữ liệu.

Ví dụ

Ví dụ truy vấn sau cơ sở dữ liệu cho các lệnh # 11000, và sau đó thay đổi các giá trị của ShipName và ShipVia trong kết quả của đối tượng Object. Cuối cùng, các thay đổi với các thành viên các giá trị được gửi đến cơ sở dữ liệu như là thay đổi trong ShipName và ShipVia cột.

```
// Query the database for the row to be updated.
var query =
    from ord in db.Orders
    where ord.OrderID == 11000
    select ord;
```

```
// Execute the query, and change the column values
// you want to change.
foreach (Order ord in query)
{
   ord.ShipName = "Mariner";
   ord.ShipVia = 2;
   // Insert any additional changes to column values.
}
// Submit the changes to the database.
try
{
   db.SubmitChanges();
}
catch (Exception e)
{
   Console.WriteLine(e);
   // Provide for exceptions.
}
```

VI.6.5 Cập nhật

Để cập nhật thông tin cơ sở dữ liệu cho một mục nhập, đầu tiên lấy mục và chỉnh sửa nó trực tiếp đối tượng trong mô hình. Sau khi bạn đã sửa đổi đối tượng, gọi SubmitChanges trên DataContext để cập nhật cơ sở dữ liệu.

Trong ví dụ sau, tất cả các khách hàng đang được tải về từ London. Sau đó, tên của thành phố là thay đổi từ "London" thành "London - Metro". Cuối cùng, SubmitChanges được gọi là để gửi các thay đổi đối với cơ sở dữ liệu.

```
Northwnd nw = new Northwnd(@"northwnd.mdf");

var cityNameQuery =
   from cust in nw.Customers
   where cust.City.Contains("London")
   select cust;

foreach (var customer in cityNameQuery)
{
   if (customer.City == "London")
   {
      customer.City = "London - Metro";
```

```
    }
}
nw.SubmitChanges();
```

VI.7 Cách xóa hàng trong cơ sở dữ liệu (LINQ to SQL)

Bạn có thể xoá các hàng trong một cơ sở dữ liệu tương ứng bằng cách gỡ bỏ các đối tượng LINQ SQL tương ứng từ tập hợp bảng liên quan. LINQ cho SQL dịch bạn thay đổi vào lệnh DELETE thích hợp trong SQL.

LINQ cho SQL không hỗ trợ hoặc xác định hoạt động xếp tầng delete. Nếu bạn muốn xóa một hàng trong bảng gặp khó khăn, bạn phải hoàn tất một trong các nhiệm vụ sau:

- Tạo rule trên DELETE CASCADE trong rang buộc foreign-key trên cơ sở dữ
- Sử dụng mã của bạn để xóa các đối tượng con đề phòng đối tượng cha bị xóa.

Mặt khác, vẫn có trưởng hợp ngoại lệ. Xem ví dụ mã thứ 2 cho chủ đề này

Dưới đây là các bước tóm tắt các DataContex hợp lệ kết nối bạn vào cơ sở dữ liệu Northwind.

VI.7.1 Để xóa hàng trong cơ sở dữ liệu

1. Query the database for the row to be deleted.
2. Call the DeleteOnSubmit method.
3. Submit the change to the database.

1. Truy vấn cơ sở dữ liệu cho các hàng sẽ được xóa.
2. Gọi các phương pháp DeleteOnSubmit.
3. Gửi các thay đổi tới cơ sở dữ liệu.

Ví dụ:Các ví dụ mã đầu tiên truy vấn cơ sở dữ liệu cho các chi tiết theo thứ tụ thuộc Order # 11000, đánh dấu những chi tiết theo thứ tự cho các lệnh xóa, và gửi các thay đổi này vào cơ sở dữ liệu.

```
// Query the database for the rows to be deleted.
var deleteOrderDetails =
   from details in db.OrderDetails
   where details.OrderID == 11000
   select details;
```

```
foreach (var detail in deleteOrderDetails)
{
   db.OrderDetails.DeleteOnSubmit(detail);
}
try
{
   db.SubmitChanges();
}
catch (Exception e)
{
   Console.WriteLine(e);
   // Provide for exceptions.
}
```

Trong ví dụ thứ hai, mục tiêu để bỏ đó là order #10250. Mã này kiểm tra bảng OrderDetails để biết được nếy Order đó bị bỏ có trẻ con ở đó. Nếu Order đó có trẻ con, đứa trẻ con đầu tiên và sau đó Order đó được đánh dấu để xóa. DataContext đặt lệnh xóa cuối cùng trong Order đúng sau đó xóa các lệnh đã gửi vào cơ sở dữ liệu chờ bằng các ràng buộc cơ sở dữ liệu

```
Northwnd db = new Northwnd(@"c:\northwnd.mdf");
db.Log = Console.Out;
// Specify order to be removed from database
int reqOrder = 10250;
// Fetch OrderDetails for requested order.
var ordDetailQuery =
   from odq in db.OrderDetails
   where odq.OrderID == reqOrder
   select odq;
foreach (var selectedDetail in ordDetailQuery)
{
   Console.WriteLine(selectedDetail.Product.ProductID);
   db.OrderDetails.DeleteOnSubmit(selectedDetail);
}
// Display progress.
Console.WriteLine("detail section finished.");
Console.ReadLine();
// Determine from Detail collection whether parent exists.
if (ordDetailQuery.Any())
{
   Console.WriteLine("The parent is presesnt in the Orders collection.");
```

```
    // Fetch Order.
    try
    {
        var ordFetch =
            (from ofetch in db.Orders
             where ofetch.OrderID == reqOrder
             select ofetch).First();
        db.Orders.DeleteOnSubmit(ordFetch);
        Console.WriteLine("{0} OrderID is marked for deletion.", ordFetch.OrderID);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        Console.ReadLine();
    }
}
else
{
    Console.WriteLine("There was no parent in the Orders collection.");
}
// Display progress.
Console.WriteLine("Order section finished.");
Console.ReadLine();
try
{
    db.SubmitChanges();
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
    Console.ReadLine();
}
// Display progress.
Console.WriteLine("Submit finished.");
Console.ReadLine();
```

Xóa

Để Xoá một mục từ collection và sau đó gọi SubmitChanges trên DataContext để xác nhận sự thay đổi LINQ to SQL, không xác nhận các hoạt động xóa xếp tầng . Nếu bạn muốn xóa một hàng trong một bảng có các ràng buộc chống lại nó.

Trong ví dụ sau, các khách hàng có ID khách hàng là 98128 là lấy từ các cơ sở dữ liệu. Sau đó, sau khi xác nhận rằng các hàng củakhách hàng được gọi ra, lệnh DeleteOnSubmit được gọi là để loại bỏ các đối tượng từ các bộ sưu tập. Cuối cùng, SubmitChanges được gọi là để gửi lệnh xóa tới cơ sở dữ liệu.

```
Northwnd nw = new Northwnd(@"northwnd.mdf");
var deleteIndivCust =
   from cust in nw.Customers
   where cust.CustomerID == "98128"
   select cust;

if (deleteIndivCust.Count() > 0)
{
   nw.Customers.DeleteOnSubmit(deleteIndivCust.First());
   nw.SubmitChanges();
}
```

VI.8 Quy trình lưu trữ (LINQ to SQL)

LINQ to SQL để sử dụng các mô hình đối tượng để trình bày các quy trình lưu trữ trong cơ sở dữ liệu. Bạn dùng phương pháp như các quy trình lưu trữ dữ liệu băng các áp dụng các thuộc tính FunctionAttribute khi cần thiết và các thuộc tính ParameterAttribute. Các chuyên gia phát triển ứng dụng sử dụng Visual Studio thông thường sẽ dử dụng Object Relational Designer để ánh xạ các quy trình lưu trữ dữ liệu. Chủ để trong phần này chỉ cho bạn cách thiết lập và gọi các phương pháp trong các ứng dụng khi bạn mã hóa dữ liệu.

Phần này không mô tả cách sử dụng nhưng mô tả các thực thi các lệnh chèn, cập nhật, và xóa các hoạt động của cơ sở dữ liệu

VI.8.1 Chèn, cập nhật và xóa các hoạt động của cơ sở dữ liệu trong LINQ to SQL

Bạn thực hiện các Chèn, cập nhật và xóa các hoạt động trong LINQ cho SQL bằng cách thêm, thay đổi, và loại bỏ các đối tượng trong mô hình đối tượng. Theo mặc định, LINQ cho SQL dịch các hành động của bạn để SQL và gửi các thay đổi tới cơ sở dữ liệu. LINQ to SQL để cung cấp tối đa tính linh hoạt trong việc thao tác và giữa các thay đổi bạn đã thực hiện với các đối tượng. Ngay sau khi các đối tượng có sẵn (hoặc bằng cách tải chúng thông qua một truy vấn, hoặc bằng cách xây dựng lại chúng), bạn có thể thay đổi chúng như các đối tượng tiêu biểu trong ứng dụng của bạn. Tức là, bạn có thể thay đổi giá trị của chúng, bạn có thể thêm chúng vào tập hợp và bạn cũng có thể loại bỏ chúng khỏi tập hợp. LINQ SQL để theo dõi các thay đổi và đã sẵn sàng để truyền tải chúng trở lại cơ sở dữ liệu khi bạn gọi SubmitChanges.

Chú ý: LINQ cho SQL không hỗ trợ hoặc ghi nhận các hoạt động xóa xếp tầng. Nếu bạn gặp khó khăn khi muốn xóa một hàng trong một bảng, bạn cần phải hoặc là thiết lập rule: **ON DELETE CASCADE** ở rang buộc về khóa trong cơ sở dữ liệu, hoặc sử dụng của riêng bạn để xóa các đối tượng trẻ em mà có thể không cho đối tượng cha mẹ bị xóa. Mặt khác vẫn có ngoại lệ. Đoạn trích sau đây sử dụng lớp Customer và Order từ cơ sở dữ liệu mẫu Northwind.

```
Northwnd db = new Northwnd(@"c:\Northwnd.mdf");
// Query for a specific customer.
var cust =
    (from c in db.Customers
     where c.CustomerID == "ALFKI"
     select c).First();
// Change the name of the contact.
cust.ContactName = "New Contact";
// Create and add a new Order to the Orders collection.
Order ord = new Order { OrderDate = DateTime.Now };
cust.Orders.Add(ord);
// Delete an existing Order.
Order ord0 = cust.Orders[0];
// Removing it from the table also removes it from the Customer's list.
db.Orders.DeleteOnSubmit(ord0);
```

```
// Ask the DataContext to save all the changes.
db.SubmitChanges();
```

VI.8.2 Cách gửi những thay đổi đến cơ sở dữ liệu (LINQ to SQL)

Bất kể có bao nhiêu thay đổi bạn để làm cho các đối tượng của bạn, chỉ có thay đổi trong bản sao trong bộ nhớ. Bạn đã không có thay đổi vào dữ liệu cuối cùng trong cơ sở dữ liệu. Những thay đổi của bạn không được chuyển đến các máy chủ cho đến khi bạn gọi lệnh SubmitChanges trên DataContext.

Khi bạn thực hiện lệnh gọi này, các DataContext cố gắng dịch các thay đổi của bạn vào lệnh SQL tương ứng. Bạn có thể sử dụng nguyên lý thiết kế máy tính tùy thích để ghi đè những hành động này, nhưng trật tự của những submission được sắp xếp lại bởi một dịch vụ của DataContext đó là: *change processor theo t*rình tự như sau:

1. Khi bạn gọi lệnh SubmitChanges, LINQ SQL kiểm tra các thiết lập của các đối tượng được biết đến để xác định xem trường mới đã được đính kèm với chúng. Nếu chúng có, các trường hợp này sẽ được thêm vào để thiết lập các đối tượng được kiểm tra.
2. Tất cả các đối tượng có lệnh đang chờ thay đổi được sắp xếp vào trong vào một chuỗi các đối tượng dựa trên các phụ thuộc giữa chúng. Các đối tượng có thay đổi phụ thuộc vào các đối tượng khác được sắp xếp sau sự phụ thuộc đó.
3. Trước khi thực sự thay đổi bất kỳ được truyền, LINQ SQL bắt đầu một giao dịch để tóm tắt một chuỗi các lệnh riêng lẻ.

Những thay đổi đối với các đối tượng được dịch từng bước một vào các lệnh trong SQL và chuyển vào máy chủ.

Tại thời điểm này, bất kỳ lỗi nào phát hiện bởi các cơ sở dữ liệu tạo ra quá trình Submision để ngừng và vẫn có ngoại lệ xảy ra. Tất cả các thay đổi đó trong cơ sở dữ liệu được cuộn trở lại nếu không có ***Submission*** nào xuất hiện. DataContext vẫn có thể ghi lại đầy đủ tất cả sự thay đổi đó. Ngoài ra, bạn có thể cố gắng khắc phục sự cố và gọi lệnh SubmitChangé như ví dụ sau: .

Ví dụ**:** Khi giao dịch xung quanh ***Submission*** thành công DataContext chấp nhận sự thay đổi đó vào các đối tượng bằng cách bỏ qua các thống tin về sự thay đổi bản ghi.

```
Northwnd db = new Northwnd(@"c:\northwnd.mdf");
// Make changes here.
try
{
   db.SubmitChanges();
}
catch (ChangeConflictException e)
{
   Console.WriteLine(e.Message);
   // Make some adjustments.
   // ...
   // Try again.
   db.SubmitChanges();
}
```

Cách tạo mới các lớp được ánh xạ vào các bảng và các khung nhìn trong LINQ to SQL (O / R Designer). Các lớp trong LINQ to SQL được ánh xạ vào các bảng cơ sở dữ liệu và được gọi là lớp thực thể. Lớp thực thể này ánh xạ tới bản ghi trong khi các đặc tính riêng lẻ của lớp được ánh xạ vào các cột riêng mà có thể được ghi lại. Tạo mới các lớp thực thể dựa trên bảng cơ sở dữ liệu hoặc tổng quan cơ sở dữ liệu bằng cách kéo các bảng hoặc nhìn từ **Server Explorer/Database Explorer**. O/R Designer tạo ra các lớp và áp dụng vào các thuộc tính LINQ to SQL đặc biệt để kích hoạt các chức năng LINQ to SQL (truyền dữ liệu và hiệu chỉnh các tính năng của DataContext. Thông tin chi tiết về lớp LINQ to SQL xem mô hình đối tượng LINQ to SQL.

Chú ý: O / R Designer là một đối tượng đơn giản liên quan đến ánh xạ dữ liệu bởi vì nó chỉ hỗ trợ mối quan hệ mapping 1:1. Nói cách khác, một tổ chức lớp học có thể chỉ có một 1:1 mapping relationship với bảng cơ sở dữ liệu. Quá trình chuyển đổi dữ liệu phức tạp như ánh xạ entity class vào các bảng phức tạp không được hỗ trợ. Tuy nhiên, bạn có thể ánh xạ entity class tới một entity class view mà có thể nối các bảng có liên quan.

VI.8.3 Tạo các lớp LINQ to SQL được ánh xạ vào bảng cơ sở dữ liệu or các khung nhìn.

Kéo các bảng hoặc các khung nhìn từ Server Explorer / Explorer vào các O / R Designer để tạo ra ngoài các lớp thực thể . Hơn nữa phương pháp DataContext là phương pháp được sử dụng để thực thi cập nhật cơ sở dữ liệu.

Theo mặc định, các LINQ SQL runtime tạo ra logic để lưu các thay đổi từ một entity class được cập nhật vào cơ sở dữ liệu. Logic này được dựa trên sơ đồ của bảng (định nghĩa cột và các thông tin quan trọng) Nếu bạn không muốn hành động này, bạn có thể cấu hình một lớp thực thể sử dụng quy trình lưu trữ dữ liệu để thực thi việc chèn, cập nhật, xóa thay vì sử dụng các hoạt động mặc định của LINQ to SQL tại thời gian chạy.

Chú ý: Máy tính của bạn có thể hiện ra các tên khác nhau hay địa phương khác nhau trong số các yếu tố gioa diện người dùng Visual Studio trong hướng dẫn sau. Các ấn bản Visual Studio mà bạn có và các thiết lập mà bạn xác định các yếu tố này.

VI.8.4 Để tạo các lớp được ánh xạ vào dữ liệu bảng hoặc các khung nhìn trong LINQ to SQL.

1. Trong **Server/Database Explorer**, mở rộng **Tables** or **Views** và định vị database table or view mà bạn muốn sử dụng trong ứng dụng của bạn.
2. Kéo bảng hoặc View vào trong O/R Designer.

Một lớp thực thể được tạo ra và xuất hiện trong bề mặt của thiết kế. Lớp thực thể có đặc tính mà ánh xạ vào các cột trong bảng hay View dữ liệu được chọn.

Tạo đối tượng dữ liệu nguồn và hiển thị dữ liệu cho một Form

Sau khi bạn tạo các lớp thực thể bằng cách sử dụng O / R Designer, bạn có thể tạo đối tượng nguồn dữ liệu và định cư Data Sources Window trên các lớp thực thể.

Để tạo một đối tượng nguồn dữ liệu dựa trên các lớp thực thể trong SQL LINQ

1. Trên trình đơn Build, nhấp chuột vào **Build Solution** để xây dựng cho dự án của bạn.
2. Trên trình đơn Data, nhấp chuột vào **Show Data Sources**

3. Trên cửa sổ **Data Sources**, nhấp chuột vào **Add New Data Source**.

4. Click vào **Object** trên trang **Choose a Data Source Type** và sau đó nhấp chuột vào **Next**

5. Mở rộng các nodes và vị trí và chọn lớp của bạn.

Chú ý: Nếu lớp khách hàng không sẵn có, hủy trình wizard, xây dựng các dự án, và chạy trình wizard lại.

1. Click vào **Finish** để tạo ra các nguồn dữ liệu và thêm vào và thêm vào entity class **Customer** vào cửa sổ nguồn dữ liệu.

2. Kéo các mục từ cửa sổ **Data Sources vào trong Form**

VII. LINQ to XML

LINQ to XML cung cấp giao diện lập trình XML mà đòn bẩy là .NET Language-Integrated Query (LINQ) Framework. LINQ to XML sử dụng ngôn ngữ mới nhất của .NET Framework và được so sánh và thiết kế lại với giao diện lập trình XML Document Object Model (DOM)

LINQ to XML bao gồm các phương pháp khác nhau cho phép bạn có thể thay đổi sơ đồ XML một cách trực tiếp. bạn có thể thêm các yếu tố, xóa các yếu tố, thay đổi nổi dung của một yếu tố đó và thêm các thuộc tính vv. Giao diện lập trình được mô tả trong sơ đồ Modifying XML. Nếu bạn đang làm lại thông qua một trong các axis như Element và b ạn đang thay đổi sơ đồ XML như làm lạ thông qua một axis như khi bạn làm thông qua axis, bạn có thể gặp với một vài lỗi lạ. Sự cố này được biết đến như là "The Halloween Problem".

VII.1 Định nghĩa.

Khi bạn viết một vài mã sử dụng LINQ mà thông qua một tập hợp, bạn đang viết mã với lối viết tường thuật. Nó gần giống như cái bạn muốn và hơn nữa là “How”- bạn muốn làm như thế nào.

1) Nhận yếu tố đầu tiên.

2) Kiểm tra với một vài điều kiện.

3) Thay đổi nó.

4) Đặt nó trở lại danh sách sau đó sẽ là mã bắt buộc. Bạn đang nói với máy tính rằng bạn làm cái bạn muốn như thế nào.

Trộn lẫn với những lối viết mã này trong một số hoạt động tương tự là cái dẫn đến các sự cố như sau:

Giả dụ bạn có một danh sách các liên kết với 3 mục (a, b, và c):

a -> b -> c

Bây giờ giả dụ bạn muốn chuyển danh sách các liên kết và thêm vào 3 mục mới (a', b', và c'). Bạn muốn danh sách các liên kết có kết quả là:

a -> a' -> b -> b' -> c -> c'

Cho nên bạn sẽ viết mã thông qua danh sách và cho các mục khác bạn thêm một mục mới bào bên phải. Điều gì sẽ xảy ra nếu mã của bạn sẽ nhìn thấy một yếu tố đầu tiên và chèn thêm a’ sau nó. Bây giờ mã của bạn sẽ chuyển sang nút tiếp theo trong danh sách mà bây giờ là a’. nó sẽ thêm vào một mục mới trong danh sách là a’’.

Cách xử lý sự cố này trong thực tế như thế nào? Bạn có thể có bản copy của danh sách các liên kết gốc và tạo một danh sách hoàn toàn mới khác. Hoặc nếu bạn đang mã hóa lệnh, bạn có thể tìm thấy mục đầu tiên, thêm vào mục mới và sau đó tiến lên phía trước 2 lần trong danh sách các liên kết, trên danh sách mà bạn vừa thêm.

VII.2 Thêm vào trong khi lặp.

Ví dụ, giả sử bạn muốn viết một vài mã cho các yếu tố trong một sơ đồ, bạn muốn tạo ra một yếu tố khác tương tự:

```
XElement root = new XElement("Root",
   new XElement("A", "1"),
   new XElement("B", "2"),
   new XElement("C", "3")
);
foreach (XElement e in root.Elements())
   root.Add(new XElement(e.Name, (string)e));
```

Đoạn mã này đi vào vòng lặp vô hạn. Trạng thái foreach lặp lại thông qua axis các phần tử, thêm vào các yếu tố mới trong doc. Nó kết thúc việc lặp đi lặp lại thông qua các yếu tố mới thêm vào. Và bởi vì nó định phần các đối tượng mới với vòng lặp nó sẽ chiếm hết toàn bộ bộ nhớ hiện hữu.

Bạn có thể khắc phục một số sự cố bằng cách đẩy các tập hợp sử dụng các hoạt động truy vấn chuẩn ToList(Tsource) như sau:

Bây giờ các mã làm việc như sơ đồ XML bên dưới:

VII.3 Xóa trong khi lặp.

Nếu bạn muốn xóa tất cả các nút tại mọi thời điểm bạn có thể gặp rủi ro khi viết mã như ví dụ sau:

```
XElement root = new XElement("Root",
   new XElement("A", "1"),
   new XElement("B", "2"),
   new XElement("C", "3")
);
foreach (XElement e in root.Elements().ToList())
   root.Add(new XElement(e.Name, (string)e));
Console.WriteLine(root);

XElement root = new XElement("Root",
   new XElement("A", "1"),
   new XElement("B", "2"),
   new XElement("C", "3")
);
foreach (XElement e in root.Elements())
   e.Remove();
Console.WriteLine(root);
```

Tuy nhiên, điều này không phải là cái bạn muốn. Trong trường hợp này, sau khi bạn gỡ bỏ yếu tố đầu tiên,nó được gỡ bỏ từ sơ đồ XML không chứa gốc, và mã trong phương thức Element mà đang được lặp lại không thể tìm thấy trong các yếu tố tiếp theo.

Quy trình mã trước theo dữ liệu vào sau:

Giải pháp này được gọi là ToList(TSource) để cụ thể hóa tập hợp như sau:

```
XElement root = new XElement("Root",
   new XElement("A", "1"),
   new XElement("B", "2"),
   new XElement("C", "3")
);
foreach (XElement e in root.Elements().ToList())
   e.Remove();
Console.WriteLine(root);
```

Và cuối cùng bạn có thể loại trừ các trường hợp lặp bằng cách RemoveAll trong các yếu tố cha:

```
XElement root = new XElement("Root",
   new XElement("A", "1"),
   new XElement("B", "2"),
   new XElement("C", "3")
);
root.RemoveAll();
Console.WriteLine(root);
```

VII.4 Tại sao không thể xử lý LINQ tự động?

Phương pháp tiếp cận này sẽ luôn mang lại tài nguyên cho bộ nhớ thay vì chỉ đánh giá không. Tuy vậy, nó sẽ rất tốn tài nguyên trong điều kiện thực thi và sử dụng bộ nhớ. Trong thực tế, nếu LINQ và LINQ to XML là để tiếp cận thì nó sẽ rơi vào tình huống thật.

Một phương pháp tiếp cận khác là đặt vào trong một trong số cú pháp giao dịch vào trong LINQ và chương trình biên dịch để phân tích mã và xác định tập hợp đặc biệt đó có cần thiết để được cụ thể hóa. Tuy vậy, việc cố gắng xác định tất cả các mã mà có tác động thứ yếu rất phức tạp. Hãy xem đoạn mã sau:

```
var z =
   from e in root.Elements()
   where TestSomeCondition(e)
   select DoMyProjection(e);
```

Như mã sẽ cần phân tích các phương thức TestSomeCondition và DoMyProjection và tất cả các phương thức để xác định nếu bất kì mã nào có tác động thứ yếu. Nhưng nếu các mã phân tích có thể không thể tìm ra bất kì mã nào là mã thứ yếu. Nó sẽ cần phải chọn mã có tác động thứ yếu trong các yếu tố con của trong tình huống này.

LINQ to XML không phân tích. Nó phụ thuộc vào sự cố xảy ra.

Hướng dẫn.

Đầu tiên không trộn lẫn mã tường thuật với mã lệnh.

Thậm chí khi bạn biết chính xác ngữ nghĩa của nó trong tập hợp của bạn và ngữ nghĩa của phương thức mà có thể thay đổi trong sơ đồ XML, nếu bạn viết một vài mã rõ ràng để tránh được một số tình huống xấu xảy ra, mã của bạn sẽ cần phải duy trì bởi các chuyên gia phát triển ứng dụng khác trong tương lai và họ phải biết rõ các vấn đề này.

Nếu bạn trộn mã tường thuật với mã lệnh, thì mã của bạn dễ bị bẻ gẫy hơn.

Nếu bạn viết mã cụ thể hóa cho một tập hợp cho nên những vấn đề này nên tránh, chú ý nó với phần diễn giải thích hợp trong đoạn mã của bạn cho nên các lập trình viên bảo trì sẽ hiểu được vấn đề đó.

Thứ hai, nếu việc thực thi cho phép dự dụng mã tường thuật, đừng thay đổi các sơ đồ XML hiện có, hãy tạo ra một sơ đồ mới

```
XElement root = new XElement("Root",
   new XElement("A", "1"),
   new XElement("B", "2"),
   new XElement("C", "3")
);
XElement newRoot = new XElement("Root",
   root.Elements(),
   root.Elements()
);
Console.WriteLine(newRoot);
```

Thêm vào các yếu tố, thuộc tính và các nút trong một cây XML.

VII.5 Làm thế nào để: viết một phương thức axis LINQ to XML.

Bạn có thể viết bằng chính phương pháp axis để xây dựng tập hợp từ sơ đồ XML. Cách tốt nhất để làm điều đó là viết một phương thức mở rộng mà có thể trả về tập hợp các yếu tố hay các thuộc tính. Bạn cũng có thể viết phương thức mở rộng để trả lại tập hợp con của các yếu tố hay các thuộc tính dựa trên các yêu cầu của ứng dụng.

Ví dụ**:** Ví dụ sau sử dụng hai phương pháp mở rộng. Đầu tiên đó là phương pháp GetXPath tổ chức trên đối tượng X và trả về biểu thức Xpath rằng khi được đánh giá sẽ

trả về nút hoặc thuộc tính. Phương pháp thứ hai đó là Find, tổ chức trên yếu tố X. Nó trả về tập hợp của các đối tượng Xatribute và đối tượng Xelement và chứa những văn bản đặc biệt.

Chú ý: Ví dụ sau sử dụng cấu trúc xây dựng trả về yield trong C#. Bởi vì không có khía cạnh nào tương đương trong Visual Basic 2008, đây là một ví dụ chỉ được cung cấp trong C#.

```
public static class MyExtensions
{
   private static string GetQName(XElement xe)
   {
      string prefix = xe.GetPrefixOfNamespace(xe.Name.Namespace);
      if (xe.Name.Namespace == XNamespace.None || prefix == null)
         return xe.Name.LocalName.ToString();
      else
         return prefix + ":" + xe.Name.LocalName.ToString();
   }

   private static string GetQName(XAttribute xa)
   {
      string prefix =
         xa.Parent.GetPrefixOfNamespace(xa.Name.Namespace);
      if (xa.Name.Namespace == XNamespace.None || prefix == null)
         return xa.Name.ToString();
      else
         return prefix + ":" + xa.Name.LocalName;
   }

   private static string NameWithPredicate(XElement el)
   {
      if (el.Parent != null && el.Parent.Elements(el.Name).Count() != 1)
         return GetQName(el) + "[" +
            (el.ElementsBeforeSelf(el.Name).Count() + 1) + "]";
      else
         return GetQName(el);
   }

   public static string StrCat<T>(this IEnumerable<T> source,
      string separator)
```

```
{
    return source.Aggregate(new StringBuilder(),
                (sb, i) => sb
                    .Append(i.ToString())
                    .Append(separator),
                s => s.ToString());
}

public static string GetXPath(this XObject xobj)
{
    if (xobj.Parent == null)
    {
        XDocument doc = xobj as XDocument;
        if (doc != null)
            return ".";
        XElement el = xobj as XElement;
        if (el != null)
            return "/" + NameWithPredicate(el);
        // the XPath data model does not include white space text nodes
        // that are children of a document, so this method returns null.
        XText xt = xobj as XText;
        if (xt != null)
            return null;
        XComment com = xobj as XComment;
        if (com != null)
            return
                "/" +
                (
                    com
                    .Document
                    .Nodes()
                    .OfType<XComment>()
                    .Count() != 1 ?
                    "comment()[" +
                    (com
                    .NodesBeforeSelf()
                    .OfType<XComment>()
                    .Count() + 1) +
                    "]" :
                    "comment()"
                );
```

```
        XProcessingInstruction pi = xobj as XProcessingInstruction;
        if (pi != null)
            return
                "/" +
                (
                    pi.Document.Nodes()
                    .OfType<XProcessingInstruction>()
                    .Count() != 1 ?
                    "processing-instruction()[" +
                    (pi
                    .NodesBeforeSelf()
                    .OfType<XProcessingInstruction>()
                    .Count() + 1) +
                    "]" :
                    "processing-instruction()"
                );
        return null;
    }
    else
    {
        XElement el = xobj as XElement;
        if (el != null)
        {
            return
                "/" +
                el
                .Ancestors()
                .InDocumentOrder()
                .Select(e => NameWithPredicate(e))
                .StrCat("/") +
                NameWithPredicate(el);
        }
        XAttribute at = xobj as XAttribute;
        if (at != null)
            return
                "/" +
                at
                .Parent
                .AncestorsAndSelf()
                .InDocumentOrder()
                .Select(e => NameWithPredicate(e))
```

```
            .StrCat("/") +
            "@" + GetQName(at);
        XComment com = xobj as XComment;
        if (com != null)
            return
                "/" +
                com
                .Parent
                .AncestorsAndSelf()
                .InDocumentOrder()
                .Select(e => NameWithPredicate(e))
                .StrCat("/") +
                (
                    com
                    .Parent
                    .Nodes()
                    .OfType<XComment>()
                    .Count() != 1 ?
                    "comment()[" +
                    (com
                    .NodesBeforeSelf()
                    .OfType<XComment>()
                    .Count() + 1) + "]" :
                    "comment()"
                );
        XCData cd = xobj as XCData;
        if (cd != null)
            return
                "/" +
                cd
                .Parent
                .AncestorsAndSelf()
                .InDocumentOrder()
                .Select(e => NameWithPredicate(e))
                .StrCat("/") +
                (
                    cd
                    .Parent
                    .Nodes()
                    .OfType<XText>()
                    .Count() != 1 ?
```

```
            "text()[" +
            (cd
            .NodesBeforeSelf()
            .OfType<XText>()
            .Count() + 1) + "]" :
            "text()"
        );
    XText tx = xobj as XText;
    if (tx != null)
        return
            "/" +
            tx
            .Parent
            .AncestorsAndSelf()
            .InDocumentOrder()
            .Select(e => NameWithPredicate(e))
            .StrCat("/") +
            (
                tx
                .Parent
                .Nodes()
                .OfType<XText>()
                .Count() != 1 ?
                "text()[" +
                (tx
                .NodesBeforeSelf()
                .OfType<XText>()
                .Count() + 1) + "]" :
                "text()"
            );
    XProcessingInstruction pi = xobj as XProcessingInstruction;
    if (pi != null)
        return
            "/" +
            pi
            .Parent
            .AncestorsAndSelf()
            .InDocumentOrder()
            .Select(e => NameWithPredicate(e))
            .StrCat("/") +
            (
```

```
                    pi
                    .Parent
                    .Nodes()
                    .OfType<XProcessingInstruction>()
                    .Count() != 1 ?
                    "processing-instruction()[" +
                    (pi
                    .NodesBeforeSelf()
                    .OfType<XProcessingInstruction>()
                    .Count() + 1) + "]" :
                    "processing-instruction()"
                );
            return null;
        }
    }

    public static IEnumerable<XObject> Find(this XElement source, string value)
    {
        if (source.Attributes().Any())
        {
            foreach (XAttribute att in source.Attributes())
            {
                string contents = (string)att;
                if (contents.Contains(value))
                    yield return att;
            }
        }
        if (source.Elements().Any())
        {
            foreach (XElement child in source.Elements())
                foreach (XObject s in child.Find(value))
                    yield return s;
        }
        else
        {
            string contents = (string)source;
            if (contents.Contains(value))
                yield return source;
        }
    }
}
```

```
class Program
{
   static void Main(string[] args)
   {
      XElement purchaseOrders = XElement.Load("PurchaseOrders.xml");

      IEnumerable<XObject> subset =
         from xobj in purchaseOrders.Find("1999")
         select xobj;

      foreach (XObject obj in subset)
      {
         Console.WriteLine(obj.GetXPath());
         if (obj.GetType() == typeof(XElement))
            Console.WriteLine(((XElement)obj).Value);
         else if (obj.GetType() == typeof(XAttribute))
            Console.WriteLine(((XAttribute)obj).Value);
      }
   }
}
```

Ví dụ này chỉ ra cách tính các trị số trung gian và có thể được sử dụng trong sắp xếp, lọc và chọn dữ liệu.

Ví dụ: Ví dụ sau sử dụng mệnh để **let**

```
XElement root = XElement.Load("Data.xml");
IEnumerable<decimal> extensions =
   from el in root.Elements("Data")
   let extension = (decimal)el.Element("Quantity") * (decimal)el.Element("Price")
   where extension >= 25
   orderby extension
   select extension;
foreach (decimal ex in extensions)
   Console.WriteLine(ex);
```

VII.6 Cách tạo một tài liệu với Namespaces (LINQ to XML) (C#)

Để tạo một yếu tố hay một thuộc tính mà là xuất hiện trong namespace, bạn có thể tạo nó với một Xname chứa một Xnamespace chứa URI mong muốn cho namespace. Mọi yếu tố hay thuộc tính chứa một Xname và mọi Xname chứa một Xnamespace. Thậm chí một yếu tố không phải là một namespace, các yếy tố Xname vẫn chứa một namespace, đặc tính XNamespace.None. The XName.Namespace được bảo đảm sẽ ko bị Null

Bạn có tùy chọn để tạo ra một thuộc tính namespace trong sơ đồ XML. Những thuộc tính này biểu thị cho namespace. Tạo thuộc tính namespace là cách bạn kiểm sóat được tiền tố của namespace và nếu một namespace là một namespace mặc định.

Để tạo một thuộc tính mà có thể mô tả được namespace mặc định với một tiền tố, bạn có thể tạo một thuộc tính nơi mà namespace của thuộc tính tên trong Xmlns và tên của thuộc tính là tiền tố của namespace. Trị số của thuộc tính là URI của namespace

Nếu sơ đồ XML chứa các yếu tố hau các thuộc tính trong một namespace và nếy không có bất kì thuộc tính nào có thể mô tả namespace trong suốt quá trình LINQ to XML tạo ra các namespace mặc định hoặc tạo ra các namespace không mặc định và gán các tiền tố vào chúng.

Đối tượng Xnamespace được đảm bảo sẽ thu nhỏ nếu có hai đối tượng Xnamespace có chính xác URI tương tự, như thế nó sẽ chia sẻ với nhau trường hợp tương tự.

Ví dụ: Ví dụ sau là ví dụ cho việc tạo tài liệu với một namespace bởi vì tài liệu trong ví dụ này chỉ chứa duy nhất một namespace và không có thuộc tính namespace nào có thể mô tả namespace, LINQ to XML sẽ tạo ra một namespace mặc định.

```
// Create an XML tree in a namespace.
XNamespace aw = "http://www.adventure-works.com";
XElement root = new XElement(aw + "Root",
   new XElement(aw + "Child", "child content")
);
Console.WriteLine(root);
```

Ví dụ sau tạo ra một tài liệu với một namespace. Nó cũng tạo ra một thuộc tính mô tả namespace với một tiền tố namespace. Để tạo ra một thuộc tính mà có thể mô tả namespace với một tiền tố, bạn có thể tạo một thuộc tính nơi mà namespace của tên thuộc tính là Xmlns và tến của thuộc tính là tiền tố namespace. Trị số của thuộc tính này là URI của namespace.

```
// Create an XML tree in a namespace, with a specified prefix
XNamespace aw = "http://www.adventure-works.com";
XElement root = new XElement(aw + "Root",
   new XAttribute(XNamespace.Xmlns + "aw", "http://www.adventure-works.com"),
   new XElement(aw + "Child", "child content")
);
Console.WriteLine(root);
```

Ví dụ sau chỉ ra cách tạo trong tài liệu chứa hai namespace. Một là namespace mặc định, một là namespace với một tiền tố.

Bằng cách bao gồm cả thuộc tính namespace trong yếu tố gố, namespace được sắp xếp theo từng chuỗi cho nên http://www.adventure-works.com là namespace mặc định và www.fourthcoffee.com là được đánh chuỗi với một tiền tố là fc

```
// The http://www.adventure-works.com namespace is forced to be the default
namespace.
XNamespace aw = "http://www.adventure-works.com";
XNamespace fc = "www.fourthcoffee.com";
XElement root = new XElement(aw + "Root",
   new XAttribute("xmlns", "http://www.adventure-works.com"),
   new XAttribute(XNamespace.Xmlns + "fc", "www.fourthcoffee.com"),
   new XElement(fc + "Child",
      new XElement(aw + "DifferentChild", "other content")
   ),
   new XElement(aw + "Child2", "c2 content"),
   new XElement(fc + "Child3", "c3 content")
);
Console.WriteLine(root);
```

Ví dụ sau chỉ ra cách tạo một tài liệu mà có chứa hai namespace với cả tiền tố của namespace.

```
XNamespace aw = "http://www.adventure-works.com";
XNamespace fc = "www.fourthcoffee.com";
XElement root = new XElement(aw + "Root",
   new XAttribute(XNamespace.Xmlns + "aw", aw.NamespaceName),
   new XAttribute(XNamespace.Xmlns + "fc", fc.NamespaceName),
   new XElement(fc + "Child",
      new XElement(aw + "DifferentChild", "other content")
   ),
   new XElement(aw + "Child2", "c2 content"),
   new XElement(fc + "Child3", "c3 content")
);
Console.WriteLine(root);
```

Phương pháp khác cũng cho kết quả tương tự đó là sử dụng tên mở rộng thay vì mô tả và tạo đối tượng Xnamespace.

Phương pháp này thực thi sự liên quan. Tại mỗi thời điểm bạn có thể cho qua một chuỗi kí tự mà chứa các tên trong LINQ to XML, nó phải phân tách tên, tìm ra các namespace được thu nhỏ. Quá trình này chiếm nhiều tài nguyên của CPU. Nếu việc thực thi này quan trọng, bạn có thể muốn sử dụng phương pháp mà có thể mô tả và sử dụng đối tượng Xnamespace một cách rõ ràng.

```
// Create an XML tree in a namespace, with a specified prefix
XElement root = new XElement("{http://www.adventure-works.com}Root",
   new XAttribute(XNamespace.Xmlns + "aw", "http://www.adventure-works.com"),
   new XElement("{http://www.adventure-works.com}Child", "child content")
);Console.WriteLine(root);
```

VII.7 Cách Stream XML Fragments từ một XmlReader

Khi bạn phải xử lý một tập tin XML cỡ lớn, nó có thể không khả thi để tải toàn bộ sơ đồ XML vào trong bộ nhớ. Một trong những cách hiệu quả nhất là sử dụng XmlReader để đọc các đối tượng Xelement để viết cho phương thức axis tùy chỉnh. Một phương thức axis trả lại tập hợp như IEnumerable(T) của Xelement được chỉ ra trong ví dụ trong chủ đề này**.** Trong phương thức axis tùy chỉnh, sau khi bạn tạo ra đoạn XML

bằng cách gọi phương pháp ReadFrom, trả về tập hợp bằng cách sử dụng yield return. Điều này tạo ra các thực hiện chậm ngữ nghĩa cho tùy chỉnh phương thức axis.

Khi bạn tạo một sơ đồ cây XML từ đối tượng XmlReader, XmlReader phải được định vị trong một yếu tố. Phương pháp ReadFrom không trả lại cho tới khi nó đọc được thẻ đóng của các phần tử.

Nếu bạn muốn tạo một sơ đồ cây theo từng phần, bạn có thể chứng minh một XmlReader, vị trí của reader trên node mà bạn muốn chuyển sang sơ đồ cây Xelement và sau đó tạo các đối tượng Xelement.

Ví dụ: Ví dụ này tạo ra một tùy chỉnh phương pháp axis. Bạn có thể truy vấn nó bằng cách sử dụng một truy vấn LINQ. Các tùy chỉnh phương thức axis, StreamRootChildDoc, là một phương pháp được thiết kế đặc biệt để đọc một tài liệu có lặp lại một yếu tố Child.

Chú ý:Ví dụ sau sử dụng cấu trúc yield return của C # bởi vì không có tính năng tương đương trong Visual Basic 2008, ví dụ này chỉ được cung cấp trong C #.

```
static IEnumerable<XElement> StreamRootChildDoc(StringReader stringReader)
{
   using (XmlReader reader = XmlReader.Create(stringReader))
   {
      reader.MoveToContent();
      // Parse the file and display each of the nodes.
      while (reader.Read())
      {
         switch (reader.NodeType)
         {
            case XmlNodeType.Element:
               if (reader.Name == "Child") {
                  XElement el = XElement.ReadFrom(reader) as XElement;
                  if (el != null)
                     yield return el;
               }
               break;
         }
      }
```

```
    }
}

static void Main(string[] args)
{
    string markup = @"<Root>
      <Child Key=""01"">
        <GrandChild>aaa</GrandChild>
      </Child>
      <Child Key=""02"">
        <GrandChild>bbb</GrandChild>
      </Child>
      <Child Key=""03"">
        <GrandChild>ccc</GrandChild>
      </Child>
    </Root>";

    IEnumerable<string> grandChildData =
        from el in StreamRootChildDoc(new StringReader(markup))
        where (int)el.Attribute("Key") > 1
        select (string)el.Element("GrandChild");

    foreach (string str in grandChildData) {
        Console.WriteLine(str);
    }
}
```

VII.8 Cách tạo một sơ đồ (Tree) từ một XmlReader.

Chủ đề này chỉ cho bạn cách tạo một sơ đồ trực tiếp từ một XmlReader. Để tạo một Xelement từ một XmlReader, bạn phải định vị XmlReader trên một node. XmlReader sẽ bỏ qua những thành phần và xử lý các hướng dẫn, nhưng nếu XmlReader được định vị trên node văn bản thì sẽ xuất hiện một lỗi. Để tránh những lỗi đó thì bạn luôn luôn phải định vị XmlReader trên element trước khi bạn tạo một sơ đồ XML (Cây XML) từ XmlReader.

Ví dụ: Mã sau tạo ra một đối tượng T:System.Xml.XmlReader và sau đó đọc các node cho đến khi nó tìm ra element node đầu tiên và tải đối tượng XElement.

```
XmlReader r = XmlReader.Create("books.xml");
while (r.NodeType != XmlNodeType.Element)
   r.Read();
XElement e = XElement.Load(r);
Console.WriteLine(e);
```

VII.9 Thay đổi cây XML tròn bộ nhớ trong so với Functional Construction (LINQ to XML)

Việc thay đổi một cây XML tại một vị trí là cách tiếp cận truyền thống để thay đổi hình dạng của một tài liệu XML. Một ứng dụng nổi bật tải một tài liệu vào để lưu trữ như DOM hoặc LINQ to XML; sử dụng một giao diện lập trình để chèn nodes, xóa nodes, hoặc thay đổi nội dung của nodes; và sau đó lưu là một tập tin XML để chuyển nó qua mạng máy tính, LINQ to XML cho phép cách tiếp cận đó mà có ích trong nhiều kịch bản: *functional construction*.. *functional construction* làm thay đổi các dữ liệu như là một vấn đề của quá trình chuyển đổi. Nếu bạn có thể biểu diễn dữ liệu hay chuyển nó từ mẫu này qua mẫu khác mà cho một kết quả tương tự như khi bạn lấy dữ liệu và thao tác nó sang một dạng khác. Điều quan trọng đối với phương pháp *functional construction* đó là bỏ qua kết quả của truy vấn với XDocument và XElement constructors.

Trong nhiều trường hợp, bạn có thể viết các mã truyền dữ liệu trong khoảng thời gian ngắn mà nó có thể tao tác các kho lưu trữ dữ liệu và mã đó sẽ mạnh hơn và dễ dàng duy trì hơn. Trong những trường hợp này, mặc dù các phương pháp truyền dữ liệu có thể tốn nhiều tài nguyên xử lý của máy tính ,đó là một cách hiệu quả hơn để sửa đổi dữ liệu. Nếu một chuyên gia phát triển ứng dụng đã làm quen với phương pháp này thì trong nhiều trường hợp sẽ hiểu hơn. Và cũng sẽ dễ dàng hơn để tìm mã mà có thể thay đổi mỗi phần của cây.

Chủ đề này cung cấp một ví dụ mà từ đó là triển khai thực hiện với cả hai phương pháp tiếp cận. Các phương pháp tiếp cận nơi bạn sửa đổi một XML cây tại chỗ là quen thuộc để có thêm nhiều DOM lập trình, trong khi mã văn bản bằng cách sử dụng chức năng, cách tiếp cận có thể hình không quen thuộc với một người phát triển, không có gì

chưa hiểu rằng cách tiếp cận. Nếu bạn có phải chỉ làm cho một nhỏ để sửa đổi, bổ sung một XML cây lớn, các phương pháp tiếp cận nơi bạn sửa đổi một cây ở nơi trong nhiều trường hợp CPU sẽ mất ít thời gian.

VII.10 Chuyển đổi thuộc tính vào các phần tử.

Đối với ví dụ này, giả sử bạn muốn sửa đổi các tài liệu XML đơn giản sau đó, để trở thành yếu tố thuộc tính. Chủ đề này trình bày các truyền thống đầu tiên tại-chỗ sửa đổi cách tiếp cận. Sau đó nó cho thấy các chức năng xây dựng phương pháp tiếp cận.

VII.10.1 Chỉnh sửa một cây XML.

Bạn có thể viết một số mã để tạo từ các thành phần thuộc tính, và sau đó xóa các thuộc tính, như sau.

```
XElement root = XElement.Load("Data.xml");
foreach (XAttribute att in root.Attributes()) {
   root.Add(new XElement(att.Name, (string)att));
}
root.Attributes().Remove();
Console.WriteLine(root);
```

VII.10.2 Cách tiếp cận Functional Construction.

Ngược lại, một phương pháp tiếp cận bao gồm các chức năng mã để tạo thành một cây mới, chọn và lựa chọn thành phần và các thuộc tính từ các nguồn gốc cây, và chuyển chúng trong trường hợp thích hợp khi chúng được đưa vào các cây mới. Phương pháp tiếp cận các chức năng như sau:

```
XElement root = XElement.Load("Data.xml");
XElement newTree = new XElement("Root",
   root.Element("Child1"),
   from att in root.Attributes()
   select new XElement(att.Name, (string)att)
);
Console.WriteLine(newTree);
```

Ví dụ này kết quả đầu ra trong cùng một XML ví dụ như là người đầu tiên. Tuy nhiên, thông báo rằng bạn có thể thực sự xem các kết quả của các cấu trúc XML mới trong chức năng, cách tiếp cận. Bạn có thể xem thông sáng tạo của người dân gốc nguyên

tố, rằng hai bên nguyên tố Child1 từ nguồn gốc cây, và mã chuyển biến rằng thuộc tính từ cây nguồn để yếu tố mới trong cây.

Các chức năng như trong trường hợp này không phải là bất kỳ ngắn hơn là người đầu tiên ví dụ, và nó không phải là bất kỳ thực sự đơn giản. Tuy nhiên, nếu bạn có nhiều thay đổi để làm cho một XML cây, các phương pháp tiếp cận không chức năng sẽ trở nên khá phức tạp và đôi chút mơ hồ. Ngược lại, khi sử dụng các chức năng, cách tiếp cận, bạn chỉ cần nhấn chuột mong muốn XML, đưa vào truy vấn và biểu thức như là thích hợp, để kéo trong nội dung mong muốn. Các mã sản lượng chức năng, cách tiếp cận được dễ dàng hơn để duy trì.

Chú ý rằng trong trường hợp này, cách tiếp cận các chức năng có lẽ sẽ không thực hiện khá cũng như các cây. Các vấn đề chính là cách tiếp cận các chức năng tạo thêm ngắn sống các đối tượng. Tuy nhiên, sự cân bằng là một hiệu quả nhất nếu sử dụng các chức năng cho phép các phương pháp tiếp cận lớn hơn cho các lập trình hiệu quả.

Đây là một ví dụ đơn giản, nhưng nó phục vụ để hiển thị sự khác biệt trong triết lý giữa hai phương pháp tiếp cận. Phương pháp tiếp cận các chức năng sản xuất đạt sản lượng lớn cho việc chuyển các tài liệu XML lớn hơn.

Chú ý rằng trong trường hợp này, cách tiếp cận các chức năng có lẽ sẽ không thực hiện khá cũng như các cây sử dụng cách tiếp cận bằng tay. Các vấn đề chính là cách tiếp cận các chức năng tạo thêm ngắn sống các đối tượng. Tuy nhiên, sự cân bằng là một hiệu quả nhất nếu sử dụng các chức năng cho phép các phương pháp tiếp cận lớn hơn cho các lập trình hiệu quả

VII.10.3 Removing Elements, Attributes, and Nodes from an XML Tree

Bạn có thể sửa đổi một XML cây, loại bỏ các phần tử, thuộc tính, và các loại nodes. Xoá bỏ một nguyên tố hoặc một thuộc tính từ một tài liệu XML là đơn giản. Tuy nhiên, khi loại bỏ các bộ sưu tập của yếu tố hay thuộc tính, trước tiên bạn cần một bộ sưu tập vào một danh sách, và sau đó xóa các yếu tố hoặc thuộc tính từ danh sách. Cách tiếp cận tốt nhất là sử dụng các phương thức mở rộng, mà sẽ làm việc này cho bạn.

Lý do chính cho công việc này là hầu hết các bộ sưu tập của bạn lấy từ một cây đang có yielded XML bằng cách sử dụng thực hiện làm chậm. Nếu bạn không làm chúng xuất hiện đầu tiên vào một danh sách, hoặc nếu bạn không sử dụng các phương pháp mở rộng có thể gặp một lỗi của một số lớp.

Những phương thức sau đây xoá bỏ nodes và các thuộc tính từ một cây XML

[M:System.Xml.Linq.XAttribute.Remove()]	Loại bỏ một XAttribute từ cha mẹ của nó.
[M:System.Xml.Linq.XContainer.RemoveNodes()]	Loại bỏ các nodes con từ một XContainer.
XElement.RemoveAll	Loại bỏ nội dung và các thuộc tính từ một XElement.
XElement.RemoveAttributes	Loại bỏ các thuộc tính của một XElement.
XElement.SetAttributeValue	Nếu bạn bỏ qua cho các giá trị null, sau đó loại bỏ các thuộc tính.
XElement.SetElementValue	Nếu bạn bỏ qua cho các giá trị null, sau đó loại bỏ các phần tử con.
XNode.Remove	Loại bỏ một XNode từ cha mẹ của nó.
Extensions.Remove	Loại bỏ tất cả các thuộc tính hay phần tử trong nguồn thu từ một loạt phần tử gốc.

Mô tả.

Ví dụ này đã chứng minh ba phương pháp tiếp cận để loại bỏ các phần tử. Trước tiên, nó loại bỏ một phần tử. Thứ hai, nó truy một tập hợp các phần tử, cho hiện chúng bằng cách sử dụng Enumerable.ToList (TSource), và loại bỏ những bộ sưu tập. Cuối cùng, nó truy xuất một tập hợp các phần tử và loại bỏ chúng bằng cách sử dụng các phương pháp mở rộng.

Code

```
XElement root = XElement.Parse(@"<Root>
   <Child1>
      <GrandChild1/>
      <GrandChild2/>
      <GrandChild3/>
   </Child1>
   <Child2>
      <GrandChild4/>
      <GrandChild5/>
      <GrandChild6/>
   </Child2>
   <Child3>
      <GrandChild7/>
      <GrandChild8/>
      <GrandChild9/>
   </Child3>
</Root>");
root.Element("Child1").Element("GrandChild1").Remove();
root.Element("Child2").Elements().ToList().Remove();
root.Element("Child3").Elements().Remove();
Console.WriteLine(root);
```

Ví dụ: Để ý rằng các yếu tố đầu tiên, cháu ngoại đã được xoá khỏi Child1. Cháu tất cả các yếu tố đã được xoá khỏi Child2 và từ Child3. Khi bạn gọi một trong những phương pháp mà trở lại IEnumerable (T) của XElement, bạn có thể lọc trên các yếu tố tên.

```
XElement po = XElement.Load("PurchaseOrder.xml");
IEnumerable<XElement> items =
   from el in po.Descendants("ProductName")
   select el;
foreach(XElement prdName in items)
   Console.WriteLine(prdName.Name + ":" + (string) prdName);
```

Các phương pháp khác mà trả về IEnumerable (T) của XElement bộ sưu tập theo cùng một khuôn mẫu. Chữ ký của họ là tương tự với yếu tố và Descendants. Sau đây là danh sách đầy đủ các phương pháp đó có phương pháp tương tự chữ ký

- Ancestors
- Descendants

- Elements
- ElementsAfterSelf
- ElementsBeforeSelf
- AncestorsAndSelf
- DescendantsAndSelf

Các phần tử.

- AncestorsAndSelf
- Ancestors
- Descendants
- ElementsAfterSelf
- ElementsBeforeSelf
- DescendantsAndSelf

.

Ví dụ sau hiển thị cùng một truy vấn cho XML là trong một namespace.

```
XNamespace aw = "http://www.adventure-works.com";
XElement po = XElement.Load("PurchaseOrderInNamespace.xml");
IEnumerable<XElement> items =
   from el in po.Descendants(aw + "ProductName")
   select el;
foreach (XElement prdName in items)
   Console.WriteLine(prdName.Name + ":" + (string)prdName);
```

VII.10.4 Làm thế nào để: Lọc trên một Tùy chọn Element.

Ví dụ: Đôi khi bạn muốn lọc cho một yếu tố mặc dù bạn không chắc chắn nó tồn tại trong tài liệu XML của bạn. Nên việc tìm kiếm được thực hiện như vậy là nếu các yếu tố, không có gì đặc biệt không có các yếu tố con, bạn không kích hoạt một tham chiếu null ngoại lệ bằng cách lọc cho nó. Trong ví dụ sau, các yếu tố Child5 không có một yếu tố Loại hình child, nhưng vẫn còn thi các truy vấn một cách chính xác.

```
XElement root = XElement.Parse(@"<Root>
  <Child1>
    <Text>Child One Text</Text>
    <Type Value=""Yes""/>
  </Child1>
  <Child2>
    <Text>Child Two Text</Text>
    <Type Value=""Yes""/>
  </Child2>
  <Child3>
```

```
    <Text>Child Three Text</Text>
    <Type Value=""No""/>
  </Child3>
  <Child4>
    <Text>Child Four Text</Text>
    <Type Value=""Yes""/>
  </Child4>
  <Child5>
    <Text>Child Five Text</Text>
  </Child5>
</Root>");
var cList =
    from typeElement in root.Elements().Elements("Type")
    where (string)typeElement.Attribute("Value") == "Yes"
    select (string)typeElement.Parent.Element("Text");
foreach(string str in cList)
    Console.WriteLine(str);
```

Ví dụ sau hiển thị cùng một truy vấn cho XML là trong một namespace.

```
XElement root = XElement.Parse(@"<Root xmlns='http://www.adatum.com'>
  <Child1>
    <Text>Child One Text</Text>
    <Type Value=""Yes""/>
  </Child1>
  <Child2>
    <Text>Child Two Text</Text>
    <Type Value=""Yes""/>
  </Child2>
  <Child3>
    <Text>Child Three Text</Text>
    <Type Value=""No""/>
  </Child3>
  <Child4>
    <Text>Child Four Text</Text>
    <Type Value=""Yes""/>
  </Child4>
  <Child5>
    <Text>Child Five Text</Text>
  </Child5>
</Root>");
XNamespace ad = "http://www.adatum.com";
```

```
var cList =
   from typeElement in root.Elements().Elements(ad + "Type")
   where (string)typeElement.Attribute("Value") == "Yes"
   select (string)typeElement.Parent.Element(ad + "Text");
foreach (string str in cList)
   Console.WriteLine(str);
```

VII.10.5 Làm thế nào để: Tìm một đơn Descendant rõ Phương thức sử dụng.

Bạn có thể sử dụng một rõ phương thức axis để nhanh chóng viết mã để tìm một phần tử có tên duy nhất. Kỹ thuật này là đặc biệt hữu ích khi bạn muốn tìm một phần tử con với một tên cụ thể. Bạn có thể viết mã để điều hướng tới các yếu tố mong muốn, nhưng thường là nhanh hơn và dễ dàng hơn để viết mã bằng cách sử dụng rõ axis.

Ví dụ:Ví dụ này sử dụng các nhà điều hành đầu tiên đạt tiêu chuẩn yêu cầu tìm kiếm

```
XElement root = XElement.Parse(@"<Root>
  <Child1>
    <GrandChild1>GC1 Value</GrandChild1>
  </Child1>
  <Child2>
    <GrandChild2>GC2 Value</GrandChild2>
  </Child2>
  <Child3>
    <GrandChild3>GC3 Value</GrandChild3>
  </Child3>
  <Child4>
    <GrandChild4>GC4 Value</GrandChild4>
  </Child4>
</Root>");
string grandChild3 = (string)
   (from el in root.Descendants("GrandChild3")
   select el).First();
Console.WriteLine(grandChild3);
```

VII.10.6 Làm thế nào để: Tìm tất cả các Nodes trong một Namespace.

Bạn có thể lọc trên namespace của mỗi phần tử hay thuộc tính để tìm tất cả các nodes trong đó đặc biệt là namespace.

Ví dụ sau tạo ra một XML cây với namepace. Sau đó nó lặp lại thông qua các cây và các bản in tranh tên của tất cả các phần tử và các thuộc tính trong một trong những namespace.

```
string markup = @"<aw:Root xmlns:aw='http://www.adventure-works.com' 
xmlns:fc='www.fourthcoffee.com'>
  <fc:Child1>abc</fc:Child1>
  <fc:Child2>def</fc:Child2>
  <aw:Child3>ghi</aw:Child3>
  <fc:Child4>
    <fc:GrandChild1>jkl</fc:GrandChild1>
    <aw:GrandChild2>mno</aw:GrandChild2>
  </fc:Child4>
</aw:Root>";
XElement xmlTree = XElement.Parse(markup);
Console.WriteLine("Nodes in the http://www.adventure-works.com namespace");
IEnumerable<XElement> awElements =
    from el in xmlTree.Descendants()
    where el.Name.Namespace == "http://www.adventure-works.com"
    select el;
foreach (XElement el in awElements)
    Console.WriteLine(el.Name.ToString());
```

Các tệp tin XML được truy cập bằng cách truy vấn chứa các thứ tự trong hai namespace khác nhau. Các truy vấn tạo ra một cây mới chỉ với các yếu tố trong một trong những namespace.

```
XDocument cpo = XDocument.Load("ConsolidatedPurchaseOrders.xml");
XNamespace aw = "http://www.adventure-works.com";
XElement newTree = new XElement("Root",
    from el in cpo.Root.Elements()
    where el.Name.Namespace == aw
    select el
);
Console.WriteLine(newTree);
```

VII.10.7 Làm thế nào để: Tìm một phần tử với phần tử cụ thể.

Chủ đề này cho thấy như thế nào để tìm một phần tử mà phần tử có một child với một giá trị cụ thể.

Ví dụ khi thử nghiệm thấy rằng có một phần tử CommandLine phần tử con với giá trị của Examp2.EXE".

```
XElement root = XElement.Load("TestConfig.xml");
IEnumerable<XElement> tests =
   from el in root.Elements("Test")
   where (string)el.Element("CommandLine") == "Examp2.EXE"
   select el;
foreach (XElement el in tests)
   Console.WriteLine((string)el.Attribute("TestId"));
```

VII.10.8 Làm thế nào để: Tìm một Element với một thuộc tính cụ thể.

Chủ đề này cho thấy làm thế nào để tìm một yếu tố mà có một thuộc tính có một giá trị cụ thể. Ví dụ cho thấy như thế nào để tìm dịa chỉ phần tử.

```
XElement root = XElement.Load("PurchaseOrder.xml");
IEnumerable<XElement> address =
   from el in root.Elements("Address")
   where (string)el.Attribute("Type") == "Billing"
   select el;
foreach (XElement el in address)
   Console.WriteLine(el);
```

Ví dụ này sử dụng các tài liệu XML sau đây:

```
XElement root = XElement.Load("PurchaseOrderInNamespace.xml");
XNamespace aw = "http://www.adventure-works.com";
IEnumerable<XElement> address =
   from el in root.Elements(aw + "Address")
   where (string)el.Attribute(aw + "Type") == "Billing"
   select el;
foreach (XElement el in address)
   Console.WriteLine(el);
```

VII.10.9 Làm thế nào để: Tìm Descendants với một cụ thể Element namespace.

Đôi khi bạn muốn tìm thấy tất cả descendants với một cái tên riêng. Bạn có thể viết mã để lặp qua tất cả các phần tử con, nhưng nó được dễ dàng hơn khi sử dụng rõ axis.

Ví dụ: Ví dụ sau cho thấy như thế nào để tìm descendants dựa trên các yếu tố tên

```
XElement root = XElement.Parse(@"<root>
  <para>
    <r>
      <t>Some text </t>
    </
```

VII.10.10 Làm thế nào để: Tạo văn bản từ tập tin XML.

Ví dụ này cho thấy như thế nào để tạo ra một giá trị phân cách bằng dấu phẩy (CSV) tập tin từ một tệp tin XML

Ví dụ: Phiên bản C # của ví dụ này sử dụng cú pháp phương thức tổng hợp và các nhà điều hành để tạo ra một tệp CSV từ một tài liệu XML trong một biểu thức đơn lẻ.

Ví dụ này sử dụng các tài liệu XML sau đây:

```
XElement custOrd = XElement.Load("CustomersOrders.xml");
string csv =
    (from el in custOrd.Element("Customers").Elements("Customer")
    select
        String.Format("{0},{1},{2},{3},{4},{5},{6},{7},{8},{9}{10}",
            (string)el.Attribute("CustomerID"),
            (string)el.Element("CompanyName"),
            (string)el.Element("ContactName"),
            (string)el.Element("ContactTitle"),
            (string)el.Element("Phone"),
            (string)el.Element("FullAddress").Element("Address"),
            (string)el.Element("FullAddress").Element("City"),
            (string)el.Element("FullAddress").Element("Region"),
            (string)el.Element("FullAddress").Element("PostalCode"),
            (string)el.Element("FullAddress").Element("Country"),
            Environment.NewLine
        )
    )
    .Aggregate(
        new StringBuilder(),
        (sb, s) => sb.Append(s),
        sb => sb.ToString()
    );
Console.WriteLine(csv);
```

VII.10.11 Làm thế nào để: tạo ra hệ đẳng cấp bằng cách sử dụng nhóm.

Ví dụ này cho thấy như thế nào để nhóm dữ liệu, và sau đó tạo ra XML dựa trên các nhóm.

Ví dụ:Ví dụ này lần đầu tiên một nhóm dữ liệu theo thể loại, sau đó mới tạo ra một tệp tin XML, trong đó XML Hierarchy phản ánh các nhóm.

Ví dụ này sử dụng các tài liệu XML sau đây:

```
XElement doc = XElement.Load("Data.xml");
var newData =
   new XElement("Root",
      from data in doc.Elements("Data")
      group data by (string)data.Element("Category") into groupedData
      select new XElement("Group",
         new XAttribute("ID", groupedData.Key),
         from g in groupedData
         select new XElement("Data",
            g.Element("Quantity"),
            g.Element("Price")
         )
      )
   );
Console.WriteLine(newData);
```

VII.10.12 Làm thế nào để: Join hai bộ sưu tập.

Một phần tử hoặc thuộc tính trong một tài liệu XML đôi khi có thể tham giao các phần tử khác, hay thuộc tính. Ví dụ, các khách hàng đặt hàng và các tài liệu XML có chứa một danh sách các khách hàng và một danh sách các đơn đặt hàng. Tất cả các yếu tố khách hàng có chứa một thuộc tính ID khách hàng. Tất cả các yếu tố đơn hàng có chứa một yếu tố ID khách hàng. ID khách hàng các yếu tố trong mỗi đơn hàng tham chiếu đến các thuộc tính ID khách hàng trong một khách hang.

Chủ đề mẫu XSD File: Khách hàng Đặt hàng và chứa một XSD có thể được sử dụng để xác minh tài liệu này. Nó dùng xs: chìa khóa và xs: keyref các tính năng của XSD để thiết lập các thuộc tính ID khách hàng của các khách hàng là một yếu tố chủ

chốt, và để thiết lập một mối quan hệ giữa các yếu tố ID khách hàng ở mỗi đơn hàng và các yếu tố ID khách hàng thuộc tính trong mỗi yếu tố khách hàng. Với LINQ to XML, bạn có thể tận dụng các mối quan hệ này bằng cách sử dụng mệnh đề **join**.

Lưu ý rằng vì không có chỉ mục có sẵn, như vậy join sẽ có hiệu suất kém tại thời gian chạy.

Ví dụ: Ví dụ sau join các yếu tố khách hàng đến yếu tố đơn hàng, và tạo ra một tài liệu XML mới trong đó bao gồm các yếu tố CompanyName trong đơn đặt hàng.

Trước khi thi hành các truy vấn, ví dụ validates rằng các tài liệu tuân thủ với các schema trong mẫu XSD File: Khách hàng và Đặt hàng. Điều này đảm bảo rằng mệnh đề join sẽ luôn luôn làm việc.

Truy vấn này trước tiên trả về ất cả các phần tử khách hàng, và sau đó join chúng vào yếu tố đơn hàng. Nó chỉ lựa chọn các đơn đặt hàng cho khách hàng với một ID khách hàng lớn hơn "K". Sau đó nó là một dự án mới Đặt hàng yếu tố có chứa các thông tin khách hàng trong phạm vi của mỗi đơn đặt hàng.

Ví dụ này sử dụng các tài liệu XML sau đây: Lưu ý rằng join trong thời trang này sẽ không thực hiện rất tốt. Tham gia được thực hiện thông qua một tìm kiếm tuyến tính. Hiện không có trong bảng băm hoặc lập chỉ mục để giúp đỡ với hiệu quả hoạt động.

```
XmlSchemaSet schemas = new XmlSchemaSet();
schemas.Add("", "CustomersOrders.xsd");

Console.Write("Attempting to validate, ");
XDocument custOrdDoc = XDocument.Load("CustomersOrders.xml");

bool errors = false;
custOrdDoc.Validate(schemas, (o, e) =>
                {
                   Console.WriteLine("{0}", e.Message);
                   errors = true;
                });
Console.WriteLine("custOrdDoc {0}", errors ? "did not validate" : "validated");
```

```
if (!errors)
{
   // Join customers and orders, and create a new XML document with
   // a different shape.

   // The new document contains orders only for customers with a
   // CustomerID > 'K'
   XElement custOrd = custOrdDoc.Element("Root");
   XElement newCustOrd = new XElement("Root",
      from c in custOrd.Element("Customers").Elements("Customer")
      join o in custOrd.Element("Orders").Elements("Order")
               on (string)c.Attribute("CustomerID") equals
                  (string)o.Element("CustomerID")
      where ((string)c.Attribute("CustomerID")).CompareTo("K") > 0
      select new XElement("Order",
         new XElement("CustomerID", (string)c.Attribute("CustomerID")),
         new XElement("CompanyName", (string)c.Element("CompanyName")),
         new XElement("ContactName", (string)c.Element("ContactName")),
         new XElement("EmployeeID", (string)o.Element("EmployeeID")),
         new XElement("OrderDate", (DateTime)o.Element("OrderDate"))
      )
   );
   Console.WriteLine(newCustOrd);
}
```

VII.10.13 Làm thế nào để: Load XML từ một tệp.

Chủ đề này cho thấy như thế nào để tải XML từ một URI bằng cách sử dụng phương thức XElement.Load

Ví dụ:Ví dụ sau cho thấy như thế nào để tải một XML tài liệu từ một tập tin. Ví dụ sau tải books.xml và đầu ra của cây XML để bàn điều khiển.
Ví dụ này sử dụng các tài liệu XML sau đây:

```
XElement booksFromFile = XElement.Load(@"books.xml");
Console.WriteLine(booksFromFile);
```

VII.11 Sửa đổi XML Trees.

LINQ to XML là một trong bộ nhớ lưu trữ cho một cây XML. Sau khi bạn tải hoặc phân tích cú pháp XML một cây từ một nguồn, LINQ to XML để cho phép bạn sửa đổi

cây, và sau đó xauats bản các cây, có lẽ nó để tiết kiệm một tập tin hoặc gửi nó cho một máy chủ từ xa.

Khi bạn sửa đổi một cây ở chỗ, bạn sử dụng một số phương thức, chẳng hạn như thêm. Tuy nhiên, đó là một cách tiếp cận để sử dụng chức năng xây dựng để tạo ra một cây mới với một hình dạng khác nhau. Tùy thuộc vào loại thay đổi mà bạn cần phải làm để cây XML của bạn, và tùy thuộc vào kích cỡ của cây, cách tiếp cận này có thể được thêm mạnh mẽ và dễ dàng hơn để phát triển. Chủ đề đầu tiên trong phần này so sánh hai phương pháp tiếp cận.

VII.11.1 Làm thế nào để: Viết một truy vấn mà các phần tử dựa trên bối cảnh.

Đôi khi bạn có thể viết một truy vấn mà sự lựa chọn dựa trên các yếu tố bối cảnh của họ. Bạn có thể muốn lọc dựa trên trước hay các yếu tố sau đây A / C / E . Bạn có thể muốn lọc dựa trên con hoặc các phần tử.

Bạn có thể làm được điều này bằng cách viết một truy vấn và sử dụng các kết quả của các truy vấn trong mệnh đề where. Nếu bạn có bài kiểm tra đầu tiên khác null, và sau đó thử nghiệm các giá trị, đó là thuận tiện hơn để truy vấn làm việc trong một mệnh đề **let**, và sau đó sử dụng các kết quả trong mệnh đề **where**.

Ví dụ:Ví dụ sau p chọn tất cả các yếu tố đó là ngay lập tức, theo sau là một yếu tố ul

```
XElement doc = XElement.Parse(@"<Root>
   <p id=""1""/>
   <ul>abc</ul>
   <Child>
      <p id=""2""/>
      <notul/>
      <p id=""3""/>
      <ul>def</ul>
      <p id=""4""/>
   </Child>
   <Child>
      <p id=""5""/>
      <notul/>
      <p id=""6""/>
      <ul>abc</ul>
```

```
      <p id=""7""/>
   </Child>
</Root>");

IEnumerable<XElement> items =
   from e in doc.Descendants("p")
   let z = e.ElementsAfterSelf().FirstOrDefault()
   where z != null && z.Name.LocalName == "ul"
   select e;

foreach (XElement e in items)
   Console.WriteLine("id = {0}", (string)e.Attribute("id"));
```

Ví dụ sau hiển thị cùng một truy vấn cho XML là trong một namespace.

```
XElement doc = XElement.Parse(@"<Root xmlns='http://www.adatum.com'>
   <p id=""1""/>
   <ul>abc</ul>
   <Child>
      <p id=""2""/>
      <notul/>
      <p id=""3""/>
      <ul>def</ul>
      <p id=""4""/>
   </Child>
   <Child>
      <p id=""5""/>
      <notul/>
      <p id=""6""/>
      <ul>abc</ul>
      <p id=""7""/>
   </Child>
</Root>");

XNamespace ad = "http://www.adatum.com";

IEnumerable<XElement> items =
   from e in doc.Descendants(ad + "p")
   let z = e.ElementsAfterSelf().FirstOrDefault()
   where z != null && z.Name == ad.GetName("ul")
   select e;

foreach (XElement e in items)
```

```
   Console.WriteLine("id = {0}", (string)e.Attribute("id"));
```

VII.11.2 Làm thế nào để: Viết truy vấn với lọc phức tạp.

Đôi khi bạn muốn viết LINQ để truy vấn XML voeis các bộ lọc phức tạp. Ví dụ, bạn có thể tìm tất cả các yếu tố đó có một yếu tố con bằng một cái tên riêng và giá trị.

Ví dụ: Ví dụ này cho thấy như thế nào để tìm tất cả các phần tử PurchaseOrder rằng có một Address phần tử đó có một nghề thuộc tính là " Shipping ". Nó sử dụng một truy vấn lồng trong mẹng đề **where**, và các nhà điều hành trả về giá trị true và bất kỳ giá trị trả về là true, nếu bộ sưu tập có bất kỳ phần tử nào trong nó.

Sau đây là đoạn ví dụ:

```
XElement root = XElement.Load("PurchaseOrders.xml");
IEnumerable<XElement> purchaseOrders =
   from el in root.Elements("PurchaseOrder")
   where
      (from add in el.Elements("Address")
      where
         (string)add.Attribute("Type") == "Shipping" &&
         (string)add.Element("State") == "NY"
      select add)
      .Any()
   select el;
foreach (XElement el in purchaseOrders)
   Console.WriteLine((string)el.Attribute("PurchaseOrderNumber"));
```

Ví dụ sau hiển thị cùng một truy vấn cho XML là trong một namespace. Ví dụ này sử dụng các tài liệu XML sau đây:

```
XElement root = XElement.Load("PurchaseOrdersInNamespace.xml");
XNamespace aw = "http://www.adventure-works.com";
IEnumerable<XElement> purchaseOrders =
   from el in root.Elements(aw + "PurchaseOrder")
   where
      (from add in el.Elements(aw + "Address")
       where
          (string)add.Attribute(aw + "Type") == "Shipping" &&
          (string)add.Element(aw + "State") == "NY"
       select add)
      .Any()
```

```
   select el;
foreach (XElement el in purchaseOrders)
   Console.WriteLine((string)el.Attribute(aw + "PurchaseOrderNumber"));
```

VII.11.3 Làm thế nào để: Truy vấn LINQ để sử dụng XML xpath.

Chủ đề này giới thiệu các phương thức mở rộng cho phép bạn truy vấn một XML bằng cách dùng xpath cây.

Trừ khi bạn có một lý do rất cụ thể cho các câu hỏi bằng cách sử dụng xpath, chẳng hạn như sử dụng rộng rãi của thừa mã, bằng cách sử dụng xpath với LINQ to XML không phải là để khuyến khích. Xpath truy vấn sẽ không thực hiện cũng như LINQ để truy vấn XML.

Ví dụ:Ví dụ sau tạo một XML cây nhỏ và sử dụng XPathSelectElements để chọn một bộ các phần tử.

```
XElement root = new XElement("Root",
   new XElement("Child1", 1),
   new XElement("Child1", 2),
   new XElement("Child1", 3),
   new XElement("Child2", 4),
   new XElement("Child2", 5),
   new XElement("Child2", 6)
);
IEnumerable<XElement> list = root.XPathSelectElements("./Child2");
foreach (XElement el in list)
   Console.WriteLine(el);
```

VII.11.4 Làm thế nào để: Xắp sếp các phần tử.

Đoạn mã sau cho thấy việc xắp sếp các phần tử như thế nào.

```
XElement root = XElement.Load("Data.xml");
IEnumerable<decimal> prices =
   from el in root.Elements("Data")
   let price = (decimal)el.Element("Price")
   orderby price
```

```
   select price;
foreach (decimal el in prices)
   Console.WriteLine(el);
```

VII.11.5 Làm thế nào để: xắp sếp các phần tử có nhiều khóa.

```
XElement co = XElement.Load("CustomersOrders.xml");
var sortedElements =
   from c in co.Element("Orders").Elements("Order")
   orderby (string)c.Element("ShipInfo").Element("ShipPostalCode"),
         (DateTime)c.Element("OrderDate")
   select new {
      CustomerID = (string)c.Element("CustomerID"),
      EmployeeID = (string)c.Element("EmployeeID"),
      ShipPostalCode = (string)c.Element("ShipInfo").Element("ShipPostalCode"),
      OrderDate = (DateTime)c.Element("OrderDate")
   };
foreach (var r in sortedElements)
   Console.WriteLine("CustomerID:{0}        EmployeeID:{1}        ShipPostalCode:{2}
OrderDate:{3:d}",    r.CustomerID, r.EmployeeID, r.ShipPostalCode, r.OrderDate);
```

VII.11.6 Làm thế nào để: Xắp sếp theo chính sách thực hiện chuyển đổi của tài liệu XML lớn.

Đôi khi, bạn cần phải chuyển đổi lớn các file XML, và viết ứng dụng của bạn đó, để dấu vết bộ nhớ của ứng dụng là có thể đoán trước. Nếu bạn cố gắng để đưa đến một XML cây với một tệp tin XML rất lớn, bộ nhớ của bạn sẽ được sử dụng tương ứng để tỷ lệ vói kích thước của tập tin (có nghĩa là, quá nhiều). Vì vậy, bạn nên sử dụng một kỹ thuật xắp sếp theo chính sách, thay vì sử dụng các kỹ thuật khác.

Kỹ thuật sắp xếp theo chính sách áp dụng tốt nhất trong tình huống mà bạn cần để xử lý các nguồn tài liệu chỉ một lần, và bạn có thể xử lý các yếu tố trong tài liệu thứ tự. Truy vấn vận hành một tiêu chuẩn nhất định, chẳng hạn như OrderBy,lặp lại nguồn của mình, thu thập tất cả các dữ liệu, phân loại nó, và sau đó cuối cùng trả về phần tử đầu tiên trong dãy. Lưu ý rằng nếu bạn sử dụng một truy vấn mà nhà điều hành cho hiện ra nguồn trước khi trả về mục đầu tiên, bạn sẽ không giữ lại một dấu chân nhỏ trong bộ nhớ cho ứng dụng của bạn.

Ở đây có hai phương thức tiếp cận chính. Một cách tiếp cận là để sử dụng những đặc điểm của XStreamingElement. Một cách tiếp cận là để tạo ra một XmlWriter, và sử dụng khả năng của LINQ to XML để viết các phần tử tới một XmlWriter. Chủ đề này đã chứng minh cả hai phương pháp tiếp cận trên.

Ví dụ: Ví dụ này sử dụng làm chậm khả năng thực hiện của XStreamingElement cho các luồng đầu ra. Ví dụ này có thể chuyển đổi một tài liệu rất lớn trong khi vẫn duy trì một dấu vết bộ nhớ.

Lưu ý rằng tùy chỉnh axis (StreamCustomerItem) là một cách cụ thể bằng cách viết nó cho rằng tài liệu có Customer, Name và các phần tử, và rằng những phần tử sẽ được bố trí như sau Source.xml trong tài liệu. Một chi tiết mạnh mẽ việc triển khai thực hiện, tuy nhiên, bạn sẽ được chuẩn bị để phân tích cú pháp cho một tài liệu không hợp lệ.

The following is the source document, Source.xml:

```
static IEnumerable<XElement> StreamCustomerItem(string uri)
{
   using (XmlReader reader = XmlReader.Create(uri))
   {
      XElement name = null;
      XElement item = null;

      reader.MoveToContent();

      // Parse the file, save header information when encountered, and yield the
      // Item XElement objects as they are created.

      // loop through Customer elements
      while (reader.Read())
      {
         if (reader.NodeType == XmlNodeType.Element
            && reader.Name == "Customer")
         {
            // move to Name element
            while (reader.Read())
            {
               if (reader.NodeType == XmlNodeType.Element &&
                  reader.Name == "Name")
```

```
                {
                    name = XElement.ReadFrom(reader) as XElement;
                    break;
                }
            }

            // loop through Item elements
            while (reader.Read())
            {
                if (reader.NodeType == XmlNodeType.EndElement)
                    break;
                if (reader.NodeType == XmlNodeType.Element
                    && reader.Name == "Item")
                {
                    item = XElement.ReadFrom(reader) as XElement;
                    if (item != null)
                    {
                        XElement tempRoot = new XElement("Root",
                            new XElement(name)
                        );
                        tempRoot.Add(item);
                        yield return item;
                    }
                }
            }
        }
    }
}

static void Main(string[] args)
{
    XStreamingElement root = new XStreamingElement("Root",
        from el in StreamCustomerItem("Source.xml")
        select new XElement("Item",
            new XElement("Customer", (string)el.Parent.Element("Name")),
            new XElement(el.Element("Key"))
        )
    );
    root.Save("Test.xml");
    Console.WriteLine(File.ReadAllText("Test.xml"));
```

```
}
```

VII.11.7 Làm thế nào để:truy cập luồng XML phân mảnh với thông tin cho tiêu đề.

Đôi khi bạn có đọc các tập tin XML lớn bất kỳ, và viết ứng dụng của bạn để dấu vết bộ nhớ của ứng dụng là có thể dự đoán trước được. Nếu bạn cố gắng để lấy một cây XML lớn với một tệp tin XML, bộ nhớ của bạn sẽ được sử dụng để tỷ lệ kích cỡ của tập tin có nghĩa là, quá nhiều. Vì vậy, bạn nên sử dụng một kỹ thuật streaming, thay vì kĩ thuật theo chính sách.

Một tùy chọn khác là để viết ứng dụng của bạn bằng cách sử dụng XmlReader. Tuy nhiên, bạn có thể muốn sử dụng LINQ để truy vấn cây XML. Nếu nằm trong trường hợp này, bạn có thể viết riêng của bạn tùy chỉnh phương thức axis.

Để viết riêng phương thức axis của bạn, bạn viết một nhỏ sử dụng các phương pháp mà XmlReader để đọc nodes cho đến khi đạt đến một trong những nodes trong đó bạn quan tâm. Các phương pháp sau đó gọi ReadFrom, mà đọc từ XmlReader và bằng một XML phân mảnh. Nó sau đó trả về mỗi phân mảnh thông qua sản lượng trở về phương pháp đó là liệt kê tuỳ chỉnh phương thức axis của bạn. Sau đó bạn có thể viết truy vấn LINQ trên phương thức axis tuỳ chỉnh của bạn

Kỹ thuật Streaming được áp dụng tốt nhất trong tình huống mà bạn cần để xử lý các nguồn tài liệu chỉ một lần, và bạn có thể xử lý các yếu tố trong tài liệu có thứ tự. Truy vấn vận hành một tiêu chuẩn nhất định, chẳng hạn như OrderBy, lặp lại nguồn của mình, thu thập tất cả các dữ liệu, phân loại nó, và sau đó cuối cùng trả về mục đầu tiên trong dãy. Lưu ý rằng nếu bạn sử dụng một truy vấn mà nhà điều hành hiện ra nguồn trước khi trả về mục đầu tiên, bạn sẽ không giữ một dấu vết bộ nhớ nhỏ.

Các phương pháp tiếp cận ví dụ này cũng sẽ là để xem phần thông tin tiêu đề này, lưu các thông tin tiêu đề, và sau đó xây dựng một cây XML nhỏ có chứa cả các tiêu đề thông tin và các chi tiết mà bạn đang liệt kê. Các phương thức axis sau đó trả về mới này,

cây XML nhỏ. Các truy vấn sau đó có quyền truy cập vào các thông tin tiêu đề cũng như các thông tin chi tiết.

Phương pháp tiếp cận này có một dấu vết bộ nhớ nhỏ. Từng chi tiết XML phân mảnh như là sinh ra, không có tham chiếu được giữ lại, để trước phân mảnh, và nó có sẵn tập hợp rác. Lưu ý rằng kỹ thuật này tạo ra nhiều các đối tượng sống ngắn trên heap.

Ví dụ sau cho thấy làm thế nào để triển khai thực hiện và sử dụng một phương thức mà tùy chỉnh axis luồng XML phân mảnh từ các tập tin chỉ định bởi các URI. Đây là tùy chỉnh axis cụ thể bằng văn bản rằng nó như một tài liệu có Customer, Name, và các phần tử, và rằng những phần tử sẽ được bố trí như trong Source.xml tài liệu ở trên. Đó là một cách dễ dàng nhất để triển khai thực hiện. Một chi tiết mạnh mẽ việc triển khai thực hiện sẽ được chuẩn bị để phân tích cú pháp cho một tài liệu không hợp lệ.


```
static IEnumerable<XElement> StreamCustomerItem(string uri)
{
   using (XmlReader reader = XmlReader.Create(uri))
   {
      XElement name = null;
      XElement item = null;

      reader.MoveToContent();

      // Parse the file, save header information when encountered, and yield the
      // Item XElement objects as they are created.

      // loop through Customer elements
      while (reader.Read())
      {
         if (reader.NodeType == XmlNodeType.Element
            && reader.Name == "Customer")
         {
            // move to Name element
            while (reader.Read())
            {
               if (reader.NodeType == XmlNodeType.Element &&
                  reader.Name == "Name")
```

```
                {
                    name = XElement.ReadFrom(reader) as XElement;
                    break;
                }
            }

            // loop through Item elements
            while (reader.Read())
            {
                if (reader.NodeType == XmlNodeType.EndElement)
                    break;
                if (reader.NodeType == XmlNodeType.Element
                    && reader.Name == "Item")
                {
                    item = XElement.ReadFrom(reader) as XElement;
                    if (item != null) {
                        XElement tempRoot = new XElement("Root",
                            new XElement(name)
                        );
                        tempRoot.Add(item);
                        yield return item;
                    }
                }
            }
        }
    }
}

static void Main(string[] args)
{
    XElement xmlTree = new XElement("Root",
        from el in StreamCustomerItem("Source.xml")
        where (int)el.Element("Key") >= 3 && (int)el.Element("Key") <= 7
        select new XElement("Item",
            new XElement("Customer", (string)el.Parent.Element("Name")),
            new XElement(el.Element("Key"))
        )
    );
    Console.WriteLine(xmlTree);
}
```

VII.12 So sánh các Xpath và LINQ to XML.

Xpath 1,0 đưa ra trạng thái rằng một bộ sưu tập mà là kết quả của đánh giá là một biểu thức Xpath không được khai báo.

Tuy nhiên, khi lặp lại thông qua một bộ sưu tập trả về bởi một LINQ to XML Xpath phương thức axis, các nodes trong bộ sưu tập đang có trong tài liệu trở về trật tự. Đây là trường hợp ngay cả khi truy cập vào axis Xpath xác nhận, nơi xác nhận có trong ện trong điều khoản của tài liệu đảo ngược trật tự, chẳng hạn như preceding and preceding-sibling.

Ngược lại, hầu hết các LINQ to XML axis trả về trong bộ sưu tập tài liệu có thứ tự, nhưng hai trong số chúng, Ancestors và AncestorsAndSelf, trả về trong bộ sưu tập tài liệu thứ tự đảo ngược. Bảng sau liệt kê các axis, và cho biết bộ sưu tập lệnh cho mỗi thứ tự:

LINQ to XML axis	Ordering
XContainer.DescendantNodes	Document order
XContainer.Descendants	Document order
XContainer.Elements	Document order
XContainer.Nodes	Document order
XContainer.NodesAfterSelf	Document order
XContainer.NodesBeforeSelf	Document order
XElement.AncestorsAndSelf	Reverse document order
XElement.Attributes	Document order
XElement.DescendantNodesAndSelf	Document order
XElement.DescendantsAndSelf	Document order
XNode.Ancestors	Reverse document order
XNode.ElementsAfterSelf	Document order

XNode.ElementsBeforeSelf	Document order
XNode.NodesAfterSelf	Document order
XNode.NodesBeforeSelf	Document order

VIII. LINQ to Objects

Thuật ngữ "LINQ to Objects" đề cập đến việc sử dụng các truy vấn LINQ với bất kỳ tập hợp IEnumerable hay IEnumerable (T), mà không cần sử dụng một nhà cung cấp hay API như LINQ to SQL hay LINQ to XML. Bạn có thể sử dụng LINQ để truy vấn bất kỳ bộ sưu tập enumerable như: List (T), Array, hoặc Dictionary (TKey, TValue). Các tập hợp có thể được người dùng xác định hoặc có thể được trả lại bởi một. NET Framework API.

Trong một ý nghĩa cơ bản, LINQ to Objects đại diện cho một phương pháp tiếp cận mới tới tập hợp. Trong cách cũ, bạn phải viết vòng lặp foreach phức tạp theo lý thuyết để xác định rằng làm thế nào truy xuất dữ liệu từ một tập hợp. Trong LINQ đưa ra cách tiếp cận mới, bạn viết mã có tính mô tả những gì bạn muốn truy xuất. Ngoài ra, các truy vấn LINQ cung cấp ba sự tiện lợi hơn các vòng lặp foreach truyền thống:

1. Chúng ngắn gọn và dễ đọc, đặc biệt là khi có nhiều điều kiện lọc.
2. Chúng cung cấp bộ lọc mạnh mẽ, sắp xếp, và khả năng gom nhóm với đoạn mã ứng dụng nhỏ nhất.
3. Chúng có thể được chuyển đến các nguồn dữ liệu khác với một vài hoặc không có sửa đổi, bổ sung.

Nhìn chung, các hoạt động phức tạp hơn mà bạn muốn thực hiện trên cơ sở dữ liệu, các bạn sẽ thấy rõ hơn lợi ích bằng cách sử dụng LINQ thay vì kỹ thuật lặp truyền thống.

VIII.1 Làm thế nào để: Truy vấn với một ArrayList LINQ

Khi sử dụng LINQ để truy vấn các tập hợp không có đặc điểm chung IEnumerable như ArrayList, bạn phải khai báo rõ ràng kiểu phạm vi của các biến để phản ánh cụ thể của các loại đối tượng trong tập hợp. Ví dụ, nếu bạn có một ArrayList của các đối tượng ***Student*** , mệnh đề from của bạn nên trông như thế này:

```
// C#
var query = from Student s in arrList
```

Ví dụ sau cho thấy một truy vấn đơn giản trên một ArrayList. Lưu ý rằng ví dụ này khởi chạy khi đoạn code gọi phương thức Add, nhưng điều này không phải là một yêu cầu.

```
using System;
using System.Collections;
using System.Linq;

namespace NonGenericLINQ
{
    public class Student
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int[] Scores { get; set; }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ArrayList arrList = new ArrayList();
            arrList.Add(
                new Student
                    {
                        FirstName = "Svetlana", LastName = "Omelchenko",
Scores = new int[] { 98, 92, 81, 60 }
                    });
            arrList.Add(
                new Student
                    {
                        FirstName = "Claire", LastName = "O'Donnell", Scores
= new int[] { 75, 84, 91, 39 }
                    });
            arrList.Add(
                new Student
                    {
                        FirstName = "Sven", LastName = "Mortensen", Scores =
new int[] { 88, 94, 65, 91 }
```

```
                    });
            arrList.Add(
                new Student
                    {
                        FirstName = "Cesar", LastName = "Garcia", Scores =
new int[] { 97, 89, 85, 82 }
                    });

            var query = from Student student in arrList
                        where student.Scores[0] > 95
                        select student;

            foreach (Student s in query)
                Console.WriteLine(s.LastName + ": " + s.Scores[0]);

            // Keep the console window open in debug mode.
            Console.WriteLine("Press any key to exit.");
            Console.ReadKey();
        }
    }
}
```

VIII.2 LINQ and Strings

LINQ có thể được sử dụng để truy vấn và biến đổi những chỗi và tập tập của những chuỗi. Nó đặc biệt hữu ích với cấu trúc dữ liệu trong file văn bản. Các truy vấn LINQ có thể được kết hợp với các hàm và các biểu thức của chuỗi bình thường. Ví dụ, bạn có thể sử dụng các phương thức Split để tạo ra một mảng của những chuỗi mà bạn có thể truy vấn sau đó hoặc sửa đổi bằng cách sử dụng LINQ. Bạn có thể sử dụng các phương thức *IsMatch* trong mệnh đề ***where*** của một truy vấn LINQ. Và bạn có thể sử dụng LINQ để truy vấn hoặc sửa đổi *MatchCollection* các kết quả trả lại bởi một biểu thức chính quy.

VIII.3 Làm thế nào để: Đếm sự xuất hiện của một từ trong một chuỗi (LINQ)

Ví dụ này cho thấy cách sử dụng một truy vấn LINQ để đếm các xuất hiện của một từ trong một chuỗi. Lưu ý rằng để thực hiện việc đếm, trước tiên là gọi phương thức *Split* để tạo ra một mảng các từ. Ở đây là một chi phí cho sự thực thi phương thức *Split*. Nếu chỉ thao tác trên các chuỗi là để đếm các từ, bạn nên cân nhắc việc sử dụng các phương thức Matches hoặc IndexOf phù hợp để thay thế. Tuy nhiên, nếu chi phí không phải là một vấn đề nghiêm trọng, hoặc bạn đã phân chia các câu để thực hiện các loại truy

vấn trên nó, thì nó làm cho cảm giác sử dụng LINQ để truy cập các từ hoặc cụm từ cũng như.

```
class CountWords
{
    static void Main()
    {
        string text = @"Historically, the world of data and the world of
objects" +
          @" have not been well integrated. Programmers work in C# or Visual
Basic" +
          @" and also in SQL or XQuery. On the one side are concepts such as
classes," +
          @" objects, fields, inheritance, and .NET Framework APIs. On the
other side" +
          @" are tables, columns, rows, nodes, and separate languages for
dealing with" +
          @" them. Data types often require translation between the two
worlds; there are" +
          @" different standard functions. Because the object world has no
notion of query, a" +
          @" query can only be represented as a string without compile-time
type checking or" +
          @" IntelliSense support in the IDE. Transferring data from SQL
tables or XML trees to" +
          @" objects in memory is often tedious and error-prone.";

        string searchTerm = "data";

        //Convert the string into an array of words
        string[] source = text.Split(new char[] { '.', '?', '!', ' ', ';',
':', ',' }, StringSplitOptions.RemoveEmptyEntries);

        // Create and execute the query. It executes immediately
        // because a singleton value is produced.
        // Use ToLowerInvariant to match "data" and "Data"
        var matchQuery = from word in source
                         where           word.ToLowerInvariant()           ==
searchTerm.ToLowerInvariant()
                         select word;

        // Count the matches.
        int wordCount = matchQuery.Count();
        Console.WriteLine("{0} occurrences(s) of the search term \"{1}\" were
found.", wordCount, searchTerm);

        // Keep console window open in debug mode
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}
```

VIII.4 Làm thế nào để: Truy vấn cho câu đó chứa một bộ từ.

Ví dụ này cho thấy như thế nào để tìm câu trong một tập tin văn bản có chứa kết quả phù hợp cho mỗi một bộ từ. Mặc dù các điều kiện tìm kiếm là một đoạn code cứng trong ví dụ này, nó cũng có thể được lấy ra tại thời gian chạy. Trong ví dụ này, các truy vấn sẽ trả về các câu có chứa các cụm từ "Historically", "data," và "integrated"

```
class FindSentences
{
    static void Main()
    {
        string text = @"Historically, the world of data and the world of
objects " +
        @"have not been well integrated. Programmers work in C# or Visual
Basic " +
        @"and also in SQL or XQuery. On the one side are concepts such as
classes, " +
        @"objects, fields, inheritance, and .NET Framework APIs. On the other
side " +
        @"are tables, columns, rows, nodes, and separate languages for
dealing with " +
        @"them. Data types often require translation between the two worlds;
there are " +
        @"different standard functions. Because the object world has no
notion of query, a " +
        @"query can only be represented as a string without compile-time type
checking or " +
        @"IntelliSense support in the IDE. Transferring data from SQL tables
or XML trees to " +
        @"objects in memory is often tedious and error-prone.";

        // Split the text block into an array of sentences.
        string[] sentences = text.Split(new char[] { '.', '?', '!' });

        // Define the search terms. This list could also be dynamically
populated at runtime.
        string[] wordsToMatch = { "Historically", "data", "integrated" };

        // Find sentences that contain all the terms in the wordsToMatch
array.
        // Note that the number of terms to match is not specified at compile
time.
        var sentenceQuery = from sentence in sentences
                            let w = sentence.Split(new char[] { '.', '?',
'!', ' ', ';', ':', ',' },

StringSplitOptions.RemoveEmptyEntries)
                            where
w.Distinct().Intersect(wordsToMatch).Count() == wordsToMatch.Count()
                            select sentence;
```

```
        // Execute the query. Note that you can explicitly type
        // the iteration variable here even though sentenceQuery
        // was implicitly typed.
        foreach (string str in sentenceQuery)
        {
            Console.WriteLine(str);
        }

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}
```

Các truy vấn làm việc bằng cách trước tiên phân đôi văn bản vào câu, và sau đó tách các câu vào một mảng có chứa những chuỗi của mỗi từ. Đối với mỗi arrays này, phương thức *Distinct* loại bỏ tất cả các kí tự nào bị trùng lặp, và sau đó truy vấn thực hiện các hoạt động trên một phân cắt từ mảng và các mảng ***wordsToMatch***. Nếu việc đếm các điểm giao là giống như đếm của mảng *wordsToMatch*, tất cả các từ đã được tìm thấy trong từ và ban đầu là câu trả về.

Trong lúc gọi đến Split, các dấu chấm câu như dấu tách được sử dụng để loại bỏ chúng khỏi chuỗi. Nếu bạn đã không làm được điều này, ví dụ như bạn có thể có một chuỗi "Historically," rằng sẽ không phù hợp với "Historically" trong mảng *wordsToMatch*. Bạn có thể sử dụng để có thêm các dấu tách, tùy thuộc vào loại dấu chấm câu được tìm thấy trong các nguồn văn bản.

VIII.5 Làm thế nào để: Truy vấn cho các ký tự trong một String (LINQ)

Bởi vì trong thực hiện các lớp String có chung interface IEnumerable (T), bất kỳ chuỗi có thể được truy vấn như là một chuỗi các ký tự. Tuy nhiên, điều này không phải là một cách sử dụng chung của LINQ.

Ví dụ sau truy vấn một chuỗi để xác định số lượng số chữ số chứa trong nó. Lưu ý rằng các truy vấn là "reused" sau khi được thực hiện lần đầu tiên. Điều này có thể vì các truy vấn tự nó không lưu trữ bất kỳ kết quả thực sự.

```
class QueryAString
{
    static void Main()
    {
```

```
        string aString = "ABCDE99F-J74-12-89A";

        // Select only those characters that are numbers
        IEnumerable<char> stringQuery =
          from ch in aString
          where Char.IsDigit(ch)
          select ch;

        // Execute the query
        foreach (char c in stringQuery)
            Console.Write(c + " ");

        // Call the Count method on the existing query.
        int count = stringQuery.Count();
        Console.WriteLine("Count = {0}", count);

        // Select all characters before the first '-'
        IEnumerable<char> stringQuery2 = aString.TakeWhile(c => c != '-');

        // Execute the second query
        foreach (char c in stringQuery2)
            Console.Write(c);

        Console.WriteLine(System.Environment.NewLine + "Press any key to exit");
        Console.ReadKey();

    }
}
```

VIII.6 Làm thế nào để: Kết hợp LINQ truy vấn với các biểu thức chính quy.

Ví dụ này cho thấy cách sử dụng lớp Regex để tạo ra một biểu thức chính quy cho phù hợp hơn trong chuỗi văn bản. Các truy vấn LINQ là cách dễ dàng để lọc chính xác về các tập tin mà bạn muốn tìm kiếm với các biểu thức chính quy, và để hình thành các kết quả.

```
class QueryWithRegEx
{
    public static void Main()
    {
        string startFolder = @"c:\program files\Microsoft Visual Studio 9.0\";

        IEnumerable<System.IO.FileInfo> fileList = GetFiles(startFolder);
        System.Text.RegularExpressions.Regex searchTerm =
            new                              System.Text.RegularExpressions.Regex(@"Visual
(Basic|C#|C\+\+|J#|SourceSafe|Studio)");
        var queryMatchingFiles =
            from file in fileList
            where file.Extension == ".htm"
            let fileText = System.IO.File.ReadAllText(file.FullName)
            let matches = searchTerm.Matches(fileText)
            where searchTerm.Matches(fileText).Count > 0
            select new
            {
                name = file.FullName,
```

```
                matches = from System.Text.RegularExpressions.Match match in matches
                          select match.Value
            };

        // Execute the query.
        Console.WriteLine("The term \"{0}\" was found in:", searchTerm.ToString());


        foreach (var v in queryMatchingFiles)
        {            string s = v.name.Substring(startFolder.Length - 1);
            Console.WriteLine(s);

            // For this file, write out all the matching strings
            foreach (var v2 in v.matches)
            {
                Console.WriteLine("  " + v2);
            }
        }
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }     static IEnumerable<System.IO.FileInfo> GetFiles(string path)
    {
        if (!System.IO.Directory.Exists(path))
            throw new System.IO.DirectoryNotFoundException();

        string[] fileNames = null;
        List<System.IO.FileInfo> files = new List<System.IO.FileInfo>();

        fileNames          =          System.IO.Directory.GetFiles(path,          "*.*",
System.IO.SearchOption.AllDirectories);
        foreach (string name in fileNames)
        {
            files.Add(new System.IO.FileInfo(name));
        }
        return files;
    }
}
```

Lưu ý rằng bạn cũng có thể truy vấn đối tượng MatchCollection được trả lại bởi một RegEx tìm kiếm. Trong ví dụ này chỉ có giá trị của mỗi tương xứng là được trả về trong các kết quả. Tuy nhiên, đây cũng là để có thể sử dụng LINQ để thực hiện tất cả các loại lọc, phân loại, và các nhóm trên tập hợp. Bởi vì MatchCollection là một tập hợp không có kiểu chung IEnumerable, bạn cần phải rõ ràng trạng tái trong phạm vi của các loại biến trong truy vấn.

VIII.7 Câu hỏi bán cấu trúc dữ liệu ở định dạng văn bản

Nhiều loại khác nhau của các file văn bản bao gồm một loạt các dòng, thường xuyên với các định dạng tương tự, chẳng hạn như tab hay dấu phẩy phân các tập tin hoặc cố định-chiều dài dòng. Sau khi bạn đọc như một tập tin văn bản vào bộ nhớ, bạn có thể

sử dụng LINQ để truy vấn và / hoặc sửa đổi dòng. Các truy vấn LINQ cũng đơn giản hóa các nhiệm vụ, kết hợp dữ liệu từ nhiều nguồn

VIII.7.1 Làm thế nào để: Tìm các tập khác biệt giữa hai danh sách (LINQ)

Ví dụ này cho thấy như thế nào để sử dụng LINQ để so sánh hai danh sách của những chuỗi và những dòng có trong names1.txt nhưng không có trong names2.txt.

```
class CompareLists
{
    static void Main()
    {
        // Create the IEnumerable data sources.
        string[] names1 = System.IO.File.ReadAllLines(@"../../../names1.txt");
        string[] names2 = System.IO.File.ReadAllLines(@"../../../names2.txt");

        // Create the query. Note that method syntax must be used here.
        IEnumerable<string> differenceQuery =
          names1.Except(names2);

        // Execute the query.
        Console.WriteLine("The following lines are in names1.txt but not names2.txt");
        foreach (string s in differenceQuery)
            Console.WriteLine(s);

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }
}
```

VIII.7.2 Làm thế nào để: Sắp xếp hay Lọc dữ liệu Văn bản bởi bất kì một từ hoặc một trường (LINQ)

Ví dụ sau cho thấy làm thế nào để phân loại cấu trúc của dòng văn bản, chẳng hạn như giá trị phân cách bằng dấu phẩy, bởi bất kỳ trường nào trong dòng. Các trường có thể được xác định theo kiểu động tại thời gian chạy. Giả định rằng các trường trong scores.csv của sinh viên đại diện cho một số ID, tiếp theo là một loạt các bài kiểm tra bốn điểm số.

```
public class SortLines
{
    static void Main()
    {
        // Create an IEnumerable data source
        string[] scores = System.IO.File.ReadAllLines(@"../../../scores.csv");

        // Change this to any value from 0 to 4.
        int sortField = 1;

        Console.WriteLine("Sorted highest to lowest by field [{0}]:", sortField);
```

```
        // Demonstrates how to return query from a method.
        // The query is executed here.
        foreach (string str in RunQuery(scores, sortField))
        {
            Console.WriteLine(str);
        }

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }

    // Returns the query variable, not query results!
    static IEnumerable<string> RunQuery(IEnumerable<string> source, int num)
    {
        // Split the string and sort on field[num]
        var scoreQuery = from line in source
                         let fields = line.Split(',')
                         orderby fields[num] descending
                         select line;

        return scoreQuery;
    }
}
```

VIII.7.3 Làm thế nào để: Sắp xếp lại các trường được định giới trong file.

Một giá trị phân cách bằng dấu phẩy (CSV) file là một tập tin văn bản nó thường được sử dụng để lưu trữ dữ liệu bảng tính hay xếp như bảng các dữ liệu được đại diện bởi các hàng và cột. Bằng việc sử dụng phương thức Split để phân cách các lĩnh vực, nó là rất dễ dàng để truy vấn và thao tác các file bằng cách sử dụng LINQ. Trong thực tế, cùng một kỹ thuật có thể được sử dụng để sắp xếp lại phần nào cấu trúc của dòng văn bản; nó không phải là giới hạn đối với các tập tin CSV.

Trong ví dụ sau, giả định rằng ba cột đại diện của học sinh "last name", "first name", và "ID". Các trường đang có theo thứ tự chữ cái dựa trên tên của mỗi học sinh. Các truy vấn tạo ra một dãy mới, trong đó các cột ID xuất hiện trước, theo sau là một cột thứ hai là kết hợp hai phần last name và first name của học sinh. Những dòng đã được sắp xếp theo ID. Các kết quả được lưu vào một tập tin mới và các dữ liệu ban đầu là không sửa đổi.

VIII.8 Để tạo các tệp dữ liệu

Tạo mới một dự ánVisual C# và sao chép những dòng này vào một tập tin văn bản gốc mà có tên spreadsheet1.csv. Lưu tập tin trong thư mục giải pháp của bạn.

```
class CSVFiles
{
    static void Main(string[] args)
    {
        // Create the IEnumerable data source
        string[] lines = System.IO.File.ReadAllLines(@"../../../spreadsheet1.csv");

        // Create the query. Put field 2 first, then
        // reverse and combine fields 0 and 1 from the old field
        IEnumerable<string> query =
            from line in lines
            let x = line.Split(',')
            orderby x[2]
            select x[2] + ", " + (x[1] + " " + x[0]);

        // Execute the query and write out the new file. Note that WriteAllLines
        // takes a string[], so ToArray is called on the query.
        System.IO.File.WriteAllLines(@"../../../spreadsheet2.csv", query.ToArray());

        Console.WriteLine("Spreadsheet2.csv written to disk. Press any key to exit");
        Console.ReadKey();
    }
}
```

VIII.8.1 Làm thế nào để: Kết hợp và so sánh các tập hợp chuỗi (LINQ)

Ví dụ này cho thấy như thế nào để trộn file có chứa dòng văn bản và sau đó phân loại các kết quả. Cụ thể, nó cho thấy như thế nào để thực hiện một cách dễ dàng để ghép, hợp trên hai bộ dòng văn bản.

```
class MergeStrings
    {
        static void Main(string[] args)
        {
            //Put text files in your solution folder
            string[]                                fileA                                =
System.IO.File.ReadAllLines(@"../../../names1.txt");
            string[]                                fileB                                =
System.IO.File.ReadAllLines(@"../../../names2.txt");

            //Simple concatenation and sort. Duplicates are preserved.
            IEnumerable<string> concatQuery =
                fileA.Concat(fileB).OrderBy(s => s);

            // Pass the query variable to another function for execution.
            OutputQueryResults(concatQuery,  "Simple  concatenate  and  sort.
Duplicates are preserved:");

            // Concatenate and remove duplicate names based on
            // default string comparer.
            IEnumerable<string> uniqueNamesQuery =
                fileA.Union(fileB).OrderBy(s => s);
            OutputQueryResults(uniqueNamesQuery,  "Union  removes  duplicate
names:");
```

```
            // Find the names that occur in both files (based on
            // default string comparer).
            IEnumerable<string> commonNamesQuery =
                fileA.Intersect(fileB);
            OutputQueryResults(commonNamesQuery,      "Merge      based      on
intersect:");

            // Find the matching fields in each list. Merge the two
            // results by using Concat, and then
            // sort using the default string comparer.
            string nameMatch = "Garcia";

            IEnumerable<String> tempQuery1 =
                from name in fileA
                let n = name.Split(',')
                where n[0] == nameMatch
                select name;

            IEnumerable<string> tempQuery2 =
                from name2 in fileB
                let n2 = name2.Split(',')
                where n2[0] == nameMatch
                select name2;

            IEnumerable<string> nameMatchQuery =
                tempQuery1.Concat(tempQuery2).OrderBy(s => s);
            OutputQueryResults(nameMatchQuery, String.Format("Concat based on
partial name match \"{0}\":", nameMatch));

            // Keep the console window open in debug mode.
            Console.WriteLine("Press any key to exit");
            Console.ReadKey();
        }

        static  void  OutputQueryResults(IEnumerable<string>  query,  string
message)
        {
            Console.WriteLine(System.Environment.NewLine + message);
            foreach (string item in query)
            {
                Console.WriteLine(item);
            }
            Console.WriteLine("{0} total names in list", query.Count());
        }
    }
```

VIII.8.2 Làm thế nào để: Lấy ra tập hợp đối tượng từ nhiều nguồn (LINQ)

Ví dụ này cho thấy như thế nào để trộn dữ liệu từ các loại nguồn khác nhau vào một chuỗi các loại mới. Các ví dụ trong các mã sau đây sẽ trộn các chuỗi với những

mảng số nguyên. Tuy nhiên, trong cùng một nguyên tắc áp dụng cho bất kỳ hai nguồn dữ liệu, bao gồm cả bất kỳ sự kết hợp của các đối tượng trong bộ nhớ. **Ví dụ:** Ví dụ sau cho thấy cách sử dụng một tên kiểu Student để lưu trữ dữ liệu từ hai bộ nhớ trong tập hợp của những chuỗi mà mô phỏng dữ liệu trong bảng tính. Trước tiên tạp hợp của các chuỗi miêu tả các tên và ID, và tiếp theo tập hợp miêu tả ID của học sinh(trong cột đầu tiên).

```
class Student
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int ID { get; set; }
    public List<int> ExamScores { get; set; }
}

class PopulateCollections
{
    static void Main()
    {
       Dissimilar Files (LINQ)
        string[] names = System.IO.File.ReadAllLines(@"../../../names.csv");
        string[]                                scores                                =
System.IO.File.ReadAllLines(@"../../../scores.csv");
        IEnumerable<Student> queryNamesScores =
            from name in names
            let x = name.Split(',')
            from score in scores
            let s = score.Split(',')
            where x[2] == s[0]
            select new Student()
            {
                FirstName = x[0],
                LastName = x[1],
                ID = Convert.ToInt32(x[2]),
                ExamScores = (from scoreAsText in s.Skip(1)
                              select Convert.ToInt32(scoreAsText)).
                              ToList()
            };

        List<Student> students = queryNamesScores.ToList();
        foreach (var student in students)
        {
            Console.WriteLine("The average score of {0} {1} is {2}.",
                student.FirstName,                              student.LastName,
student.ExamScores.Average());
        }

        //Keep console window open in debug mode
        Console.WriteLine("Press any key to exit.");
```

```
            Console.ReadKey();
        }
}
```

VIII.8.3 Làm thế nào để: Gia nhập nội dung từ các file không cùng dạng.

Ví dụ này cho thấy như thế nào để tham gia dữ liệu từ hai dấu phẩy phân chia tập tin rằng một trong các giá trị được sử dụng như một chìa khóa phù hợp. Kỹ thuật này có thể là hữu ích nếu bạn có kết hợp dữ liệu từ hai bảng tính, hay dữ liệu từ một bảng tính và một tập tin có định dạng khác, vào một tập tin mới. Bạn cũng có thể sửa đổi ví dụ này để làm việc với bất kỳ hình thức nào về cấu trúc văn bản.

```
class JoinStrings
{
    static void Main()
    {

        string[] names = System.IO.File.ReadAllLines(@"../../../names.csv");
        string[] scores = System.IO.File.ReadAllLines(@"../../../scores.csv");
        IEnumerable<string> scoreQuery1 =
            from name in names
            let nameFields = name.Split(',')
            from id in scores
            let scoreFields = id.Split(',')
            where nameFields[2] == scoreFields[0]
            select nameFields[0] + "," + scoreFields[1] + "," + scoreFields[2]
                   + "," + scoreFields[3] + "," + scoreFields[4];
        OutputQueryResults(scoreQuery1, "Merge two spreadsheets:");
        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }

    static void OutputQueryResults(IEnumerable<string> query, string message)
    {
        Console.WriteLine(System.Environment.NewLine + message);
        foreach (string item in query)
        {
            Console.WriteLine(item);
        }
        Console.WriteLine("{0} total names in list", query.Count());
    }
}
```

VIII.8.4 Làm thế nào để: Tách một file vào các file bằng cách sử dụng các nhóm (LINQ)

Ví dụ sau cho thấy công việc đó.

```
class SplitWithGroups
{
    static void Main()
    {
        string[] fileA = System.IO.File.ReadAllLines(@"../../../names1.txt");
```

```
        string[] fileB = System.IO.File.ReadAllLines(@"../../../names2.txt");
        var mergeQuery = fileA.Union(fileB);.
        var groupQuery = from name in mergeQuery
                         let n = name.Split(',')
                         group name by n[0][0] into g
                         orderby g.Key
                         select g;
        foreach (var g in groupQuery)
        {
            // Create the new file name.
            string fileName = @"../../../testFile_" + g.Key + ".txt";

            // Output to display.
            Console.WriteLine(g.Key);

            // Write file.
            using (System.IO.StreamWriter sw = new System.IO.StreamWriter(fileName))
            {
                foreach (var item in g)
                {
                    sw.WriteLine(item);
                    // Output to console for example purposes.
                    Console.WriteLine("   {0}", item);
                }
            }
        }
        // Keep console window open in debug mode.
        Console.WriteLine("Files have been written. Press any key to exit");
        Console.ReadKey();
    }
}
```

VIII.8.5 Làm thế nào để: Tính toán giá trị của cột trong một văn bản của tệp CSV (LINQ)

Ví dụ này cho thấy như thế nào để thực hiện tổng hợp thao tác tính toán như Sum, Average, Min, Max và trên các cột của một file. Csv. Ví dụ nguyên tắc được hiển thị ở đây có thể được áp dụng cho các loại cấu trúc văn bản.

```
Class SumColumns
{
    static void Main(string[] args)
    {
        string[] lines = System.IO.File.ReadAllLines(@"../../../scores.csv");
        int exam = 3;
        SingleColumn(lines, exam + 1);
        Console.WriteLine();
        MultiColumns(lines);

        Console.WriteLine("Press any key to exit");
        Console.ReadKey();
    }

    static void SingleColumn(IEnumerable<string> strs, int examNum)
    {
```

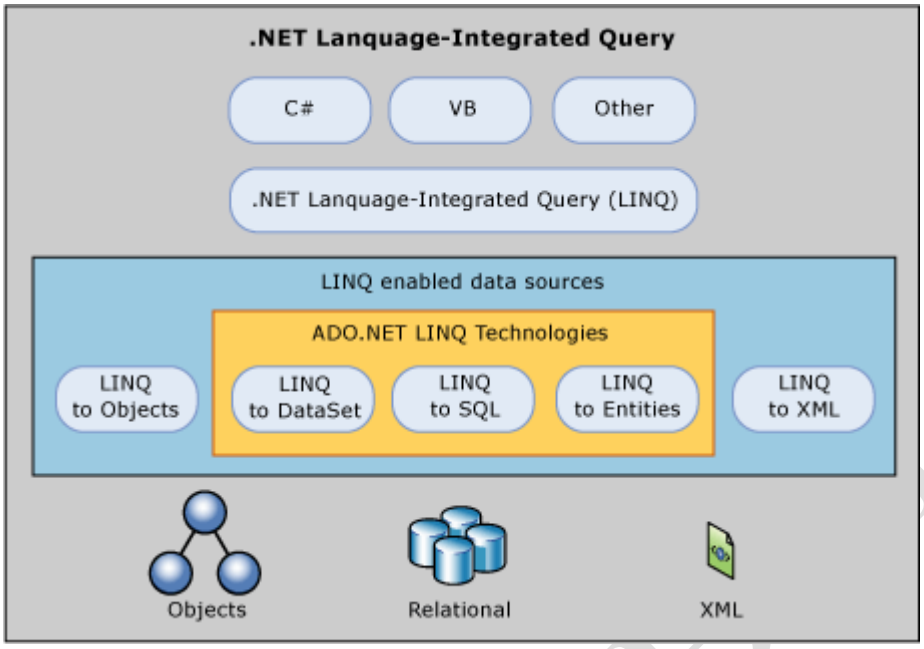
```
        Console.WriteLine("Single Column Query:");
         var columnQuery =
            from line in strs
            let x = line.Split(',')
            select Convert.ToInt32(x[examNum]);
                var results = columnQuery.ToList();

        // Perform aggregate calculations
        // on the column specified by examNum.
        double average = results.Average();
        int max = results.Max();
        int min = results.Min();

        Console.WriteLine("Exam  #{0}:  Average:{1:##.##}  High  Score:{2}  Low
Score:{3}",
                 examNum, average, max, min);
    }
    static void MultiColumns(IEnumerable<string> strs)
    {
        Console.WriteLine("Multi Column Query:");

        IEnumerable<IEnumerable<int>> query =
            from line in strs
            let x = line.Split(',')
            let y = x.Skip(1)
            select (from str in y
                    select Convert.ToInt32(str));

        // Execute and cache the results for performance.
        // ToArray could also be used here.
        var results = query.ToList();

        // Find out how many columns we have.
        int columnCount = results[0].Count();
        for (int column = 0; column < columnCount; column++)
        {
            var res2 = from row in results
                       select row.ElementAt(column);
            double average = res2.Average();
            int max = res2.Max();
            int min = res2.Min();

            // 1 is added to column because Exam numbers
            // begin with 1
            Console.WriteLine("Exam  #{0}  Average:  {1:##.##}  High  Score:  {2}
Low Score: {3}",
                              column + 1, average, max, min);
        }
    }
}
```

IX. LINQ to ADO.NET

Ngôn ngữ tích hợp truy vấn (LINQ) định nghĩa một tập các toán tử truy vấn chuẩn bạn có thể sử dụng trong ngôn ngữ lập trình trên nền .NET Framework 3.0. Các toán tử truy vấn chuẩn cho phép bạn vận hành dự án, lọc, và đi ngang bộ nhớ trong tập hợp hoặc một cơ sở dữ liệu trong bảng. Lưu ý rằng các truy vấn LINQ được thể hiện trong ngôn ngữ lập trình riêng của mình, và không như chuỗi kí tự nhúng vào đoạn mã ứng dụng. Đây là một thay đổi đáng kể từ các ứng dụng đã được viết trên các phiên bản cũ của .NET Framework. Viết truy vấn từ bên trong ngôn ngữ lập trình của bạn cung cấp một vài lợi thế chủ chốt. Nó đơn giản bằng cách loại bỏ các truy vấn cần phải sử dụng một ngôn ngữ truy vấn riêng biệt. Và nếu bạn sử dụng IDE Visual Studio 2008 , LINQ cũng cho phép bạn tận dụng lợi thế của quá trình kiểm tra tại thời gian biên dịch, loại tĩnh, và trình hỡ trợ thông minh.

LINQ được tích hợp vào nhiều khía cạnh khác nhau của việc truy cập dữ liệu trong .NET Framework, bao gồm cả việc ngắt kết nối DataSet với mô hình lập trình và hiện tại giản đồ cơ sở dữ liệu SQL Server. Phần này để mô tả trong LINQ to ADO.NET.

Dưới đây là mô hình cung cấp tổng quan của LINQ to ADO.NET làm việc như thế nào liên quan đến ngôn ngữ lập trình cao cấp , các công nghệ LINQ, và các dữ liệu ngồn mà LINQ làm việc.



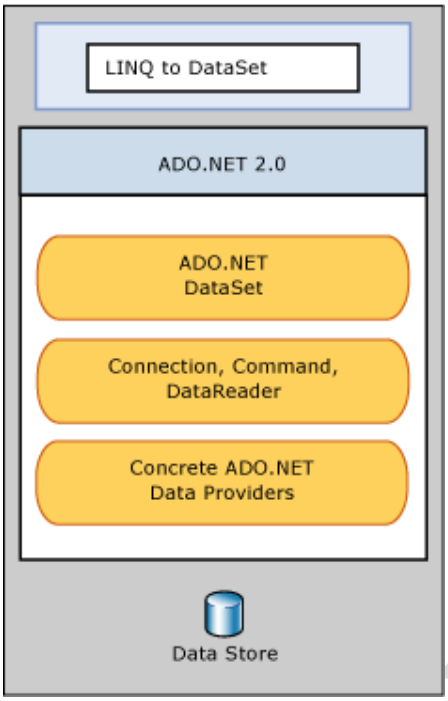
X. LINQ to DataSet

LINQ to DataSet làm cho nó dễ dàng hơn và nhanh hơn để truy vấn trên dữ liệu lưu trữ trong một đối tượng DataSet. Cụ thể, LINQ to DataSet làm đơn giản hóa câu truy vấn bằng cách cho phép các người phát triển viết các truy vấn từ các ngôn ngữ lập trình, thay vì sử dụng một ngôn ngữ truy vấn riêng biệt. Điều này đặc biệt hữu ích cho những người phát triển sử dụng Visual Studio, những người bây giờ có thể tận dụng lợi thế của thời gian biên dịch cú pháp kiểm tra, các kiểu tĩnh, và trình hỗ trợ cung cấp bởi Visual Studio trong truy vấn của họ.

LINQ to DataSet cũng có thể được sử dụng để truy vấn trên dữ liệu đã được hợp nhất từ một hoặc nhiều nguồn dữ liệu. Điều này cho phép một vài kịch bản có yêu cầu tính linh hoạt trong cách miêu tả cho dữ liệu và xử lý, như là câu truy vấn cục bộ tập hợp lại dữ liệu và giữa tầng bộ nhớ đệm trong ứng dụng web. Đặc biệt, báo cáo chung chung, phân tích, các giao dịch thông minh các ứng dụng này yêu cầu phương thức này được thao tác bằng tay.

Các hàm chức năng LINQ to DataSet là thông qua các phương phương thức mở rộng trong các lớp DataRowExtensions và DataTableExtensions. LINQ to DataSet xây

dựng và sử dụng sẵn có trên kiến trúc ADO.NET 2.0, và không nó có nghĩa là để thay thế ADO.NET 2.0 trong mã ứng dụng. Hiện nay ADO.NET 2.0 mã này sẽ tiếp tục chức năng trong một ứng dụng LINQ to DataSet. Mối quan hệ của LINQ to DataSet tới ADO.NET 2.0 và các dữ liệu lưu trữ được minh họa trong sơ đồ sau đây.



X.1 Tổng quan về LINQ to DataSet.

The DataSet là một trong những chi tiết được sử dụng rộng rãi các thành phần của ADO.NET. Đó là một yếu tố chủ chốt của các chương trình tách rời dựa trên ADO.NET, và nó cho phép bạn cache dữ liệu từ các nguồn dữ liệu khác nhau. Để tầng trình diễn, các DataSet được tích hợp chặt chẽ với GUI kiểm soát cho các liên kết dữ liệu. Đối với các tầng trung gian, nó cung cấp một bộ nhớ cache duy trì các quan hệ hình dạng của dữ liệu, bao gồm các truy vấn nhanh chóng đơn giản và hệ đẳng cấp hướng dịch vụ. Một kỹ thuật chung được sử dụng để giảm số lượng các yêu cầu trên một cơ sở dữ liệu là sử dụng DataSet cho bộ nhớ đệm ở tầng giữa. Ví dụ, lưu ý đến một điều khiển dữ liệu ứng dụng web ASP.NET . Thông thường, một phần quan trọng của các ứng dụng dữ liệu, không có thay đổi thường xuyên và là xuyên suốt một phiên làm việc hoặc người sử dụng. Dữ liệu này có thể được giữ trong bộ nhớ trên Web Server, mà làm giảm số lượng các yêu cầu

đối với cơ sở dữ liệu và đẩy mạnh tương tác của người dùng. Một khía cạnh hữu ích của DataSet là nó cho phép một ứng dụng làm cho tập con của dữ liệu từ một hoặc nhiều nguồn dữ liệu vào không gian ứng dụng . Các ứng dụng sau đó có thể thao tác các dữ liệu trong bộ nhớ, trong khi duy trì các mối quan hệ của nó.

Mặc dù một loạt các sự nhô lên, DataSet đã hạn chế khả năng truy vấn. Chọn các phương thức có thể được sử dụng để lọc và phân loại, và các phương thức GetChildRows và GetParentRow có thể được sử dụng cho sự điều hướng hệ đẳng cấp. Đối với bất cứ điều gì phức tạp hơn, tuy nhiên, những người phát triển phải viết một truy vấn tuỳ chỉnh. Điều này có thể kết quả trong các ứng dụng có hiệu suất kém và rất khó để duy trì.

LINQ to DataSet làm cho nó dễ dàng hơn và nhanh hơn để truy vấn dữ liệu lưu trữ trong một đối tượng DataSet. Các truy vấn này được thể hiện trong chính ngôn ngữ lập trình, chứ không phải là như chuỗi chữ nhúng vào mã ứng dụng. Điều này có nghĩa là các người phát triển không tìm hiểu một ngôn ngữ truy vấn riêng biệt. Ngoài ra, LINQ to DataSet cho phép nhà phát triển Visual Studio để làm việc có hiệu quả hơn, bởi vì Visual Studio IDE cung cấp cú pháp kiểm tra thời gian biên dịch, các kiểu tĩnh, và trình hỗ trợ thông minh hỗ trợ cho LINQ. LINQ to DataSet cũng có thể được sử dụng để truy vấn trên dữ liệu đã được hợp nhất từ một hoặc nhiều nguồn dữ liệu. Điều này cho phép một vài kịch bản có yêu cầu tính linh hoạt trong cách miêu tả cho dữ liệu và xử lý, như là câu truy vấn cục bộ tập hợp lại dữ liệu và giữa tầng bộ nhớ đệm trong ứng dụng web. Đặc biệt, báo cáo chung chung, phân tích, các giao dịch thông minh các ứng dụng này yêu cầu phương thức này được thao tác bằng tay.

X.2 Truy vấn các DataSet sử dụng LINQ để DataSet

Trước khi bạn có thể bắt đầu truy vấn một đối tượng DataSet bằng cách sử dụng LINQ to DataSet, bạn cần phải tải dữ liệu lên DataSet. Có một số cách để tải dữ liệu vào một DataSet, chẳng hạn như bằng cách sử dụng lớp DataAdapter hay LINQ to SQL. Sau khi dữ liệu đã được tải vào một đối tượng DataSet, bạn có thể bắt đầu truy vấn nó. Hình thành các truy vấn bằng cách sử dụng LINQ to DataSet là tương tự bằng cách sử dụng

ngôn ngữ truy vấn tích hợp (LINQ) dựa trên các dữ liệu nguồn mà LINQ cho phép. Các truy vấn LINQ có thể được thực hiện đối với một bảng trong một DataSet hay dựa trên nhiều hơn một bảng bằng cách sử dụng Join và GroupJoin vận hành theo hoạt động truy vấn chuẩn.

Truy vấn LINQ được hỗ trợ đối với cả hai typed and các đối tượng untyped DataSet. Nếu giản đồ của DataSet được biết đến tại thời gian thiết kế ứng dụng, một kiểu DataSet là một phó thác. Trong một kiểu DataSet, các bảng và hàng kiểu thành viên cho mỗi cột, nó làm cho các truy vấn đơn giản hơn và dễ đọc hơn.

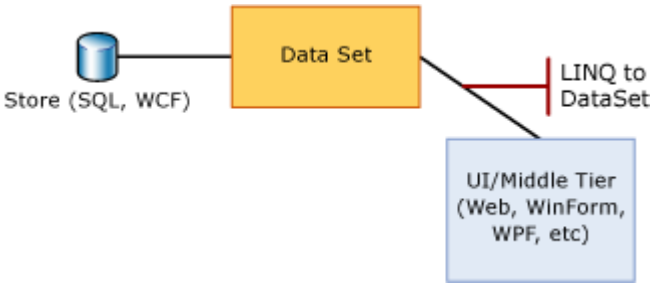
Ngoài ra các toán tử truy vấn chuẩn được triển khai thực hiện trong thư viện System.Core.dll, LINQ to DataSet cho biết thêm một vài DataSet-cụ thể mở rộng đã làm cho nó dễ dàng hơn để truy vấn trên một tập các đối tượng DataRow. Những DataSet-cụ thể mở rộng bao gồm các toán tử cho so sánh trình tự của các hàng, cũng như các phương thức là cung cấp truy cập đến các cột giá trị của một DataRow.

X.3 Ứng dụng N-tier và LINQ to DataSet

Các ứng dụng dữ liệu N-tier có trung tâm dữ liệu của ứng dụng được tách ra thành nhiều lớp logic (hoặc cp). Một điển hình của ứng dụng N-tier bao gồm một tầng trình diễn, một tầng trung chuyển dữ liệu, và một tầng dữ liệu(Mô hình tam tầng). Tách ứng dụng thành các cấu kiện vào các tầng riêng biệt làm tăng lên có thể duy trì được và quy mô của ứng dụng.

Trong các ứng dụng N-tier , các DataSet thường được sử dụng trong tầng trung gian để cache các thông tin cho một ứng dụng web.Truy vấn LINQ to DataSet thực hiện chức năng thông qua các phương thức mở rộng và mở rộng hiện có của ADO.NET 2.0 DataSet.

Dưới đây là những sơ đồ hiển thị như thế nào LINQ to DataSet liên quan đến các DataSet và fits vào một ứng dụng N-tier:



X.4 Đang tải dữ liệu vào một DataSet

Một đối tượng DataSet trước tiên phải được đổ dữ liệu lên trước khi bạn có thể truy vấn trên nó với LINQ to DataSet. Có nhiều cách khác nhau để đổ dữ liệu lên DataSet. Ví dụ, bạn có thể sử dụng LINQ to SQL để truy vấn cơ sở dữ liệu và tải các kết quả vào DataSet.

Thêm một cách nào để nạp dữ liệu vào một DataSet là sử dụng những lớp DataAdapter, để lấy các dữ liệu từ cơ sở dữ liệu. Đây là minh họa trong ví dụ sau.

```
try
{
    // Create a new adapter and give it a query to fetch sales order, contact,
    // address, and product information for sales in the year 2002. Point connection
    // information to the configuration setting "AdventureWorks".
    string connectionString = "Data Source=localhost;Initial Catalog=AdventureWorks;"
        + "Integrated Security=true;";
    SqlDataAdapter da = new SqlDataAdapter(
        "SELECT SalesOrderID, ContactID, OrderDate, OnlineOrderFlag, " +
        "TotalDue, SalesOrderNumber, Status, ShipToAddressID, BillToAddressID " +
        "FROM Sales.SalesOrderHeader " +
        "WHERE DATEPART(YEAR, OrderDate) = @year; " +

        "SELECT d.SalesOrderID, d.SalesOrderDetailID, d.OrderQty, " +
        "d.ProductID, d.UnitPrice " +
        "FROM Sales.SalesOrderDetail d " +
        "INNER JOIN Sales.SalesOrderHeader h " +
        "ON d.SalesOrderID = h.SalesOrderID  " +
        "WHERE DATEPART(YEAR, OrderDate) = @year; " +

        "SELECT p.ProductID, p.Name, p.ProductNumber, p.MakeFlag, " +
        "p.Color, p.ListPrice, p.Size, p.Class, p.Style, p.Weight  " +
        "FROM Production.Product p; " +

        "SELECT DISTINCT a.AddressID, a.AddressLine1, a.AddressLine2, " +
        "a.City, a.StateProvinceID, a.PostalCode " +
        "FROM Person.Address a " +
        "INNER JOIN Sales.SalesOrderHeader h " +
        "ON  a.AddressID = h.ShipToAddressID OR a.AddressID = h.BillToAddressID " +
        "WHERE DATEPART(YEAR, OrderDate) = @year; " +

        "SELECT DISTINCT c.ContactID, c.Title, c.FirstName, " +
        "c.LastName, c.EmailAddress, c.Phone " +
        "FROM Person.Contact c " +
        "INNER JOIN Sales.SalesOrderHeader h " +
        "ON c.ContactID = h.ContactID " +
        "WHERE DATEPART(YEAR, OrderDate) = @year;",
    connectionString);
```

```
    // Add table mappings.
    da.SelectCommand.Parameters.AddWithValue("@year", 2002);
    da.TableMappings.Add("Table", "SalesOrderHeader");
    da.TableMappings.Add("Table1", "SalesOrderDetail");
    da.TableMappings.Add("Table2", "Product");
    da.TableMappings.Add("Table3", "Address");
    da.TableMappings.Add("Table4", "Contact");

    // Fill the DataSet.
    da.Fill(ds);

    // Add data relations.
    DataTable orderHeader = ds.Tables["SalesOrderHeader"];
    DataTable orderDetail = ds.Tables["SalesOrderDetail"];
    DataRelation order = new DataRelation("SalesOrderHeaderDetail",
                             orderHeader.Columns["SalesOrderID"],
                             orderDetail.Columns["SalesOrderID"], true);
    ds.Relations.Add(order);

    DataTable contact = ds.Tables["Contact"];
    DataTable orderHeader2 = ds.Tables["SalesOrderHeader"];
    DataRelation orderContact = new DataRelation("SalesOrderContact",
                                    contact.Columns["ContactID"],
                                    orderHeader2.Columns["ContactID"], true);
    ds.Relations.Add(orderContact);
}
catch (SqlException ex)
{
    Console.WriteLine("SQL exception occurred: " + ex.Message);
}
```

X.5 Truy vấn các DataSet

Sau khi một đối tượng DataSet đã được đổ dữ liệu lên, bạn có thể bắt đầu truy vấn trên nó. Đưa vào một câu truy vấn LINQ để truy vấn với DataSet là tương tự bằng cách sử dụng LINQ dựa trên các dữ liệu ngồn mà LINQ cho phép. Hãy nhớ rằng, tuy nhiên, khi bạn sử dụng LINQ truy vấn trong một đối tượng DataSet bạn là một câu hỏi sự liệt kê của các đối tượng DataRow , thay vì một sự liệt kê của một loại tùy thích. Điều này có nghĩa là bạn có thể sử dụng bất kỳ của các thành viên lớp DataRow trong truy vấn LINQ của bạn . Điều này cho phép bạn để tạo ra các truy vấn phong phú và phức tạp.

Như với các sự triển khai của LINQ, bạn có thể tạo các truy vấn LINQ to DataSet trong hai hình thức khác nhau: cú pháp biểu thức truy vấn và cú pháp truy vấn dựa trên phương thức. Bạn có thể sử dụng cú pháp biểu thức truy vấn hoặc cú pháp truy vấn dựa trên phương thức để thực hiện các truy vấn trên cùng một bảng trong một DataSet, dựa trên nhiều bảng trong một DataSet, hay dựa trên trong một bảng kiểu DataSet.

X.6 Để truy vấn trong LINQ to DataSet.

Một truy vấn là một biểu thức lấy ra dữ liệu từ một nguồn dữ liệu. Truy vấn thường được thể hiện rõ ràng trong một ngôn ngữ truy vấn chuyên dụng, chẳng hạn như SQL cho các cơ sở dữ liệu qua hệ và XQuery cho XML. Vì vậy, người phát triển đã có để tìm hiểu một ngôn ngữ truy vấn mới cho từng loại hình dữ liệu nguồn hoặc định dạng dữ liệu mà họ yêu cầu tìm kiếm. Ngôn ngữ tích hợp truy vấn (LINQ) đưa ra một mô hình đơn giản, nhất quán để làm việc với các dữ liệu trên các loại dữ liệu nguồn và các định dạng. Trong một truy vấn LINQ, bạn luôn luôn làm việc với các chương trình đối tượng.

Một truy vấn LINQ hoạt động bao gồm ba hành động: nhận các nguồn hoặc các nguồn dữ liệu, tạo các truy vấn, và thực hiện các truy vấn.

Dữ liệu nguồn được thực hiện chung giao diện IEnumerable <T> có thể được truy vấn thông qua LINQ. Gọi AsEnumerable trên một DataTable trả về một đối tượng nào mà thi hành chung interface IEnumerable <T>, phục vụ như là các dữ liệu nguồn cho LINQ để truy vấn DataSet.

Trong truy vấn, bạn xác định chính xác của thông tin mà bạn muốn lấy từ nguồn dữ liệu. Một truy vấn như thế nào cũng có thể chỉ định rằng thông tin phải được sắp xếp, gom nhóm, và nó được hình thành trước khi nó được trả về. Trong LINQ, một truy vấn được lưu giữ trong một biến. Nếu các truy vấn được thiết kế để trả về một chuỗi của các giá trị, các biến truy vấn phải là một kiểu liệt kê. Biến truy vấn này biến mất không có hành động và không trả về dữ liệu; nó chỉ dự trữ trong máy tính thông tin truy vấn. Sau khi bạn tạo ra một truy vấn mà bạn cần phải thực hiện truy vấn, để trả về bất kỳ dữ liệu nào.

Trong một truy vấn mà sẽ trả về một chuỗi của các giá trị, các biến truy vấn chính nó không bao giờ chứa kết quả truy vấn chỉ lữu trữ trong máy tính thông tin truy vấn và các lệnh truy vấn. Sự thực thi các truy vấn là hoãn lại cho đến khi các biến truy vấn lặp lại trong một vòng lặp foreach. Điều này được gọi là thực hiện chậm; có nghĩa là, thực hiện truy vấn thời gian sau khi xảy ra một số truy vấn được xây dựng. Điều này có nghĩa

là bạn có thể thực hiện một truy vấn như thường xuyên như bạn muốn. Điều này rất hữu ích khi, ví dụ, bạn có một cơ sở dữ liệu mà đang được cập nhật bởi các ứng dụng khác. Trong ứng dụng của bạn, bạn có thể tạo một truy vấn, để lấy những thông tin mới nhất và liên tục thực hiện các truy vấn, trả về mỗi lần thông tin cập nhật .

Ngược lại chậm truy vấn, mà trả về một chuỗi của các giá trị, các truy vấn mà trả về một giá trị được thực hiện ngay lập tức. Một số ví dụ về truy vấn trả về giá trị duy nhất đó là Count, Max, Average, và First. Những thực hiện ngay lập tức bởi vì kết quả truy vấn được yêu cầu để tính toán kết quả duy nhất. Ví dụ, để tìm kết quả truy vấn trung bình phải được thực hiện như vậy mà các chức năng trung bình có dữ liệu đầu vào để làm việc với. Bạn cũng có thể sử dụng các ToList (Tsource) hoặc ToArray (TSource) các phương thức trên một truy vấn để thực thi ngay lập tức một truy vấn mà không đưa ra một giá trị duy nhất. Những kĩ thuật này để thực hiện ngay lập tức có thể là hữu ích khi bạn muốn cache kết quả của một truy vấn.

Sau đây là một đoạn mã ứng dụng mô tả một truy vấn trên DataSet. Ví dụ sau sử dụng Chọn để trở lại tất cả các dòng sản phẩm từ bảng và hiển thị các sản phẩm.

```
// Fill the DataSet.
DataSet ds = new DataSet();
ds.Locale = CultureInfo.InvariantCulture;
FillDataSet(ds);

DataTable products = ds.Tables["Product"];

IEnumerable<DataRow> query =
    from product in products.AsEnumerable()
    select product;

Console.WriteLine("Product Names:");
foreach (DataRow p in query)
{
    Console.WriteLine(p.Field<string>("Name"));
}
```

XI. Tài liệu tham khảo:

Tài liệu được trích từ MSDN Visual Studio2008.

Thư viện MSDN online: http://msdn.microsoft.com/en- us/vbasic/aa904594.aspx

Nutshell online: http://www.albahari.com/nutshell/linqquiz.aspx