ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA CÔNG NGHỆ PHẦN MỀM**



BÁO CÁO ĐỒ ÁN TÌM HIỂU VỀ LINQ

ĐỒ ÁN MÔN CÔNG NGHỆ .NET

LỚP: SE310.G22

**GVHD:** Th.S PHẠM THI VƯƠNG.

**SVTH:**

12520135 ĐỖ TRUNG HIẾU.

12520227 NGUYỄN THỊ LINH.

12520000 CAO HỮU TRỌNG.

12520000 NGUYỄN ANH DUY.

Thành phố Hồ Chí Minh 27/6

**NHẬN XÉT CỦA GIẢNG VIÊN**

**Mục lục**

[***I.*** ***GIỚI THIỆU LINQ.*** 1](#_Toc455040807)

[***1.*** ***LINQ là gì?*** 1](#_Toc455040808)

[***2.*** ***Chức năng và ứng dụng của LINQ*** 1](#_Toc455040809)

[***3.*** ***Cấu trúc thành phần của LINQ.*** 2](#_Toc455040810)

[***3.1.*** ***LINQ to Object:*** 3](#_Toc455040811)

[***3.2.*** ***LINQ to SQL:*** 3](#_Toc455040812)

[***3.3.*** ***LINQ to XML:*** 3](#_Toc455040813)

[***3.4.*** ***LINQ to DataSet:*** 4](#_Toc455040814)

[***4.*** ***So sánh LINQ TO SQL với các công nghệ khác.*** 4](#_Toc455040815)

[***4.1.*** ***LINQ to SQL với Stored Procedures.*** 4](#_Toc455040816)

[***4.2.*** ***LINQ to SQL với ADO.NET Entity Framework.*** 5](#_Toc455040817)

[***II.*** ***CÁC THÀNH PHẦN CỦA LINQ:*** 6](#_Toc455040818)

[***1.*** ***LINQ to Object:*** 6](#_Toc455040819)

[***1.1.*** ***Khái niệm:*** 6](#_Toc455040820)

[***1.2.*** ***Những object được dùng*** 6](#_Toc455040821)

[***1.3.*** ***Hiệu năng*** 11](#_Toc455040822)

[***2.*** ***LINQ to SQL.*** 14](#_Toc455040823)

[***2.1.*** ***Khái niệm:*** 14](#_Toc455040824)

[***2.2.*** ***Mô hình hóa dữ liệu:*** 15](#_Toc455040825)

[***2.3.*** ***Thực thể trong LINQ to SQL:*** 16](#_Toc455040826)

[***3.2.*** ***Binding metadata:*** 29](#_Toc455040827)

[***3.2.2.*** ***Xóa CSDL:*** 29](#_Toc455040828)

[***3.3.*** ***Stored Procedure (SPROCs):*** 29](#_Toc455040829)

[***4.*** ***LINQ to XML:*** 44](#_Toc455040830)

[***4.1.*** ***Giới thiệu:*** 44](#_Toc455040831)

[***4.2.*** ***LINQ to XML API*** 45](#_Toc455040832)

[***4.3.*** ***Những thao tác truy vấn cơ bản trên cây XML:*** 48](#_Toc455040833)

[***4.4.*** ***Những thao tác biến đổi trên cây XML:*** 52](#_Toc455040834)

[***5.*** ***LINQ to DataSet:*** 55](#_Toc455040835)

[***5.1.*** ***Giới thiệu:*** 55](#_Toc455040836)

[***5.2.*** ***Sử dụng LINQ nạp DataSet:*** 56](#_Toc455040837)

[***5.3.*** ***Sử dụng LINQ truy vấn DataSet:*** 57](#_Toc455040838)

[***5.4.*** ***Sử dụng LINQ để Truy vấn một DataSet nhập liệu:*** 58](#_Toc455040839)

[***5.5.*** ***Truy cập dữ liệu DataSet không định kiểu:*** 59](#_Toc455040840)

[***III.*** ***KẾT LUẬN:*** 59](#_Toc455040841)

[***IV.*** ***TÀI LIỆU THAM KHẢO*** 60](#_Toc455040842)

# ***GIỚI THIỆU LINQ.***

## ***LINQ là gì?***

LINQ là từ viết tắt của **“Language-Integrated Query”.**

LINQ là một thành phần trong gói .Net của Microsoft. Nó có thể thêm câu hỏi truy vấn dữ liệu vào ngôn ngữ .NET với cấu trúc gần giống với cấu trúc SQL.

LINQ được phát triển từ năm 2003, và được Microsoft công bố tại Hội nghị Microsoft Professional Developers vào năm 2005.

Theo định nghĩa trên thư viên MSDN của Microsoft, LINQ là một tập hợp các phần mở rộng của .Net Frame Work. LINQ bao gồm ngôn ngữ tích hợp truy vấn, thiếp lập và chuyển đổi hoạt động. LINQ là một tập hợp các thành phần mở rộng cho phép viết các câu truy vấn dữ liệu ngay trong một ngôn ngữ lập trình, như C# hoặc VB.NET. Sự tích hợp của LINQ nhằm loại bỏ rào cản giữa các ngôn ngữ lập trình và các cơ sở dữ liệu.

Trước khi có LINQ, khi lập trình chúng ta phải kết hợp các ngôn ngữ khác nhau như SLQ, XML, Xpath cùng với C#, VB.NET, việc này có nhiều nhược điểm.Để giảm gánh nặng thao tác trên nhiều ngôn ngữ khác nhau và cải thiện năng suất lập trình, Microsoft đã phát triển giải pháp tích hợp dữ liệu cho .NET Framework có tên gọi là LINQ (Language Integrated Query), đây là thư viện mở rộng cho các ngôn ngữ lập trình C# và Visual Basic.NET (có thể mở rộng cho các ngôn ngữ khác) cung cấp khả năng truy vấn trực tiếp dữ liệu Object, CSDL và XML.

## ***Chức năng và ứng dụng của LINQ***

LINQ là thư viện cho phép truy vấn trên nhiều kiểu dữ liệu khác nhau: Object collections, XML, LINQ to SQL (truy vấn cơ sở dữ liệu MS-SQL), LINQ to Dataset, LINQ to Entity (ADO.net Entity Framework), LINQ to SharePoint, LINQ to XML, LINQ to nHIbernate, LINQ to XYZ, LINQ to Azure SQL Service....

Trước đây, cách phổ biến nhất để ứng dụng lấy dữ liệu từ các hệ cơ sở dữ liệu (CSDL) là sử dụng SQL (Structure Query Language - ngôn ngữ truy vấn cấu trúc). SQL có cú pháp rất khác với những ngôn ngữ lập trình phổ dụng như C# và VB.NET, do vậy bạn phải nhọc công "hàn gắn" hai thực thể khác biệt này với nhau trong mỗi dự án phần mềm. LINQ ra đời để giảm gánh nặng thao tác "hàn gắn" trên nhiều ngôn ngữ khác nhau.

Một vấn đề khác với SQL là nó chỉ dùng để truy vấn dữ liệu trong các CSDL dạng quan hệ. Nếu muốn truy cập dữ liệu XML hay dạng khác (như trang HTML, email...), bạn lại phải sử dụng cú pháp truy vấn khác (XPath/XQuery). Cách giải quyết vấn đề tốt nhất hiện nay là LINQ.

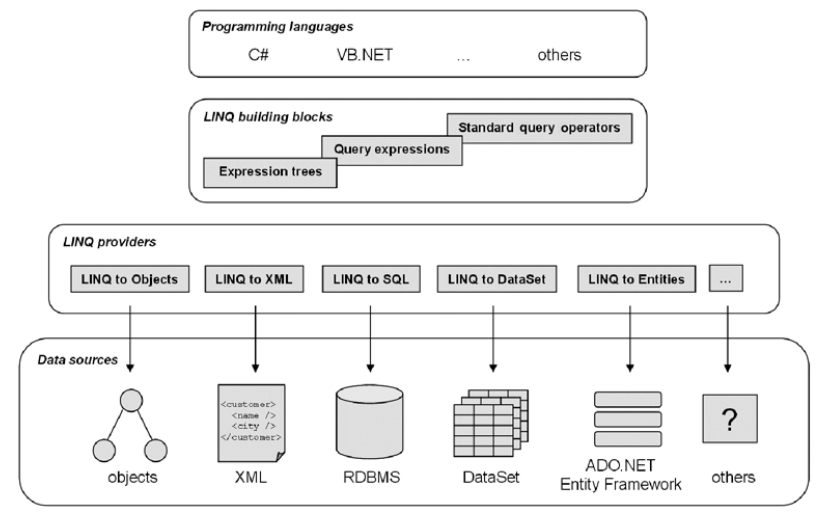
Về vấn đề cú pháp truy vấn thì trước khi có LINQ chúng ta chưa có một cú pháp chung nào cho truy vấn dữ liệu từ những nguồn khác nhau. Với LINQ nó sẽ cung cấp cách duy nhất để truy cập dữ liệu từ bất kể nguồn dữ liệu nào với cú pháp giống nhau.

LINQ đem đến các lợi ích:

* LINQ đưa ra cùng một cấu trúc để truy vấn các kiểu dữ liệu từ các nguồn khác nhau, cụ thể hơn là giờ đây bạn có thể truy vấn một tài liệu XML cùng một cách với truy vấn một CSDL SQL, một ADO.NET dataset, 1 đối tượng Collection hoặc bất cứ kiểu dữ liệu nào khác mà bạn đã chọn để kết nối và làm việc sử dụng LINQ.
* LINQ chính là cầu nối, thắt chặt mối tương quan giữa dữ liệu quan hệ và thế giới hướng đối tượng (các bạn quan tâm có thể tìm hiểu thêm về ORM nếu muốn biết rõ hơn về mối tương quan này cũng như các vấn đề liên quan).
* LINQ giảm thiểu khá nhiều thời gian phát triển sản phẩm phần mềm thông qua việc bắt được khá nhiều lỗi trong quá trình biên dịch và hỗ trợ cả debug.
* Các câu lệnh của LINQ được định kiểu rõ ràng (khác với các câu lệnh SQL truyền thống).

## ***Cấu trúc thành phần của LINQ.***

LINQ bao gồm nhiều thành phần, nó cung cấp cho chúng ta rất nhiều chức năng, *hình 1.1* biểu diễn cấu trúc những thành phần cơ bản của LINQ:



*Hình 1.1 – Cấu trúc cơ bản của LINQ*

* 1. ***LINQ to Object:***

LINQ to Objects là tên được gán cho các API về IEnumrable<T> dùng cho các thao tác truy vấn dữ liệu chuẩn (Standard Query Operations). Cho phép bạn thực thi các truy vấn trên các mảng, các collection trong bộ nhớ.

Các thao tác truy vấn dữ liệu chuẩn này là các phương thức static của lớp static System.Linq.Enumrable. Bạn sử dụng các phương thức này để tạo ra các truy vấn LINQ to Object.

* 1. ***LINQ to SQL:***

Là tên gọi được đặt cho API IQueryable<T>, cho phép LINQ làm việc với cơ sở dữ liệu Microsoft SQL Server. Để sử dụng bạn phải tham chiếu đến System.Data.Linq.

* 1. ***LINQ to XML:***

LINQ to XML là tên được gán cho API chuyên dụng làm việc với XML. Microsoft không chỉ thêm các thư viện XML cần thiết để làm việc với LINQ, họ còn nhắm đến các nhược điểm của XML DOM chuẩn. Bằng cách đó chúng ta có thể thao tác với XML dễ dàng hơn rất nhiều so với trước đây. Chúng ta sẽ không còn phải tạo ra cả một XmlDocument chỉ để làm việc với một mẫu nhỏ XML. Để sử dụng LINQ to XML chúng ta phải tham chiếu đến System.Xml.Linq.

* 1. ***LINQ to DataSet:***

**LINQ to DataSet** cho phép người lập trình sử dụng DataSet như một nguồn dữ liệu bình thường bằng các cú pháp truy vấn căn bản của LINQ.

**LINQ to DataSet** hỗ trợ truy vấn dữ liệu thông qua ADO.NET với 2 đối tượng là DataSet và DataTable.

## ***So sánh LINQ TO SQL với các công nghệ khác.***

### ***LINQ to SQL với Stored Procedures.***

|  |  |  |
| --- | --- | --- |
| **Đặc điểm** | **Linq to SQL** | **Stored Procedures** |
| Độ phức tạp | Đơn giản khi code, ít dòng lệnh, không cần biết T-SQL và data access API | Phức tạp khi code |
| Hiệu suất |  | Được tiền biên dịch trong database nên tốc độ rất nhanh |
| Hỗ trợ debugging | Dễ dàng với .NET debugger | Khó khăn |
| Network traffic | LINQ gửi toàn bộ | Chỉ cần truyền mã số sproc-name và đối số dữ liệu |
| Tính linh hoạt | Trói buộc với nền tảng hỗ trợ | Tiếp cận đầy đủ tính năng của database |
| Biên dịch lại | Nếu muốn thay đổi cách truy suất dữ liệu thì cần biên dịch lại, version, và redeploy assembly | Một số trường hợp cho phép một DBA đồng bộ data access routine mà không cần redeploy |
| Bảo mật | Bảo vệ dữ liệu bằng cách ngăn truy cập vào các bảng và đặt ACLs vào table views | Bảo vệ dữ liệu bằng cách ngăn truy cập vào các bảng trực tiếp và đặt ACLs và sprocs |
| Khả năng quản lí |  | Sử dụng views cho phép tạo lớp chắn chương trình của bạn khỏi đổ vỡ khi có sự thay đổi lược đồ. Bạn có thể update view không cần thay đổi code data access |

### ***LINQ to SQL với ADO.NET Entity Framework.***

|  |  |  |
| --- | --- | --- |
| **Đặc điểm** | **LINQ to SQL** | **Entity Framework** |
| Model | Domain model | Khái niệm data model |
| Hỗ trợ databases | SQL | Nhiều dạng database |
| Data sources | tables | Tables, replication, reporting services, BI and etc |
| Độ phức tạp | Đơn giản | Phức tạp |
| Thời gian phát triển | Nhanh chóng | Chậm hơn nhưng dung lượng lớn hơn |
| Mapping | Class to single table | Class to multiple tables |
| Kế thừa | Khó khăn | Đơn giản |
| File types | Dbml file | Sau khi biên dịch tạo file edmx với 3 phần đai diện lược đồ: csdl, msl, và ssdl |
| Có thể tạo thuộc tính phức tạp | Không hỗ trợ | Hỗ trợ VS 2010 nhưng có thể tùy chỉnh file .edmx bằng tay |
| Query | 1. LINQ to SQL (for select) 2. Data context (for update, create, delete, store produre, view) | 1. LINQ to Entities (for select) 2. Entity SQL (dẫn xuất of Transact-SQL, hỗ trợ kế thừa và quan hệ) 3. Object services (for update, create, delete, store procedure, view) 4. Entity Client (cung cấp quản lí ADO.NET, giống như SQL Client.. Nó cung cấp nhiều components như ENtityComand, EntityTransaction) |
| Language extensions support | có | có |
| Khả năng đồng bộ với database nếu lược đồ database thay đổi | Không hỗ trợ | Hỗ trợ |
| Hiệu suất | Rất chậm với truy vấn đầu tiên | Rất chậm với truy vấn đầu tiên nhưng hiệu suất trung bình tốt hơn LINQ to SQL |
| Tiếp tục được phát triển | Không | Có |
| Tạo database từ entity model | Không | Hỗ trợ ở VS 2010 |

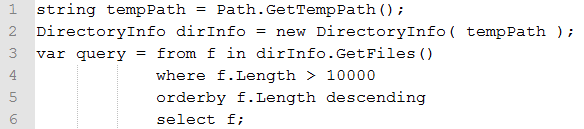
# ***CÁC THÀNH PHẦN CỦA LINQ:***

## ***LINQ to Object:***

* 1. ***Khái niệm:***

LINQ to Objects là một thành phần của LINQ làm việc với tập hợp các object trong bộ nhớ; nó có mục tiêu là điều khiển tập hợp các object có liên quan đến nhau để hình thành hệ thống hoặc biểu đồ. LINQ to Objects cài đặt mặc định sử dụng truy vấn LINQ. LINQ to Objects được sử dụng qua namespace System.Linq.

LINQ to Objects không giới hạn tập hợp dữ liệu. Ví dụ về truy vấn LINQ lấy thông tin từ file system. Danh sách file được đọc trước từ bộ nhớ trước khi lọc bởi LINQ query. LINQ query lấy các file tạm lớn hơn 10,000 bytes, sắp xếp theo kích cỡ:

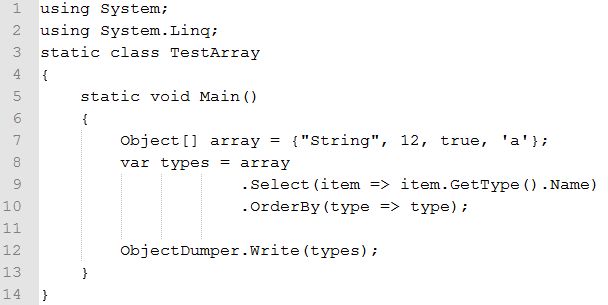


* 1. ***Những object được dùng***

Không phải mọi thứ đều có thể truy vấn với LINQ to Objects. Tiêu chuẩn đầu tiên đó là object phải có dạng collections implement từ IEnumerable<T> interface. Object implement từ IEnumerable <T> được gọi là sequences trong LINQ. Tin tốt là hầu như generic collections cung cấp bởi .NET framework đều implement từ IEnumerable<T>.

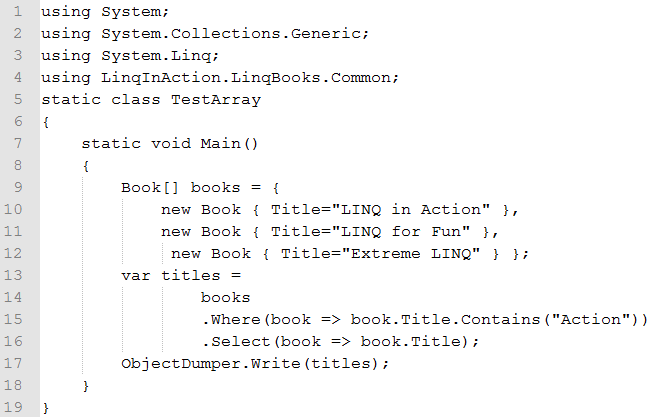
#### **Arrays:**

Mọi loại array đều được hỗ trợ Boolean, char, int32, string, kể cả untyped array như trong ví dụ sau:



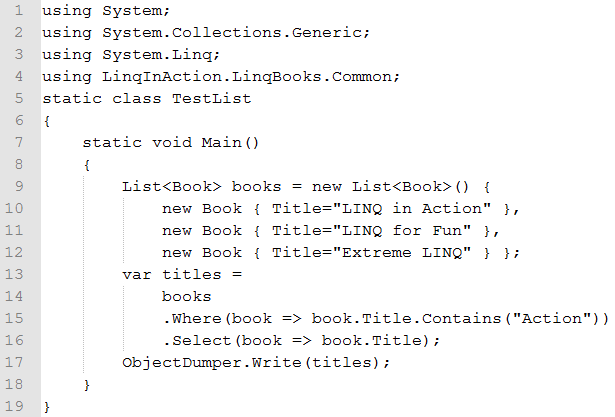
Lớp ObjectDumper là một lớp hữu dụng để biểu diễn kết quả. Nó được Microsoft cho phép như là một phần của LINQ code.

Ngoài ra truy vấn có thể áp dụng cho cả arrays của các objects do người dùng tạo ra.



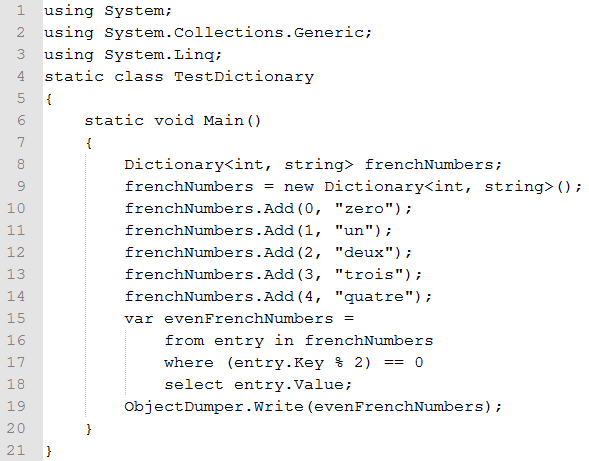
#### **Generic lists:**

LINQ to Objects có thể hoặc động tốt với tất cả generic lists: List<T>, LinkedList<T>, Queue<T>, Stack<T>, HashSet<T>, ObjectModel.Colection<T>, ComponentModel.BindingList<T>.



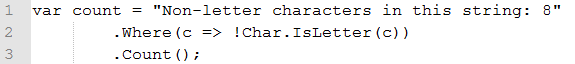
#### **Generic dictionaries:**

Cũng như generic lists, mọi loại generic dictionaries đều được truy vấn với LINQ to Objects: Dictionary<TKey, TValue>, SortedDictionary<TKey, Tvalue>, SortedList<TKey, TValue>.



#### **String:**

Mặc dù System.String có thể không được hiểu là một collection, nhưng thực ra nó đúng là một collection vì nó implement IEnumberable<Char>. Nó cũng có thể được truy vấn bằng LINQ to Objects.

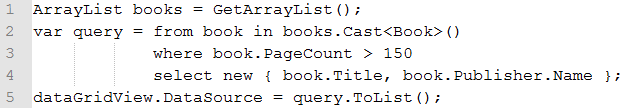


Kết quả của câu truy vấn là 8.

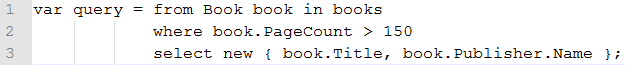
#### **Nongeneric collection:**

Tuy nhiên không phải tất cả lớp đều là generic. Ví dụ điển hình là ArrayList vốn được sử dụng rất nhiều trước khi có các generic collection.

ArrrayList không được LINQ to Object hỗ trợ. Tuy nhiên nhiều trường hợp chúng ta cần lấy kết quả từ các hàm trả về nongeneric colletion. Một cách để giải quyết đó là sử dụng hàm Cast. Cast giúp chuyển nongeneric IEnumberable thành IEnumberic<T>.



Nhờ vào đặc tính của query expression, chúng ta có thể sử dụng một cách khác mà không cần dùng hàm Cast. Đó là lặp các biến Book từ tập book.

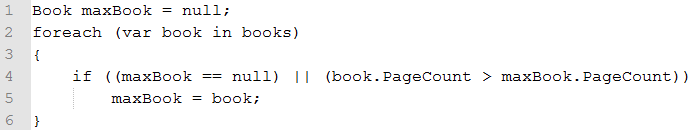


* 1. ***Hiệu năng***

LINQ không chỉ cung cấp sự mới mẻ mà còn mang lại sự đơn giản và súc tích. Đôi khi LINQ to Objects không thể cung cấp ngay được cái chúng ta cần. Như trong ví dụ sau của Jon Skeet: giả sử chúng ta có một collection của objects và cần tìm xem object nào có giá trị lớn nhất. Chúng ta có thể nghĩ đến hàm Max, một hàm chuẩn của truy vấn, nhưng hàm Max không hữu ích lắm vì nó chỉ trả về giá trị lớn nhất chứ không phải object có giá trị đó. Trường hợ này đặt ra cho lậpt trình viên nhiều lựa chọn thiết kế, bao gồm cả sử dụng LINQ. Các lựa chọn Jon Skeet đưa ra bao gồm:

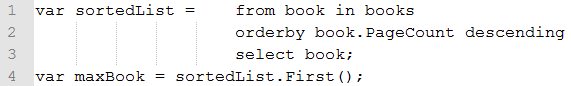
(Bài toán đuợc cụ thể hóa là chọn cuốn sách có số trang nhiều nhất trong một tập hợp sách)

Vòng lặp foreach:



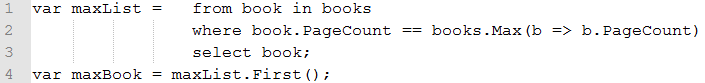
Một cách làm thẳng và nhanh. Độ phức tạp O(n)

Cách thứ hai là dùng LINQ để sắp xếp theo thứ tự gảim dần và chọn phần tử đứng đầu:



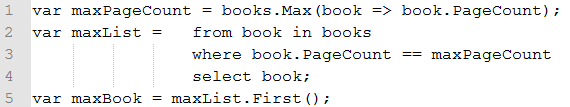
Cách này có nhược điểm là phải xắp xếp trước lấy được kết quả. Độ phức tạp O(n log n)

Cách thứ ba là sử dụng subquery, hàm Max, để tìm giá trị lớn nhất rồi tiến hành so sánh lần lượt giá trị các object với giá trị lớn nhất để chọn ra phần tử có giá trị lớn nhất:



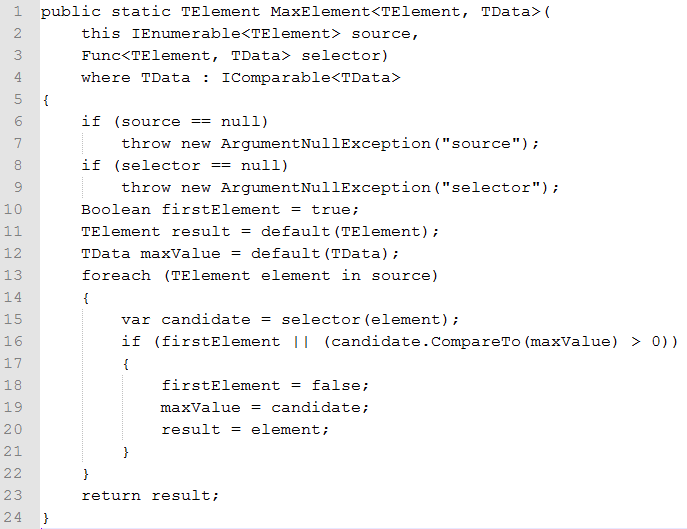
Cách này đòi hỏi tính toán so sánh trên tất cả các bước. Độ phức tạp O(n2)

Cách thứ tư là sử dụng 2 câu truy vấn riêng:



Cách này giống cách thư ba nhưng cải thiện việc lặp lại gọi hàm Max. Độ phức tạp O(n)

Cách cuối cùng là sử dụng hàm truy vấn tự định nghĩa, được dặt tên là MaxElement:



Sau đó gọi hàm trong truy vấn:

****

Bảng sau cho thấy kết quả đo được khi cho chạy thử 20 lần:

|  |  |  |  |
| --- | --- | --- | --- |
| Option | Thời gian rung bình (ms) | Thời gian ít nhất (ms) | Thời gain nhiều nhất (ms) |
| Foreach | 37 | 35 | 42 |
| OrderBy + First | 1724 | 1704 | 1933 |
| Sub-query | 37482 | 37201 | 45233 |
| Two queries | 66 | 65 | 69 |
| Custom operator | 56 | 54 | 73 |

Có thể thấy là chi phí tính toán sẽ nhỏ hơn nếu chỉ duyệt qua collection một lần. Hàm tự định nghĩa chạy tương đối nhanh nhưng vẫn thua cách không cần LINQ. Ví dụ trên cho thấy chúng ta cần suy nghĩ về sự phức tạp trong truy vấn của LINQ to Objects; bởi vì nó đi kèm với lists và vòng lặp. Cần chú ý tránh viết query tính toán trên collection nhiều hơn một lần, nếu không hiệu năng sẽ giảm xuống.

Chúng ta cũng nhận thây rằng độ hiệu quả của LINQ ảnh hưởng khá nhiều bởi query. Query được viết tốt sẽ giảm thiểu phí tổn rất nhiều.

Cuối cùng chúng ta cần nhận ra rằng cần LINQ to Object một cách khôn ngoan vì trong nhiều trường hợp sử dụng những cách tiếp cận truyền thống như vòng lặp for và foreach. Nhiều trường hợp chỉ cần dựa vào hàm chuẩn của LINQ, tuy nhiên cũng có những trường hợp sử dụng hàm tự định nghĩa sẽ hiệu quả hơn.

## ***LINQ to SQL.***

### ***Khái niệm:***

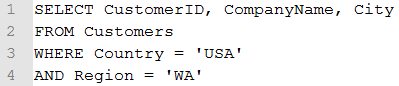
LINQ to SQL là một thành phần của LINQ cho phép truy vấn CSDL liên quan***,*** cung cấp cho lập trình viên một mẫu đối tượng dựa trên các thực thể có sẵn.

LINQ to SQL hỗ trợ đầy đủ transaction, view và các stored procedure (SP). Nó cũng cung cấp một cách dễ dàng để thêm khả năng kiểm tra tính hợp lệ của dữ liệu và các quy tắc vào trong mô hình dữ liệu của bạn.

Một ví dụ truy vấn đơn giản:



Câu truy vấn này sẽ được chuyển sang truy vấn SQL gửi cho cơ sở dữ liệu (viết tắt là CSDL):



***Chú ý: để sử dụng các mở rộng của LINQ to SQL, cần có khai báo using System.Data.Linq***

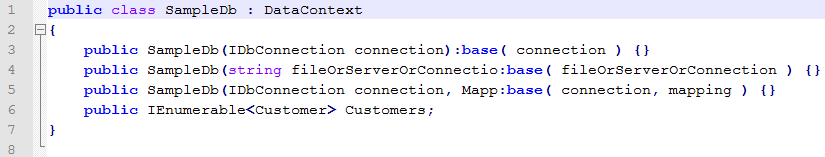
### ***Mô hình hóa dữ liệu:***

Lớp thực thể mà LINQ to SQL cần là một lớp trừu tượng trên mô hình quan hệ. Mỗi lớp thực thể định nghĩa một bảng dữ liệu có thể truy suất. Nếu đối tượng của lớp bị sửa thì có thể cập nhật thay đổi này lên cơ sở dữ liệu.

#### **DataContext:**

Lớp kiểm soát liên hệ giữa LINQ và cơ sở dữ liệu. Mỗi đối tượng là một liên kết riêng đến cơ sở dữ liệu có dạng IDbConection.

Lớp cần truy cập đặc biệt vào CSDL cụ thể có thể thừa kế từ DataContext. Những lớp này cung cấp đường dễ hơn để truy cập CSDL. Bạn có thể định nghiã một bảng có sẵn bằng cách khai báo, không cần khởi tạo, như ví dụ:



Bảng được khởi tạo tự động bởi DataContext contructor.

#### **Tạo ra một mô hình dữ liệu LINQ to SQL:**

Bạn có thể thêm một mô hình dữ liệu LINQ to SQL và một dự án ASP.NET, Class Library hay Windows bằng cách dùng tùy chọn “InsertOnSubmit New Item” bên trong Visual Studio và chọn “LINQ to SQL”:

Việc chọn mục “LINQ to SQL” sẽ khởi chạy LINQ to SQL designer, và cho phép bạn mô hình hóa các lớp mà nó biểu diễn một CSDL quan hệ. Nó cũng sẽ tạo ra một lớp kiểu “DataContext”, trong đó có các thuộc tính để biểu diễn mỗi bảng mà chúng ta mô hình hóa trong CSDL, cũng như các phương thức cho mỗi Stored Procedure mà chúng ta mô hình hóa. Lớp DataContext là thành phần trung tâm của mô hình, toàn bộ các thao tác truy vấn hoặc cập nhật dữ liệu đều được thực hiện thông qua lớp này.

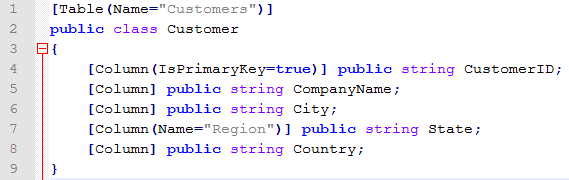
### ***Thực thể trong LINQ to SQL:***

#### **Định nghĩa:**

LINQ to SQL cho phép bạn mô hình hóa các lớp ánh xạ vào CSDL. Các lớp này thường được là “Entity Class” (lớp thực thể) và các instance của nó thường được gọi là “Entity” (thực thể). Các lớp entity ánh xạ vào các bảng bên trong một CSDL. Các thuộc tính của các lớp thông thường ánh xạ vào các cột trong bảng. Mỗi instance của một lớp thực thể biểu diễn một dòng trong bảng.

Bất cứ dữ liệu ngoài nào cũng phải được mô tả với định dạng lớp thich hợp. Mỗi bảng phải được huyển đổi thành định nghĩa lớp với những thuộc tính cụ thể tương đương các cột của dữ liệu và tất cả cột có trong dữ liệu đều phải được định nghĩa kiểu. Chỉ những kiểu đã được mộ tả mới có thể dùng trong LINQ cho việc chiếu và lọc.

Ví dụ định nghĩa một thực thể đơn giản:



Cấu trúc Customer định nghĩa nội dung của một dòng, và mỗi thuộc tính biểu diễn một cột trong bảng. Thuộc tính IsPrimaryKey đặt là true cho biết cột đó là khóa chính của bảng. Một lớp thực thể cần có khóa chính để định danh. Nếu không có, đối tượng của lớp có thể được tạo ra nhưng không thể sửa. Tham số Name chỉ ra tên của cột khác so với tên của thuộc tính (ở đây State tương đương cột Region). Thuộc tính Table chỉ ra lớp này là đại diện cho một thực thể của cơ sở dữ liệu; thuộc tính Name chỉ ra tên của bảng khác tên của thực thể. Thông thường sử dụng danh từ đơn cho tên của lớp (1 dòng) và danh từ số nhiều cho tên của bảng (tập hợp các dòng)

Khi bạn cần định nghĩa một kiểu dữ liệu đặc trưng của CSDL SQL mà .NET không có, sử dụng tham số DBType. Chỉ dùng khi bạn muốn tạo lược đồ dữ liệu từ định nghĩa lớp thực thể.



Nếu giá trị của cột được auto-generated bởi CSDL bạn sẽ cần đồng bộ thành phần của lớp thực thể với giá trị được tự động tạo khi bạn insert một đối tượng vào CSDL. Để làm điều này bạn cần đặt tham số IsDBGenerated là true, và thêm DBType và INDENTITY:



Bảng thuộc tính của cột:

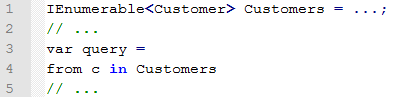
|  |  |
| --- | --- |
| **Tên** | **Mô tả** |
| AutoSync | Mô tả cách LINQ to SQL quản lí thay đổi dữ liệu trên cột sau khi Create hoặc Update, hữu ích cho những cột có giá trị mặc định. Giá trị bao gồm Default, Always, Never, OnInsert, và OnUpdate |
| CanBeNull | Cho phép dữ liệu null |
| BbType | Mô tả dữ liệuở dạng của SQL |
| Expression | Giá trị này chỉ sử dụng khi tạo CSDL với hàm CreateDatabase. Giá trị mô tả cách tính toán trường trong CSDL |
| IsDBGenerated | Cho biết CSDL có tự phát sinh giá trị cho cột này hay không. Đối số này thường dùng cho những cột có thuộc tính Identity hoặc AutoNumber. Dữ liệu sẽ được tạo tự động sau khi chạy hàm SubmitChanges |
| IsDiscriminitor | Dùng để thiết kế cột có định danh dành cho những loại thực thể phân biệt. Mô tả thêm ở các phần sau |
| IsPrimaryKey | Đặt giá trị cột này làm khóa chính của bảng |
| IsVersion | Sử dụng cho cột là timestamp hoặc version number của dữ liệu. Giá trị được cập nhật mỗi lần dòng đó bị thay đổi. |
| Name | Mô tả tên của cột trong bảng mà bạn muốn chuyển dữ liệu. Dùng khi tên cột trong bảng và thuộc tính trong lớp thực thể khác nhau |
| Storage | Chuyển dữ liệu từ cột vào thẳng trường ưu tiên dạng private trong lớp |
| UpdateCheck | Mô tả cách LINQ to SQL sử dụng cột này khi có truy cập đồng thời. Mặc định tất cả chuyển dữ liệu trong cột này được sử dụng để đánh giá khi xảy ra truy cập đồng thời. Giá trị bao gồm:  Always (mặc định) - luôn luôn kiểm tra cột này  Never – không bao giờ kiểm tra  WhenChanged – chỉ kiểm tra nếu thuộc tính thay đổi |

#### **Vai trò:**

Lớp thực thể có 2 vai trò. Thứ nhất cung cấp metadata cho LINQ to SQL truy vấn, trường hợp này không cần khởi tạo. Thứ hai là cung cấp storage lưu trữ dữ liệu, đóng vái trò ghi nhận thay đổi và lưu trở lại CSDL.

#### **Truy vấn:**

Các câu truy vấn LINQ trả về kết quả mà nó sẽ implement interrface IEnumerable – đây cũng là interface mà các control .NET dùng để hỗ trợ gắn nối các đối tượng. Điều này có nghĩa là bạn có thể gắn nối kết quả của bất kỳ câu lệnh LINQ, LINQ to SQL hay LINQ to XML vào bất kỳ control nào.

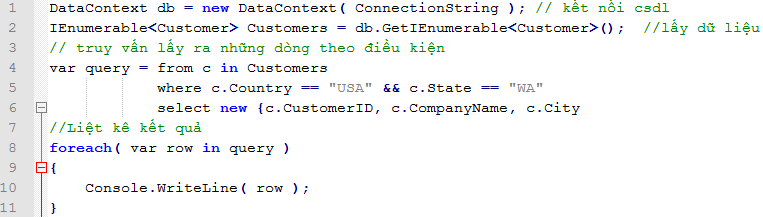


Để tạo truy vấn trên Customers, bạn cần một lớp thực thi IEnumerable<T> sử dung Customer là T.

Đối tượng bảng Customers cần phải được thực thi. Để làm điều đó, cần một đối tượng của lớp DataContext, nó định nghĩa cầu nối giữa LINQ và cơ sở dữ liệu. Hàm GetTable của nó trả về IEnumerable<T> tương đương:



Ví dụ sử dụng đầy đủ code để truy vấn:



Dữ liệu trả về từ truy vấn SQL có dạng row và đặt trong vòng lặp foreach. Sau đó được sử dụng để chiếu thành dạng ẩn danh sử dụng keyword select. Trong ví dụ trên lớp Customer không được thực thi, và được sử dụng chỉ để phân tích dữ liệu. Nếu muốn chúng ta có thể ghi ra câu lệnh SQL sử dụng hàm GetQueryText của lớp DataContext:



Kết quả có được là

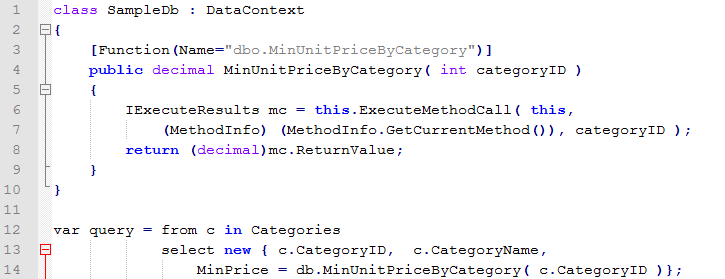


Để kiểm tra tất cả câu lệnh được chuyển cho cơ sở dữ liệu, ta đặt giá trị cho thuộc tính Log của DataContext:

Db.Log = Console.Out;

User-Defined Functions

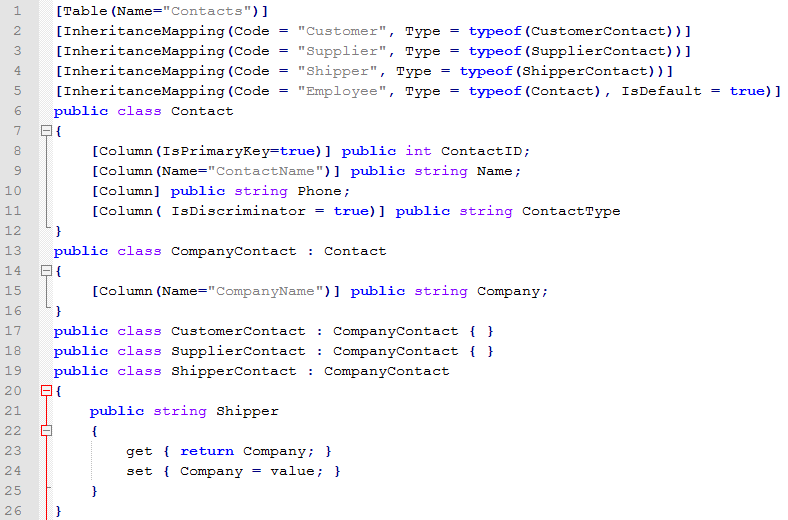
Bạn có thể tạo hàm truy suất do mình tự định nghĩa trong khai báo và sử dụng:



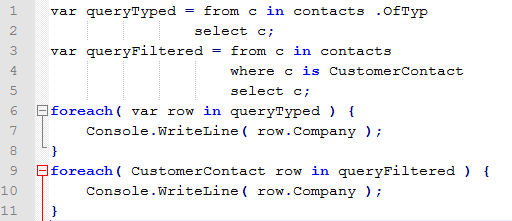
#### **Quan hệ kế thừa:**

Đôi khi một bảng có thể chứa nhiều loại thực thể. Ví dụ bảng Contact chứa thông tin của customers, suppliers, company employees. Mỗi thực thể có thể có những trường riêng biệt. Cách tốt nhất để mô hình hóa loại dữ liệu dạng này trong môi trường hướng đối tượng là tận dụng kế thừa để tạo cấp bậc cho các lớp chuyên biệt. LINQ to SQL cho phép tạo một tập lớp kế thừa từ cùng một lớp cha có cùng quan hệ như nhau.

Thuộc tính InheritanceMapping loại của class dựa vào giá trị của cột discriminator . Tham số Code định nghĩa giá trị cụ thể, và tham số Type định nghĩa liên hệ loại dẫn xuất. Cột discriminator định nghĩa bởi đối số IsDiscriminator là true:



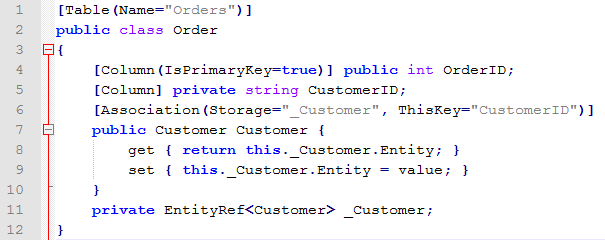
Contact là lớp cha. Nếu contact là Customer, Supplier, Shipper thì lớp liên quan dẫn xuất từ một trung gian CompanyContact, định nghĩa trường Company liên quan đến cột CompanyName của source table. Lớp trung gian CompanyContact cần thiết bởi vì bạn không thể tham khảo cùng một cột (CompanyName) trong nhiều hơn một trường. Lớp ShipperContact định nghĩa một thuộc tính Shipper có cùng giá trị với Company nhưng với một nghĩa khác về từ vựng. Truy suất:



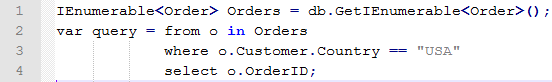
#### **Quan hệ giữa các thực thể:**

##### ***EntityRef:***

Khóa ngoại được định nghĩa với thuộc tính Association và lưu trữ bằng thành phần EntityRef<T> có tên là \_Name.

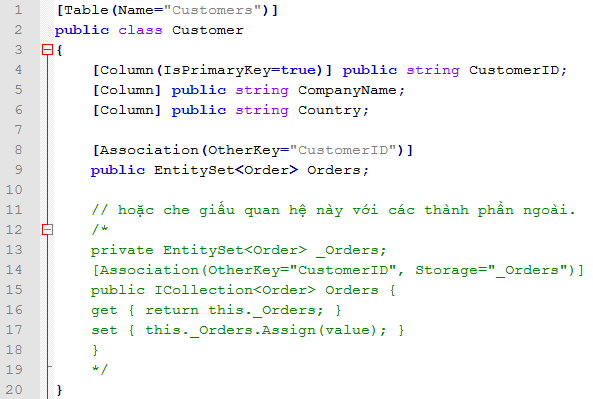


Tham số ThisKey chỉ ra cột chứa khóa ngoại. Sử dụng lớp Order trong câu truy vấn, bạn cố thể lấy ra các thuộc tính của Customer mà không cần việt truy vấn join 2 bảng. NHư ví dụ sau thành phần Country của Customer được sử dụng để lọc yêu cầu từ Customers:

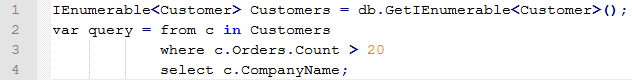


##### **EntitySet:**

Trong trường hợp bảng liên hệ với bảng khác thông qua khóa chính. Ví dụ nếu bảng Customers liên hệ với bảng Orders thông qua khóa chính CustomerID, bạn có thể định nghĩa một thuộc tính Orders ttrong lớp Customers:

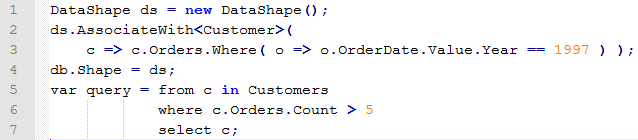


Việc truy vấn cũng tương tự như EntityRef.



##### ***Lọc truy vấn trên quan hệ:***

Trong trường hơp bạn muốn xem những khách hàng nào đặt ít nhất 5 order trong năm 1997. Bạn sử dụng phương thức AssociateWith<T> của lớp DataShape

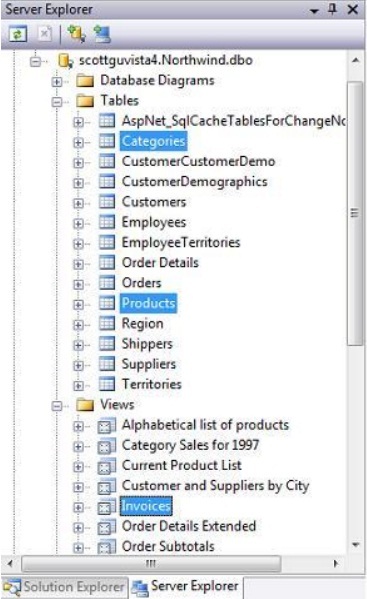


#### **Tạo các lớp thực thể bằng Visual Studio:**

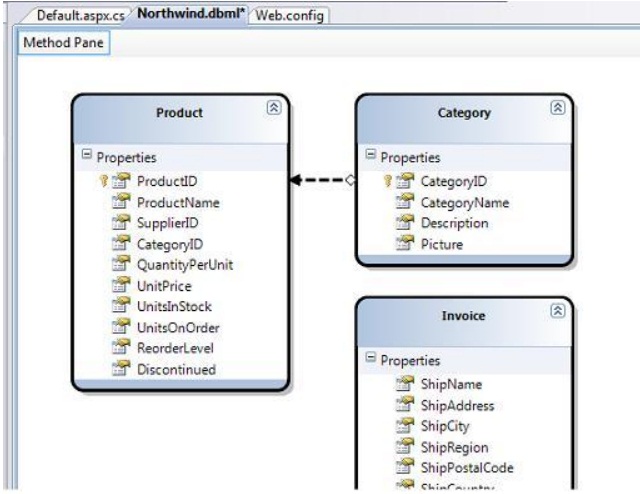
##### ***Tạo lớp thực thể:***

Nếu đã có cấu trúc cho CSDL, bạn có thể dùng nó để tạo các lớp thực thể LINQ to SQL một cách nhanh chóng.

Các dễ dàng nhất để làm điều này là mở CSDL trong cửa sổ Server Explorer bên trong Visual Studio, chọn các table và view mà bạn muốn mô hình hóa, và kéo thả chúng lên trên của sổ LINQ to SQL designer.

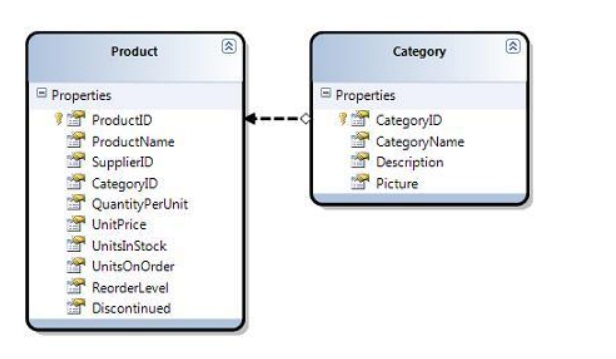


Khi bạn thêm 2 bảng (Categories and Products) và 1 view (Invoices) từ CSDL “Northwind” vào cửa sổ LINQ to SQL designer, bạn sẽ có thêm 3 lớp thực thể được tạo ra một cách tự động:



##### ***Tạo quan hệ giữa các thực thể:***

Khi bạn kéo thả các đối tượng từ Server Explorer lên trên cửa sổ LINQ to SQL designer, VS sẽ tự động xác định các mối quan hệ primary key/foreign key giữa các đối tượng, và tự động tạo các quan hệ mặc nhiên giữa các lớp thực thể khác nhau mà nó đã tạo. Ví dụ, khi bạn thêm cả hai bảng Products và Categories từ Northwind lên trên cửa sổ LINQ to SQL, bạn có thể thấy mội mối quan hệ một nhiều giữa chúng (được biểu diễn bằng một mũi tên trên của sổ soạn thảo):

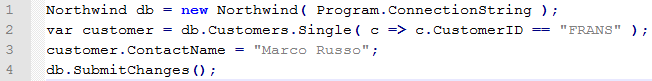


#### **Cập nhật thực thể:**

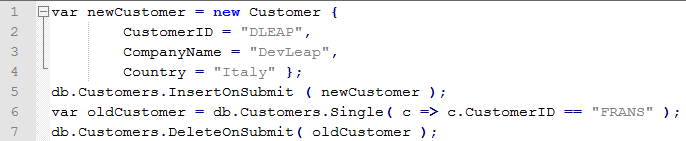
Những thay đổi thuộc tính của một đối tượng đều được ghi nhận bởi một dịch vụ của LLINQ to SQL. Nếu muốn theo dõi, bạn sử dụng hàm GetChangeText của DataContext:



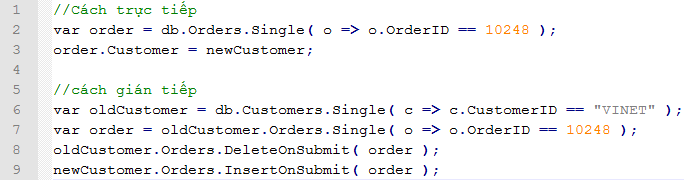
Những thay đổi sẽ không được gửi cho CSDL chừng nào chưa gọi hàm SubmitChanges.



Ta có thể thêm và xóa bằng cách gọi hàm InsertOnSubmit hoặc DeleteOnSubmit.



Cấp nhật quan hệ với đối tượng mới. Ta có 2 cách để làm. Ví dụ khi cập nhật Customer mới cho Oder có sẵn.



#### **Transactions:**

Một transaction (giao dịch) là một dịch vụ được cung cấp bởi một CSDL (hoặc một trình quản lý tài nguyên khác) để đảm bảo rằng một tập các thao tác độc lập sẽ được thực thi như một đơn vị duy nhất – có nghĩa là hoặc tất cả cùng thành công, hoặc cùng thất bại. Và trong trường hợp thất bại, tất cả các thao tác đã là làm sẽ bị hoàn tác trước khi bất kỳ thao tác nào khác được cho phép thực hiện.

Khi gọi SubmitChanges() trên lớp DataContext, các lệnh cập nhật sẽ luôn được thực thi trong cùng một transaction. Có nghĩa là CSDL của bạn sẽ không bao giờ ở trong một trạng thái không toàn vẹn nếu bạn thực thi nhiều câu lệnh – hoặc tất cả các thao tác bạn làm sẽ được lưu lại, hoặc không có bất kỳ thay đổi nào.

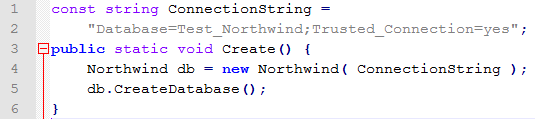
Nếu không có một transaction đang diễn ra, DataContext của LINQ to SQL sẽ tự động bắt đầu một transaction để bảo vệ các thao tác cập nhật khi gọi SubmitChanges(). Thêm vào đó, LINQ to SQL còn cho phép bạn tự định nghĩa và dùng đối tượng TransactionScope của riêng bạn. Điều này làm cho việc tích hợp các lệnh LINQ to SQL vào các đoạn mã truy cập dữ liệu đã có dễ dàng hơn. Nó cũng có nghĩa là bạn có thể đưa cả các tài nguyên không phải của CSDL vào trong cùng transaction.

Ví dụ: bạn có thể gửi đi một thông điệp MSMQ, cập nhật hệ thống file (sử dụng khả năng hỗ trợ transaction cho hệ thống file),… và nhóm tất cả các thao tác đó vào trong cùng một transaction mà bạn dùng để cập nhật CSDL dùng LINQ to SQL.

### ***Binding metadata:***

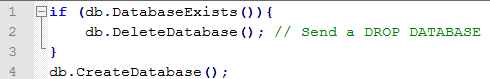
#### **Tạo CSDL:**

Sử dụng hàm CreateDatabase. Hàm này gửi những câu lệnh CREATE DATABASE cần thiết cũng như những câu lệnh CREATE TABLE và ALTER TABLE tiếp sau:



#### **Xóa CSDL:**

Bạn cũng có thể xóa CSDL và kiểm tra xem nó có tồn tại hay không.



### ***Stored Procedure (SPROCs):***

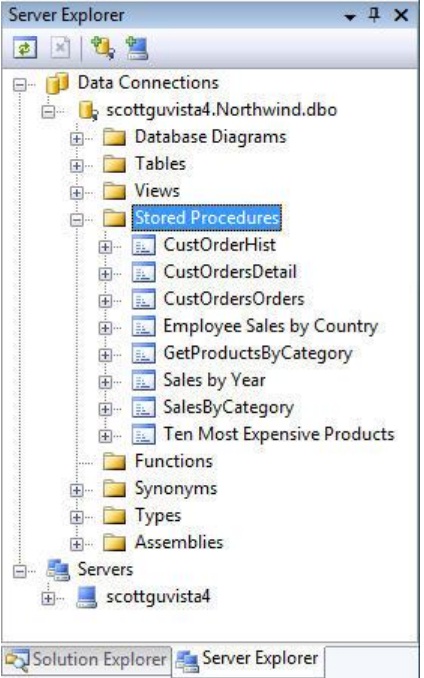
#### **Giới thiệu:**

LINQ to SQL đi cùng với .NET 3.5 rất mềm dẻo, và có thể được dùng để tạo các lớp mô hình dữ liệu, trong đó các đối tượng không phụ thuộc vào cấu trúc CSDL phía dưới, và có thể xử lý các phép kiểm tra logic cũng như xác thực tính hợp lệ của dữ liệu mà không phụ thuộc vào việc dữ liệu sẽ được lưu nạp dùng các câu SQL động hay thông qua các SPROCs.

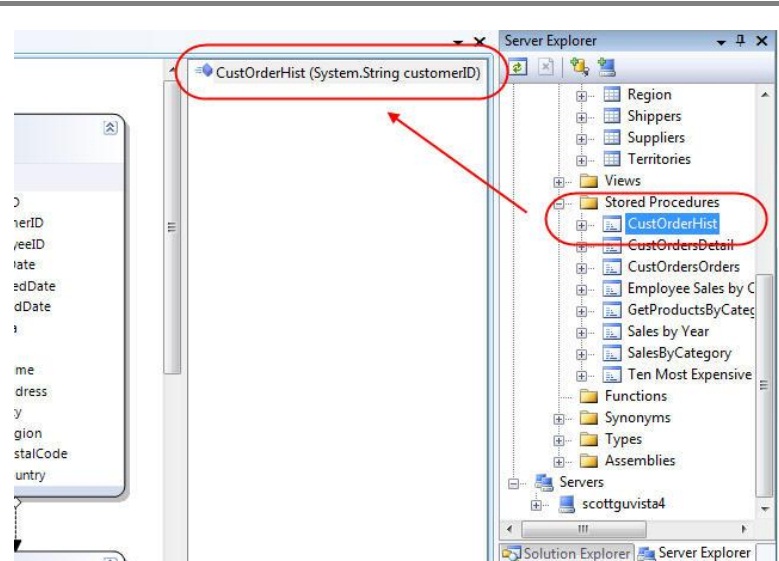
Ban cũng có thể dùng các SPROCs trong CSDL trong lớp DataContext, nó cung cấp một cách khác để lấy về các đối tượng bằng cách gọi thủ tục tương ứng. Khả năng này cho phép bạn dùng cả các câu SQL động và các SPROCs với một mô hình dữ liệu rõ ràng, mạnh mẽ cũng như cung cấp sự mềm dẻo khi làm việc với các dự án.

#### **Các bước ánh xạ và gọi SPROCs dùng LINQ to SQL:**

Để ánh xạ một SPROC vào lớp DataContext, trước tiên hãy mở cửa sổ Server Explorer trong VS 2008 và mở danh sách các SPROC trong CSDL:



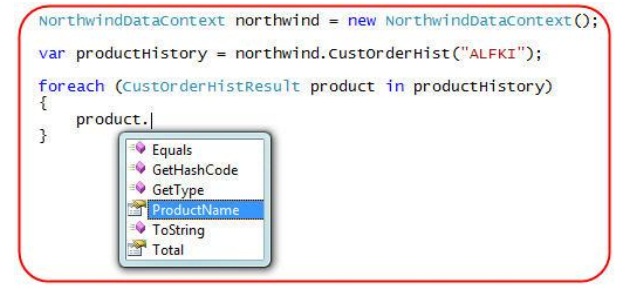
Bạn có thể mở bất cứ thủ rục SPROC ở trên và chỉnh sửa. Để ánh xạ vào SPROC ở trên vào DataContext, bạn có thể kéo/thả nó từ cửa sổ Server Explorer lên trên cửa sổ LINQ to SQL designer. Việc này sẽ làm tự động sinh ra một thủ tục trong lớp DataContext của LINQ to SQL như dưới đây:



Mặc nhiên tên của phương thức được tạo trong lớp DataContext sẽ chính là tên của SPROC, và kiểu trả về của phương thức sẽ là một kiểu được tạo tự động với cách đặt tên theo dạng “[SprocName]Result”. Ví dụ: SPROC ở trên sẽ trả vef một dãy các đối tượng có kiểu “CustOrderHistResult”. Chúng ta có thể đổi tên của phương thức nếu muốn bằng cách chọn nó rồi dùng Property Grid để đặt lại tên khác.

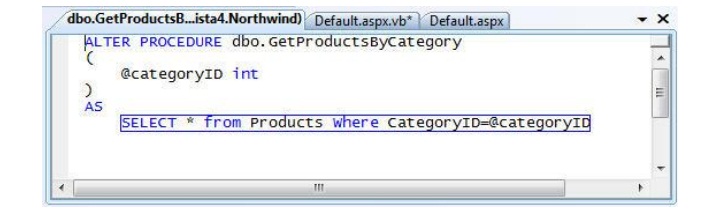
#### **Gọi SPROC mới được tạo:**

Khi đã hoàn thành các bước trên để ánh xạ một SPROC vào lớp DataContext của chúng ta, bạn có thể gọi nó một cách dễ dàng để lấy dữ liệu về. Tất cả những gì chúng ta cần làm là gọi phương thức mà chúng ta đã ánh xạ trong DataContext để lấy về một chuỗi các đối tượng về từ SPROC:

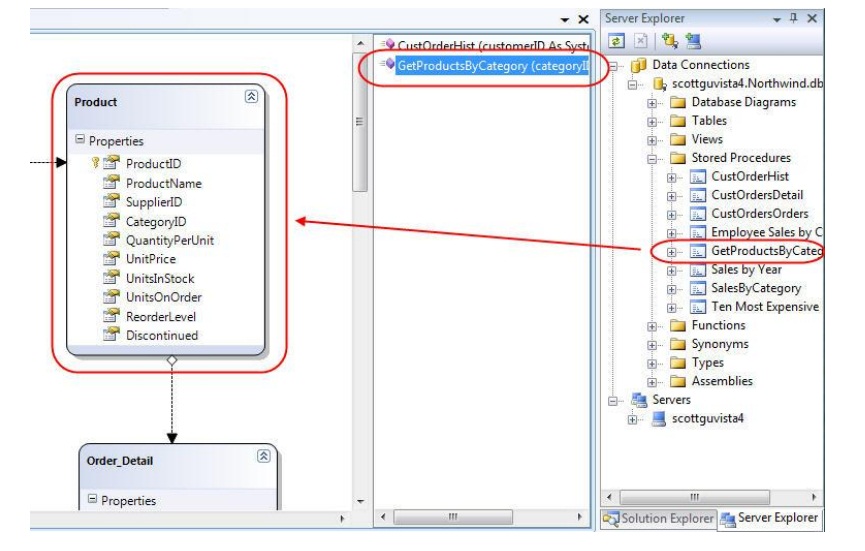


Chúng ta cũng có thể chọn cách gán kiểu trả về của thủ tục cho một lớp có sắn trong mô hình dữ liệu, ví dụ một lớp thực thể Product hay Order.

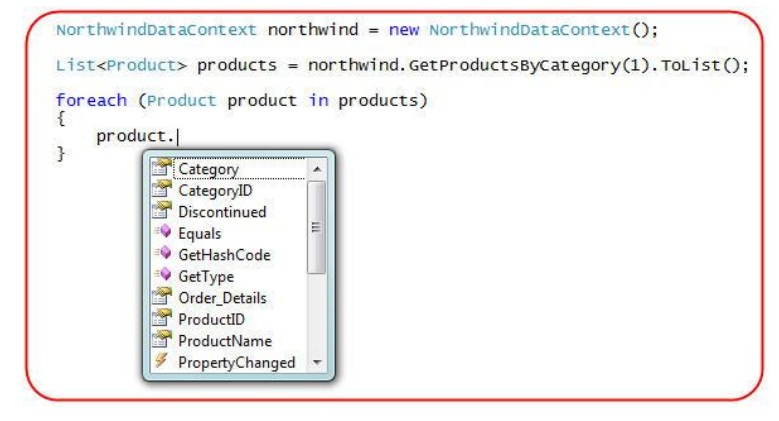
Ví dụ, cho là chúng ta có một thủ tục tênGetProductsByCategory trong CSDL trả về thông tin sản phẩm giống như sau:



Cũng như trước đây, ta có thể tạo một phương thức GetProductsByCategory ở bên trong lớp DataContext mà nó sẽ gọi thủ tục này bằng cách kéo nó vào cửa sổ LINQ to SQL designer. Thay vì thả nó vào một vị trí bất kỳ, chúng ta sẽ thả nó lên trên lớp Product mà ta đã tạo ra sẵn trên sửa sổ này:



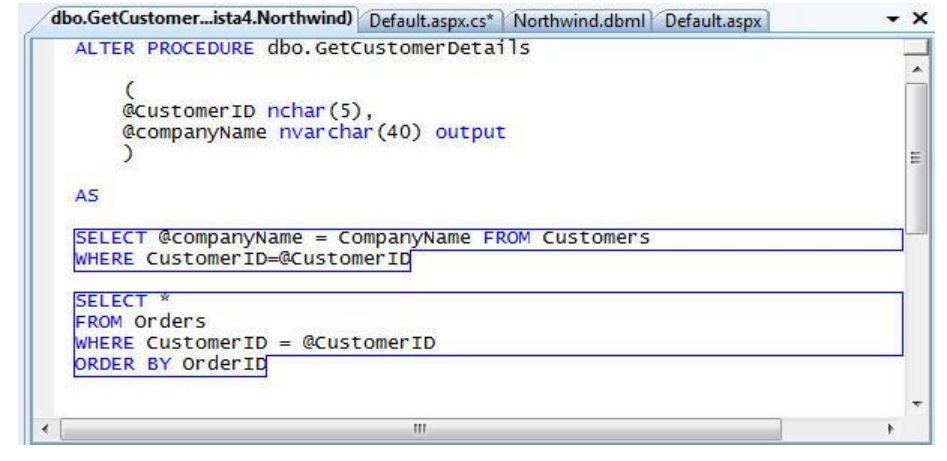
Việc kéo một SPROC và thả lên trên một lớp Product sẽ làm cho LINQ to SQL Designer tạo ra phương thức GetProductsByCategory trả về một danh sách các đối tượng có kiểu Product:



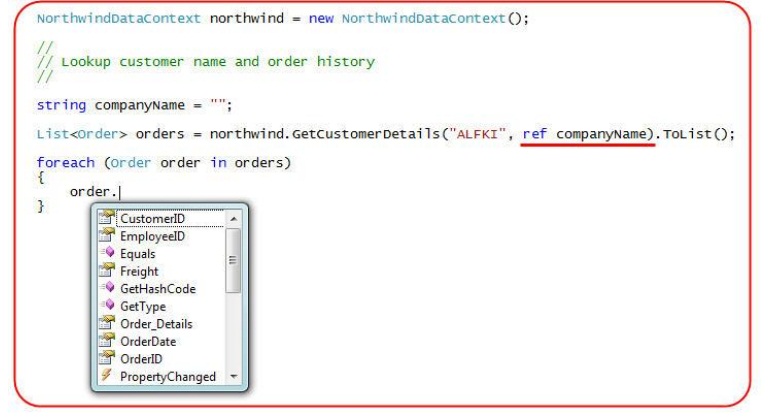
#### **Xử lí các tham số thủ tục dạng OUTPUT:**

LINQ to SQL ánh xạ các tham số dạng “OUTPUT” của các SPROC thành các tham biến (dùng từ khóa ref trong C# hoặc ByRef trong VB.NET), và với các tham trị, LINQ to SQL dùng các biến kiểu nullable (dùng ? trong C# hay <Nullable> trong VB.NET).

Ví dụ: thủ tục”GetCustomerDetails” sau sẽ nhận vào mộtCustomerID như tham số đầu vào, và trả về tên công ty như một tham số dạng OUTPUT và lịch sử giao dịch như kết quả truy vấn:



Nếu bạn kéo thủ tục trên để thả vào lớp Order trong LINQ to SQL designer, chúng ta có thể viết lệnh như sau để gọi nó:

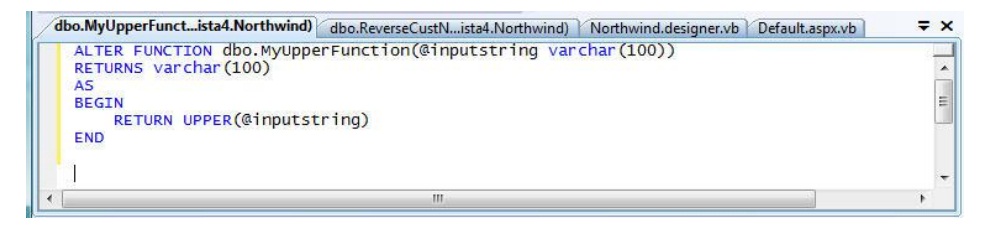


Chú ý thủ tục trên vừa trả về một tập các đối tượng Order, đồng thời trả về CompanyName thông qua một tham số output.

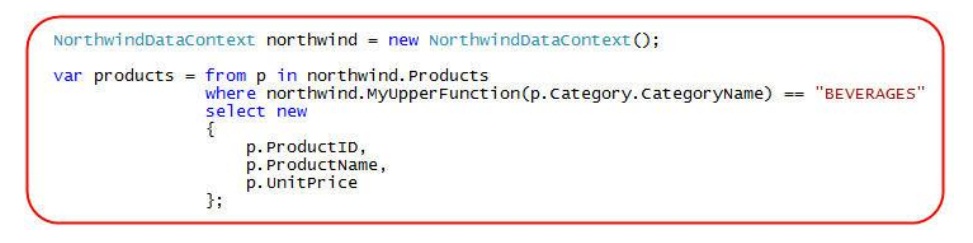
#### **Hỗ trợ các hàm do người dùng tự định nghĩa (UDF):**

Thêm vào việc hỗ trợ các các thủ tục, LINQ to SQL còn hỗ trợ các hàm trả về các giá trị vô hướng hoặc các bảng kết quả. Một khi đã được thêm vào lớp DataContext như một phương thức, bạn có thể dùng các hàm UDF này trong câu trong các câu lệnh LINQ.

Ví dụ, hãy xem các hàm UDF đơn giản có tên MyUpperFunction sau đây:



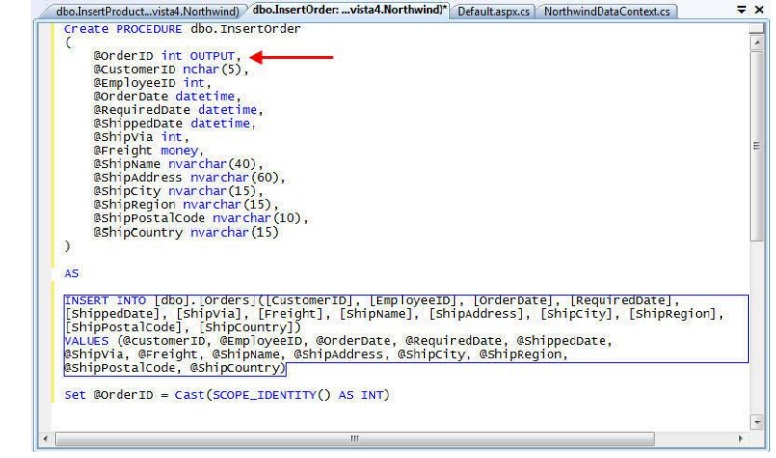
Chúng ta có thể kéo và thả nó từ cửa sổ Server Explorer lên cửa sổ LINQ to SQL Designer để thêm nó vào lớp DataContext như một phương thức. Chúng ta sau đó có thể dùng hàm UDF này ngay bên trong các biểu thức LINQ khi viết các câu truy vấn (giống như chúng ta đang dùng trong biểu thức Where như dưới đây):



#### **Cách sử dụng SPROC để thực hiện Insert/Update/Delete:**

##### ***Insert***

Đầu tiên, chúng ta đến cửa sổ "Server Explorer" , nhấp chuột phải vào Stored Procedures trong CSDL của chúng ta, và sau đó nhấn phải chuột và chọn "Add New Stored Procedure". Sau đó ta tạo thêm một thủ tục có tên "InsertOrder" có nhiệm vụ chèn thêm một bản ghi mới vào bảng Orders:

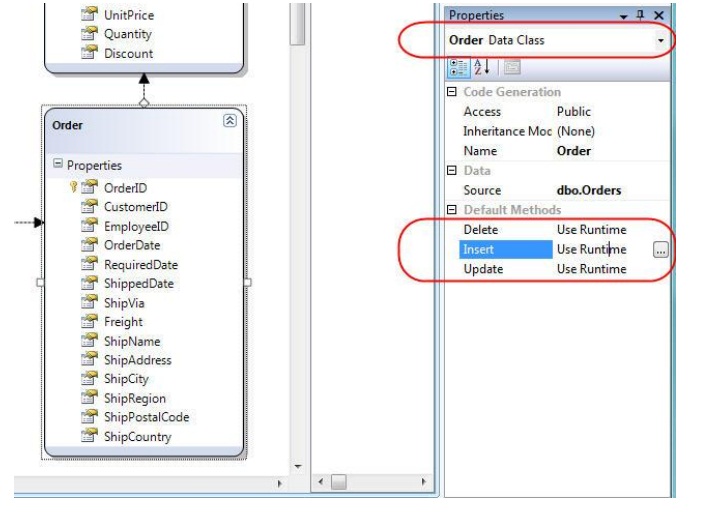


Hãy chú ý cách SPROC định nghĩa tham số OrderID như một tham số dạng OUTPUT. Đó là vì cột OrderID trong CSDL là cột tự tăng mỗi khi thêm một bản ghi mới vào. Người gọi sẽ truyền giá trị NULL khi gọi nó - và thủ tucjnafy sẽ trả về giá trị của OrderID mới được tạo ra (bằng cách gọi hàm SCOPE\_IDENTITY() ỏ cuối thủ tục).

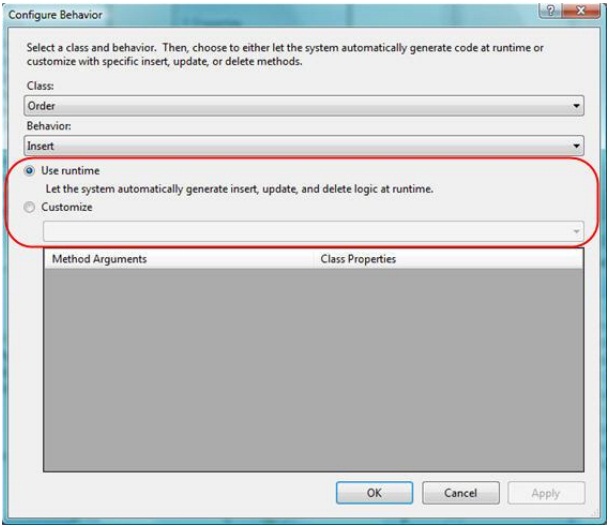
Sau khi tạo ra SPROC, chúng ta sẽ mở LINQ to SQL designer của lớp truy cập dữ liệu. Chúng ta có thể kéo thả các SPROC từ Server Explorer lên trên màn hình chính của

trình thiết kế. Chúng ta cũng sẽ làm điều tương tự với thủ tục InsertOrder vừa được tạo.

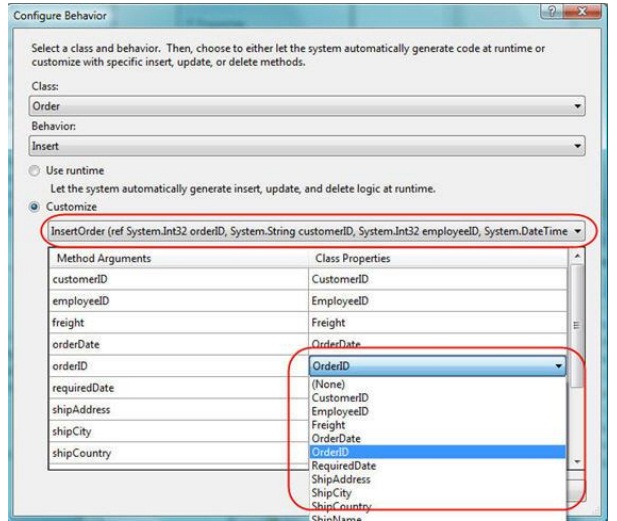
Bước cuối cùng là cấu hình lại để lớp truy cập dữ liệu dùng thủ tục SPROC khi chèn các đối tượng Order mới vào trong CSDL. Chúng ta có thể là điều này bằng cách chọn lớp Order trong cửa sổ LINQ to SQL designer, và sau đó chuyển đến bảng thuộc tính và nhấn nút 3 chấm (...) ở mục Insert để chọn thao tác tương ứng:



Khi nhấn nút này, cửa sổ sau sẽ hiện ra để có thể tùy biến hành vi Insert:



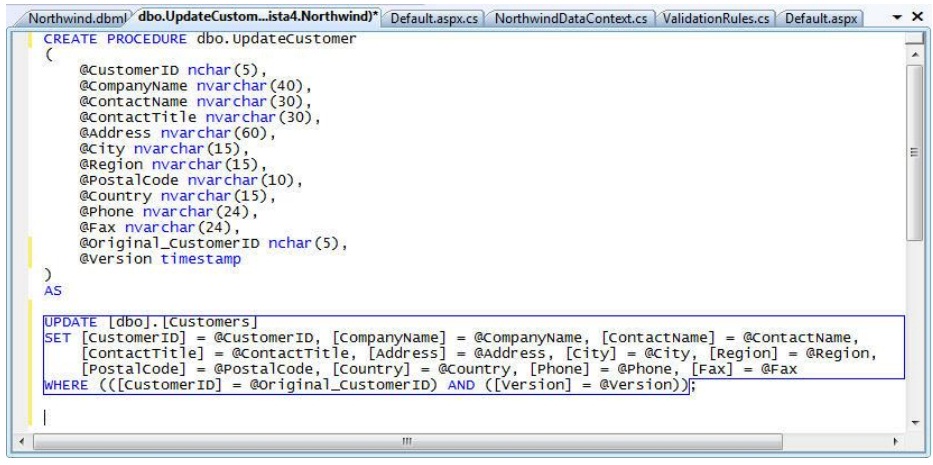
Ở trên, nếu bạn chọn chế độ mặc nhiên ("Use Runtime") thì LINQ to SQL sẽ tính toán và sinh ra câu lệnh SQL động để thực hiện các thao tác tương ứng. Chúng ta có thể thay đổi bằng cách nhấn chuột vào Customize và chọn thủ tục InsertOrder từ danh sách các SPROC:



LINQ to SQL sẽ hiển thị các tham số của thủ tục mà ta đã chọn, và cho phép ánh xạ các thuộc tính của lớp Order và các tham số của InsertOrder. Mặc nhiên, LINQ to cũng tự động xác định các tham số tương ứng theo tên, tuy nhiên bạn vẫn có thể sửa lại nếu muốn. Nhấn vào nút Ok là xong. Giờ đây bất cứ khi nào một đối tượng Order được thêm vào DataContext và phương thức SubmitChanges() được gọi, thủ tục InsertOrder sẽ được thực thi thay cho câu lệnh SQL động.

##### ***Update:***

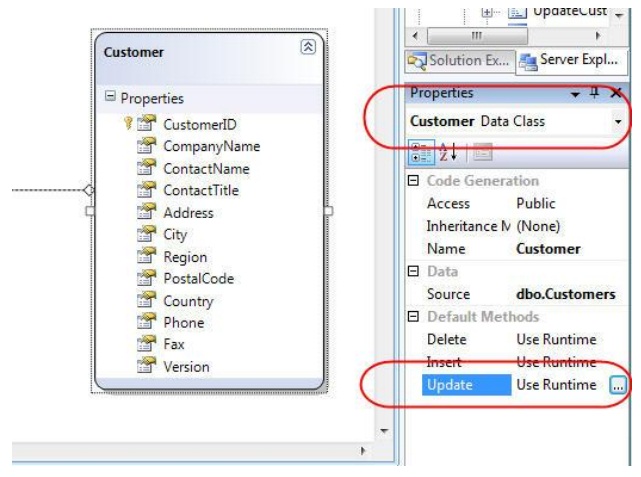
Chúng ta sẽ bắt đầu bằng cách tạo một SPROC tên "UpdateCustomer" như dưới đây:



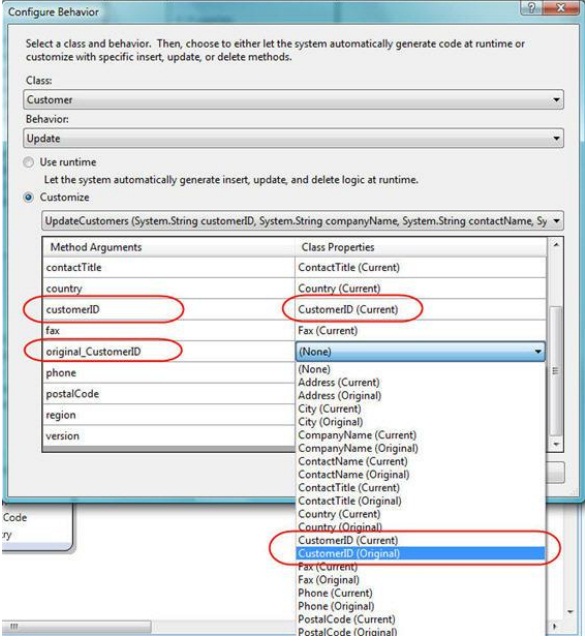
Chú ý ở trên, ngoài việc truyền giá trị cho tham số CustomerID, ví dụ truyền một tham số khác có tên @Original\_CustomerID. Cột CustomerID trong bảng Customers không phải là một cột tự tăng, và nó có thể được chỉnh sửa như một phần của thao tác cập nhật. Do vậy chúng ta sẽ phải truyền cả giá trị của CustomerID cũ và CustomerID mới để có thể cập nhật.

Bạn sẽ thấy ở trên ví dụ đã truyền một tham số có tên @Version (có kiểu timestamp) vào cho SPROC. Đây là một cột đã thêm vào bảng Customers để có thể xử lý việc tranh chấp khi các thao tác cập nhật được diễn ra đồng thời (optimistic concurrency). LINQ to SQL hỗ trợ đầy đủ optimistic concurrency, và cho phép bạn có thể chọn dùng version timestamp hay bằng cách cung cấp cả giá trị cũ/mới cho SPROC để có thể xác định được các thay đổi được tạo ra bới người khác kể từ lần cuối bạn đọc dữ liệu.

Một khi đã tạo xong SPROC, bạn có thể kéo/thả nó vào cửa sổ LINQ to SQL designer để thêm nó như một phương thức trong lớp DataContext. Chúng ta có thể chọn lớp Customer trong cửa sổ thiết kế và nhấn vào nút ... ở mục Update để dùng SPROC vừa tạo trong việc cập nhật lại dữ liệu trong bảng Customer:



Chúng ta sẽ chọn ô "Customize" và chọn để dùng UpdateCustomer:



Khi ánh xạ các thuộc tính của đối tượng Customer vào các tham số của SPROC, bạn sẽ được nhắc rằng bạn đang muốn gán các giá trị mới(Current) hay các giá trị gốc (Original) - là các giá trị mà bạn lấy về lần đầu từ CSDL. Ví dụ, bạn sẽ cần gán giá trị thuộc tính Customer.CustomerID "mới" vào cho tham số @CustomerID của SPROC, và Customer.CustomerID "gốc" vào cho @original\_customerID.

Khi nhấn "Ok" trên cửa sổ này, bạn đã hoàn thành việc ánh xạ các tham số vào các thuộc tính. Từ giờ trở đi, mỗi khi cập nhật lại giá trị cho đối tượng Customer và gọi SubmitChanges(), thủ tục UpdateCustomer sẽ được gọi thay cho câu lệnh SQL động.

## ***LINQ to XML:***

### ***Giới thiệu:***

XML là một ngôn ngữ được ứng dụng rất nhiều trong lập trình, xử lý dữ liệu cũng như truyền tải thông tin. Trước đây việc thao tác và tạo ra các tài liệu XML thông qua DOM XML API trong .Net rất dài dòng và rắc rối. Vì thế để khắc phục điều này, Microsoft đã cho ra đời LINQ cùng với LINQ to XML API.

LINQ to XML cung cấp một giao diện lập trình XML. LINQ to XML sử dụng những ngôn ngữ mới nhất của .NET Language Framework vàđược nâng cấp, thiết kế lại với giao diện lập trình XML Document Object Model (DOM). XML đã được sử dụng rộng rãi để định dạng dữ liệu trong một loạt các ngữ cảnh (các trang web, trong các tập tin cấu hình, trong các tập tin Microsoft Office Word, và trong cơ sở dữ liệu).

LINQ to XML có cấu trúc truy vấn tương tự SQL. Nhà phát triển trung bình có thể viết các truy vấn ngắn gọn, mạnh mẽ, viết mã ít hơn nhưng có ý nghĩa nhiều hơn. Họ có thể sử dụng các biểu thức truy vấn từ nhiều dữ liệu các lĩnh vực tại một thời điểm.

LINQ to XML cũng giống như Document Object Model (DOM) ở chỗ nó chuyển các tài liệu XML vào bộ nhớ. Bạn có thể truy vấn và sửa đổi các tài liệu, và sau khi bạn chỉnh sửa nó, bạn có thể lưu nó vào một tập tin hoặc xuất nó ra.

Tuy nhiên, LINQ to XML khác DOM: Nó cung cấp mô hình đối tượng mới đơn giản hơn và dễ thao tác hơn để làm việc ,và đó là tận dụng các cải tiến ngôn ngữ trong Visual C # 2008. Khả năng sử dụng kết quả truy vấn là tham số cho đối tượng XElement và XAttribute chophép một phương pháp mạnh mẽ để tạo ra cây XML. Phương pháp này, được gọi là functional construction, cho phép các nhà phát triển để dễ dàng chuyển đổi cây XML từ một hình dạng này sang hình dạng khác.

Sử dụng LINQ to XML bạn có thể:

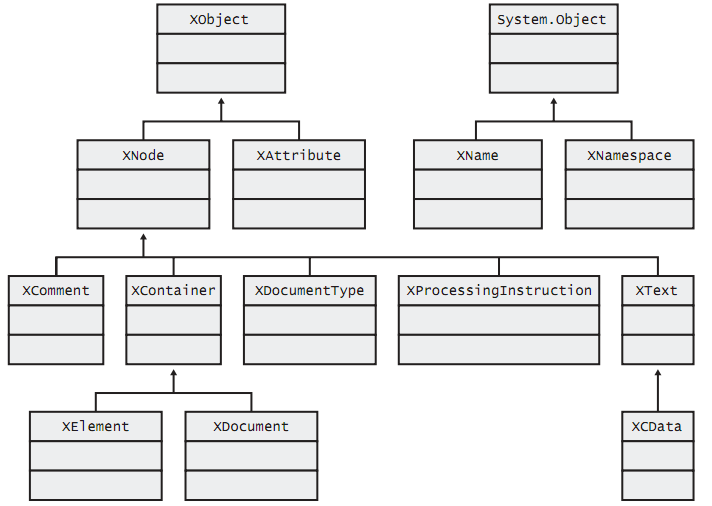
* Load XML từ nhiều file hoặc luồng.
* Xuất XML ra file hoặc luồng.
* Truy vấn cây XML bằng những truy vấn LINQ.
* Thao tác cây XML trong bộ nhớ.
* Biến đổi cây XML từ dạng này sang dạng khác.

### ***LINQ to XML API***

LINQ to XML API độc lập với các truy vấn LINQ to XML, nó cho phép nhà phát triển xây dựng và quản lý nội dung XML, bạn có thể sử dụng nó độc lập hoặc kết hợp với các truy vấn của LINQ.

Mục tiêu của LINQ to XML API là cung cấp một cách tiếp cận hướng đối tượng cho việc xây dựng, quản lý XML, tránhhoặcgiải quyết các vấn đềphổbiếnliên quan đếnthao tácXMLthôngquaW3CDOM.

API gồm một tập hợp các lớp có tên bắt đầu là X, mô hình phân cấp đối tượng chính của các lớp X\* trong API:



Tổng quan các lớp X\*:

|  |  |
| --- | --- |
| **Lớp** | **Mô tả** |
| XName | Đại diện tên cho một phần tử hoặc thuộc tính của XML. |
| XNamespace | Đại diện cho XML namespace, lớp này không thể kế thừa. |
| XObject | Đại diện cho một nút hoặc một thuộc tính trong cây XML. |
| XAttribute | Đại diệncho một thuộc tínhcủa XML. |
| XNode | Đại diện cáckhái niệm trừu tượngcủa một nút(thành phần, mô tả, loại tài liệu, hướng xử lý, hoặc nút văn bản) trong câyXML. |
| XText | Đại diện chomột nút văn bản. |
| XComment | Đại diện cho XML comment. |
| XContainer | Đại diện cho một nút có thể chứa các nút khác. |
| XElement | Đại diện cho một phần tử của XML. |
| XDocument | Đại diện cho một tài liệu XML. |

#### **Xnamespace:**

Đại diện cho một namespace trong tài liệu XML. Bạn có thể tạo thể hiện của lớp này từ một đối tượng string có dạng URI.

XNamespace ns= “namespace”;

Để gán namespace cho một phần tử, thông thường ta sẽ tạo một đối tượng XNamespace và dùng toán tử cộng để tạo ra một tên đầy đủ cho phần tử:

XNamespace ns= “namespace”;

XElement x= new XElement(ns+ “FirstElement”);

#### **Xname:**

Mỗi phần tử XML được tạo ra đều phải có tên, và tên này được đại diện bằng một đối tượng XName. Về cơ bản, XName là một định danh bao gồm tên cục bộ và tên namespace của phần tử (bao gồm XNamespace). Giống như XNamespace, XName có thể tự động được tạo ra từ một đối tượng string bằng cách overload operator:

public static implicit operator XName(string expandedName);

Một đối tượng XName sẽ được tạo ra trong mỗi dòng lệnh sau:

XName name = “Test”;

XElement root = new XElement(“Test”);

Ngoài ra bạn có thể tạo một XName bao gồm tên cục bộ và namespace bằng cách đặt tên namespace trong cặp {} và theo sau đó là tên cục bộ:

XName name=”{namespace}Test”;

Khi đó property LocalName sẽ là “Test” và Namespace sẽ là “namespace”.

#### **Xobject:**

Được thừa kế từ System.Object  và IXmlLineInfo (System.Xml), XObject là một lớp nằm ở đỉnh của cây phân cấp. Lớp này cung cấp các đặc tính cần thiết giúp ta truy xuất được những phần tử liên quan và một số thông tin cần thiết của phần tử trong tài liệu XML.

* ***Document***: đối tượng XDocument chứa toàn bộ cấu trúc và nội dung của tài liệu XML.
* ***NoteType***: lưu giá trị kiểu enum XmlNodeType xác định kiểu của phần tử.
* ***Parent***: tham chiếu đến phần tử cha có kiểu XElement.

Ngoài ra lớp này chứa hai event là ***Changed*** và ***Changing***được kích hoạt mỗi khi phần tử bị thay đổi. Hai event này có kiểu EventHandler<XObjectChangeEventArgs>. Lớp XObjectChangeEventArgs chứa property có kiểu XObjectChange, đây là một enum chỉ ra kiểu thay đổi tác động đến phần tử, bao gồm 4 giá trị: Add, Remove, Name (đổi tên), Value (đổi giá trị).

Thêm vào đó, XObject cho phép bạn thêm các **annotation** để bổ sung thông tin cho đối tượng. Các thông tin annotation chỉ là tạm thời, không phải là một phần của tài liệu XML và sẽ không được lưu xuống file.

#### **Xattribute:**

Một phần tử XML có thể chứa nhiều attribute, mỗi attribute này được đại diện bởi một đối tượng XAttribute. Bạn có thể truy xuất đến các attribute trước và sau bằng cách dùng hai property PreviousAttribute và NextAttribute. Trong trường hợp attribute là một khai báo namespace, property IsNamespaceDeclaration sẽ trả về true.

#### **Xnode:**

Đại diện cho các phần tử (element, comment, document type, processing instruction, text node). XNode cho phép bạn truy xuất đến các phần tử trước và sau với PreviousNode và NextNode.

Một đặc điểm quan trọng của XNode là cung cấp các phương thức dùng để thêm/xóa/sửa các phần tử trong tài liệu XML. Các phần tử được thêm vào sẽ có cùng level với nhau và có cùng phần tử cha.

#### **XText và Xcomment:**

Các phần tử này chỉ chứa property Value lưu giá trị của văn bản, chú thích. Một dạng wrapper của string.

#### **Xcontainer:**

Dùng để chứa các phần tử khác. Lớp này cung cấp hai property là FirstNode và LastNode cho phép truy xuất đến phần tử đầu và cuối. Ngoài ra bổ sung thêm cho lớp XNode các phương thức để thêm phần tử con vào.

#### **Xelement:**

Đại diện cho một phần tử, lớp này tương tự như XNode và XContainer, bổ sung thêm một số thuộc tính và phương thức ví dụ như Attributes() để lấy về các attribute của phần tử. Đặc trưng chính của lớp này là các phương thức Parse, Load, Save nội dung của văn bản hoặc tập tin thành mộtphần tử.

### ***Những thao tác truy vấn cơ bản trên cây XML:***

#### **Tìm phần tử trong cây XML:**

Tìm phần tử Address có thuộc tính Type có giá trị là "Billing".

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);

#### **Lọc phần tử trong cây XML:**

Lọc những phần tử có phần tử con <Type > có Value="Yes".

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);

#### **Sắp xếp các phần tử trong cây XML:**

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);

#### **Kết hai cây XML:**

Kết những phần tử Customer với những phần tử Order , và tạo ra tài liệu XML mới gồm phần tử CompanyName bên trong order.

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)

{

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);

}

#### **Nhóm các phần tử trong cùng một cây XML:**

Nhóm dữ liệu theo loại, sau đó tạo ra một tập tin XML mới, trong đó phân cấp XML theo nhóm.

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);

### ***Những thao tác biến đổi trên cây XML:***

#### **Thêm phần tử, thuộc tính và nút vào một cây XML:**

Thêm nút vào đầu hoặc cuối cây XML ta dùng phương thức .Add và .AddFirst.

XElement xmlTree = new XElement("Root",

new XElement("Child1", 1),

new XElement("Child2", 2),

new XElement("Child3", 3),

new XElement("Child4", 4),

new XElement("Child5", 5)

);

xmlTree.Add(new XElement("NewChild", "new content"));

Console.WriteLine(xmlTree);

#### **Thay đổi phần tử, thuộc tính và nút của một cây XML:**

Sử dụng phương thức **XAttribute.SetValue**thay đổi giá trị thuộc tính "Att" từ"root" thành"new content".

XElement root = new XElement("Root",

new XAttribute("Att", "root"), (“Root”)

);

Console.WriteLine(root);

Console.WriteLine(“---------------”)

XAttribute att = root.Attribute("Att");

att.SetValue("new content");

root.SetValue("new content");

Console.WriteLine(root);

Sử dụng phương thức **XNode.ReplaceWith**thay đổi nút có tên "Child3"có nội dung "child3 content" thành "NewChild" nội dung "new content".

XElement xmlTree = new XElement("Root",

new XElement("Child1", "child1 content"),

new XElement("Child2", "child2 content"),

new XElement("Child3", "child3 content"),

new XElement("Child4", "child4 content"),

new XElement("Child5", "child5 content")

);

XElement child3 = xmlTree.Element("Child3");

child3.ReplaceWith(

new XElement("NewChild", "new content")

);

Console.WriteLine(xmlTree);

Thiết lập giá trị của lớp con bằng hàm **XElement.SetElementValue**.

XElement root = new XElement("Root");

root.SetElementValue("Ele1", 1);

root.SetElementValue("Ele2", 2);

root.SetElementValue("Ele3", 3);

Console.WriteLine(root);

root.SetElementValue("Ele2", 22);

Console.WriteLine(root);

root.SetElementValue("Ele3", null);

Console.WriteLine(root);

#### **Xóa phần tử, thuộc tính và nút từ một cây XML:**

Dùng phương thức **XElement.RemoveAll**xóa những phần tử con và thuộc tính của một cây XML.

XElement root = new XElement("Root",

new XAttribute("Att1", 1),

new XAttribute("Att2", 2),

new XAttribute("Att3", 3),

new XElement("Child1", 1),

new XElement("Child2", 2),

new XElement("Child3", 3)

);

root.RemoveAll(); // removes children elements and attributes of root

Console.WriteLine(root);

Dùng phương thức **XElement.RemoveAttributes** xóa toàn bộ thuộc tính của một cây XML.

XElement root = new XElement("Root",

new XAttribute("Att1", 1),

new XAttribute("Att2", 2),

new XAttribute("Att3", 3),

new XElement("Child1", 1),

new XElement("Child2", 2),

new XElement("Child3", 3)

);

root.RemoveAttributes();

Console.WriteLine(root);

Dùng phương thức **XNode.Remove** xóa một nút của cây XML.

XElement xmlTree = new XElement("Root",

new XElement("Child1", "child1 content"),

new XElement("Child2", "child2 content"),

new XElement("Child3", "child3 content"),

new XElement("Child4", "child4 content"),

new XElement("Child5", "child5 content")

);

XElement child3 = xmlTree.Element("Child3");

child3.Remove();

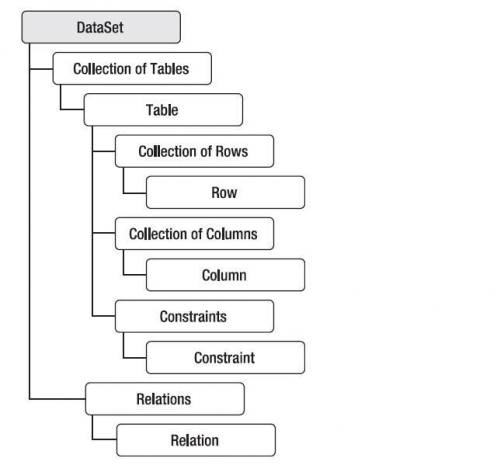
Console.WriteLine(xmlTree);

## ***LINQ to DataSet:***

### ***Giới thiệu:***

DataSet trong ADO.NET là một bước phát triển lớn trong việc phát triển ứng dụng cơ sở dữ liệu đa hệ. Khi lấy và chỉnh sửa dữ liệu, duy trì liên tục kết nối tới Data Source trong khi chờ người dùng yêu cầu thì rõ ràng là tốn tài nguyên máy rất nhiều.

DataSet rất có ích, vì nó cho phép lưu trữ dữ liệu và chỉnh sửa tại “local cache”,hay gọi là “offline mode”, nghĩa là nó thể xem xét và xử lý thông tin trong khi ngắt kết nối. Sau khi chỉnh sửa và xem xong thì tạo một kết nối và cập nhật dữ liệu từ local vào Data Source. Dữ liệu trong DataSet được lưu trữ dưới dạng một tập hợp các bảng và bạn cần phải xử lý thông qua các lớp DataTable (DataRow và DataColumn).



*Kiến trúc DataSet*

**LINQ to DataSet** cho phép người lập trình sử dụng DataSet như một nguồn dữ liệu bình thường bằng các cú pháp truy vấn căn bản của LINQ.

**LINQ to DataSet** hỗ trợ truy vấn dữ liệu thông qua ADO.NET với 2 đối tượng là DataSet và DataTable.

### ***Sử dụng LINQ nạp DataSet:***

DataSet có thể được nạp bằng cách truy vấn một cơ sở dữ liệu quan hệ, ta có thể làm việc này thông qua một DataAdapter, như ví dụ:

DataSet ds = new DataSet("CustomerOrders");

SqlDataAdapter da = new SqlDataAdapter( QueryOrders, ConnectionString );

da.SelectCommand.Parameters.AddWithValue( "@CustomerID", "QUICK" );

da.TableMappings.Add( "Table", "Orders" );

da.TableMappings.Add( "Table1", "OrderDetails" );

da.Fill( ds );

const string ConnectionString = "Database=Northwind;Trusted\_Connection=yes";

const string QueryOrders = @"

SET @CustomerID = 'QUICK'

SELECT OrderID, OrderDate, Freight, ShipName,

ShipAddress, ShipCity, ShipCountry

FROM Orders

WHERE CustomerID = @CustomerID

SELECT od.OrderID, od.UnitPrice, od.Quantity, od.Discount,

p.[ProductName]

FROM [Order Details] od

INNER JOIN Orders o ON o.[OrderID] = od.[OrderID]

LEFT JOIN Products p ON p.[ProductID] = od.[ProductID]

WHERE o.CustomerID = @CustomerID";

Các mã trước đó kết hợp hai DataTable vào một DataSet, tương ứng với các đơn đặt hàng của một khách hàng cụ thể.

### ***Sử dụng LINQ truy vấn DataSet:***

DataSet có thể truy vấn được bằng LINQ, nó giống với những danh sách IEnumerable<T> khác.

Danh sách này được tạo bởi các đối tượng DataRow, do đó, bạn phải truy vấn thông tin từng đối tượng DataRow để có được một trườnggiá trị. Sự sắp xếp này cho phép ta truy vấn đến bất kỳ đối tượng nào của DataRow thay vì sử dụng một biểu thức truy vấn trên một DataTable.

Ta có thể sử dụng phương thức truy xuất Field<T> thay vì truy xuất trực tiếp vào DataRow, chẳng hạn như o["OrderDate"], ví dụ:

DataSet ds = LoadDataSetUsingDataAdapter();

DataTable orders = ds.Tables["Orders"];

DataTable orderDetails = ds.Tables["OrderDetails"];

var query = from o in orders.AsEnumerable()

where o.Field<DateTime>( "OrderDate" ).Year >= 1998

orderby o.Field<DateTime>( "OrderDate" ) descending

select o;

AsEnumerable và Field<T> là hai phương thức tùy chình mở rộng cho DataTable và DataRow. Chúng được xác định tương ứng trong System.Data.DataTableExtensions và System.Data.DataRowExtensions.

Khi bạn đã có một vài DataTable trong DataRow, bạn sẽ cần một vài cách kết hợp chúng với nhau bằng cú pháp john.

Đoạn truy vấn trong ví dụ sau tính toán tổng số tiền để cho mỗi đơn đặt hàng từ năm 1998 đến nay.

DataSet ds = LoadDataSetUsingDataAdapter();

DataTable orders = ds.Tables["Orders"];

DataTable orderDetails = ds.Tables["OrderDetails"];

var query =

from o in orders.AsEnumerable()

join od in orderDetails.AsEnumerable()

on o.Field<int>( "OrderID" ) equals od.Field<int>("OrderID")

into orderLines

where o.Field<DateTime>( "OrderDate" ).Year >= 1998

orderby o.Field<DateTime>( "OrderDate" ) descending

select new { OrderID = o.Field<int>( "OrderID" ),

OrderDate = o.Field<DateTime>( "OrderDate" ),

Amount = orderLines.Sum(od =>

od.Field<decimal>("UnitPrice")\*od.Field<short>("Quantity"))};

Trong ví dụ trước, bạn xác định cácmối quan hệ giữa các Orders và các OrderDetails thông qua cú pháp john. Nếu DataSet có chứa thông tin về mối quan hệ hiện có giữa các thực thể,một truy vấn LINQ có thể tận dụng lợi thế này.

Một ví dụ sử dụng GetChildRows để tạo kết nối cho các OrderDetails thay vì dùng kết hợp hai bảng.

DataSet ds = LoadDataSetUsingDataAdapter();

DataTable orders = ds.Tables["Orders"];

DataTable orderDetails = ds.Tables["OrderDetails"];

ds.Relations.Add( "OrderDetails", orders.Columns["OrderID"],

orderDetails.Columns["OrderID"]);

var query = from o in orders.AsEnumerable()

where o.Field<DateTime>( "OrderDate" ).Year >= 1998

orderby o.Field<DateTime>( "OrderDate" ) descending

select new { OrderID = o.Field<int>( "OrderID" ),

OrderDate = o.Field<DateTime>( "OrderDate" ),

Amount = o.GetChildRows( "OrderDetails" ).Sum(

od => od.Field<decimal>( "UnitPrice" )

\* od.Field<short>( "Quantity" ) ) };

### ***Sử dụng LINQ để Truy vấn một DataSet nhập liệu:***

Một DataSetnhập taycóthểđượctruy vấnvới một cú phápđơn giảnvìnó không cần thiết sửdụng đếnphương thức AsEnumerable và Field<T>.Nếu bạn tạo ra một DataSet nhập tay bằng Visual Studio, lớp DataTable đó sẽ kế thừa lớp TypedTableBase<T> trong interface IEnumerable<T>. Cho nên khi truy vấn nó ta không cần sử dụng phương thức AsEnumerable và Field<T>.

Ví dụ truy vấn một DataSet nhập tay:

var query = from o in ds.Orders

where o.OrderDate.Year >= 1998

orderby o.OrderDate descending

select new { o.OrderID, o.OrderDate,

Amount = o.GetOrder\_DetailsRows().Sum(

od => od.UnitPrice \* od.Quantity ) };

Ta thấy cú pháp đơn giản hơn nhiều, nhưng phải sử dụng một DataSet nhập tay.

### ***Truy cập dữ liệu DataSet không định kiểu:***

Để truy xuất dữ liệu một DataSet không định kiểu ta phải sử dụng bộ truy xuất Field<T>và SetField<T> để lấy và gán giá trị. Những bộ truy xuất này rất quan trọng, vì trong DataSet sẽ có những giá trị rỗng, khi sử dụng bộ truy xuất thì chuyển chúng thành phương thức IsNull và trở thành giá trị đúng. Bạn nên kiểm tra điều kiện này để tránh những lỗi tiềm tàng.

Việc sử dụng cácbộ truy xuấtđược cho phéptrongbất kỳ mộtDataTable, DataRow, thậm chíbên ngoài mộtbiểu thức truy vấn, như bạn có thểthấytrong ví dụ sau:

foreach( DataRow r in orderDetails.Rows ) {

if (r.Field<decimal>( "UnitPrice" ) < 10 ){

r.SetField<decimal>( "UnitPrice", 10 );

}

}

# ***KẾT LUẬN:***

LINQ là một trong những tính năng cốt lõi của .NET Framework, được hỗ trợ đầy đủ trong các ngôn ngữ C# và VB. LINQ hiện có một dải rộng các ứng dụng tiềm năng. Nói một cách đơn giản, nó cung cấp một mô hình mang tính khai báo cho việc nhận và xử lý dữ liệu cho phép ta sử dụng cùng một cú pháp với nhiều nguồn dữ liệu khác nhau.

Nguyên lý thống nhất nằm dưới mọi ứng dụng LINQ là nó nhấn mạnh đến lập trình khai báo so với lập trình chức năng – nói một cách khác, đoạn code của bạn chỉ phát biểu kết quả mà nó muốn thay vì một chuỗi các bước để đạt được kết quả đó. Ở mức lý tưởng, điều này sẽ thúc đẩy các lập trình viên tập trung vào business logic và cho LINQ nhiều tự do hơn để tự động hóa các tác vụ cấp thấp cũng như tối ưu cách thực hiện chúng.

# ***TÀI LIỆU THAM KHẢO***

Tiếng Việt: [1] LINQ to SQL Tutorial (2009), Đào Hải Nam.

Tiếng Anh: [1] Introducing Microsoft LINQ (2007), Paolo Pialorsi Marco Russo.

[2]