Tìm hiểu về LINQ

Mục lục

[I. Giới thiệu về LINQ: 4](#_Toc8945762)

[II. Giới thiệu vài nét về quá trình hình thành và phát triển của LINQ: 4](#_Toc8945763)

[II.1. Những ý tưởng ban đầu với ODBC, DAO, RDO và ODBCDirect: 4](#_Toc8945764)

[II.2. Thành phần hóa quá trình truy cập dữ liệu (OLE DB): 5](#_Toc8945765)

[II.3. .NET Framework và C# ra đời, thống nhất các components và API vào ADO: 5](#_Toc8945766)

[II.4. Lập trình cho mô hình quan niệm (ADO.NET Entities và LINQ): 6](#_Toc8945767)

[II.5. Quá trình phát triển của LINQ: 7](#_Toc8945768)

[III. Những khái niệm cấu thành nên LINQ: 8](#_Toc8945769)

[III.1. Phương thức mở rộng - Extension method: 8](#_Toc8945770)

[III.2. Gọi lại(Callback) và Ủy thác(Delegate) (Func, Predicate): 8](#_Toc8945771)

[III.3. Biểu thức Lambda - Lambda Expression: 10](#_Toc8945772)

[III.4. Mẫu - Generic: 13](#_Toc8945773)

[III.5. IEnumerable, IQueryable và Yield: 14](#_Toc8945774)

[IV. Cấu trúc và đặc tính của LINQ: 17](#_Toc8945775)

[IV.1. Cấu trúc cơ bản của LINQ và biểu thức LINQ: 17](#_Toc8945776)

[IV.1.1. Ba phần của một biểu thức LINQ: 19](#_Toc8945777)

[IV.1.2. Các dữ liệu nguồn: 21](#_Toc8945778)

[IV.1.3. Truy vấn và thực thi truy vấn: 21](#_Toc8945779)

[IV.1.4. Các biến IEnumerable có trong các câu truy vấn LINQ: 22](#_Toc8945780)

[IV.2. Các hoạt động truy vấn cơ bản: 22](#_Toc8945781)

[IV.2.1. Truy xuất dữ liệu nguồn: 22](#_Toc8945782)

[IV.2.2. Lọc (Filtering): 22](#_Toc8945783)

[IV.2.3. Sắp xếp (Ordering): 23](#_Toc8945784)

[IV.2.4. Gom nhóm (Grouping): 23](#_Toc8945785)

[IV.2.5. Kết (Joining): 24](#_Toc8945786)

[IV.2.6. Lựa chọn (Selection/Projection): 25](#_Toc8945787)

[IV.3. Chuyển đổi dữ liệu với LINQ: 26](#_Toc8945788)

[IV.4. Cú pháp truy vấn (Query Syntax) và cú pháp phương thức (Method Syntax): 26](#_Toc8945789)

[IV.5. LINQ và Kiểu mẫu (Generic Type): 27](#_Toc8945790)

[IV.6. Mô hình đối tượng quan hệ (Object Relational Mapping) và thiết kế đối tượng quan hệ (Object Relational Designer): 27](#_Toc8945791)

[IV.6.1. Mô hình đối tượng quan hệ (ORM): 27](#_Toc8945792)

[IV.6.2. Thiết kế đối tượng quan hệ (O/R desgner): 29](#_Toc8945793)

[IV.6.3. Các đặc tính của Thiết Kế Đối Tượng Quan Hệ: 32](#_Toc8945794)

[V. LINQ to SQL: 32](#_Toc8945795)

[V.1. Các Khái niệm trong LINQ to SQL: 33](#_Toc8945796)

[V.1.1. Mô hình hóa CSDL dùng LINQ to SQL(dùng thiết kế đối tượng quan hệ): 33](#_Toc8945797)

[V.1.2. Lớp DataContext 34](#_Toc8945798)

[V.1.3. Các câu lệnh làm việc trong LINQ to SQL: 35](#_Toc8945799)

[V.2. Xây dựng chương trình với LINQ to SQL: 39](#_Toc8945800)

[V.2.1. Định nghĩa các lớp mô hình dữ liệu: 39](#_Toc8945801)

[V.2.2. Truy vấn Cơ sở dữ liệu: 47](#_Toc8945802)

[V.2.3. Cập nhật Cơ sở dữ liệu: 55](#_Toc8945803)

[V.2.4. Lấy dữ liệu dùng Stored Procedure: 66](#_Toc8945804)

[V.2.5. Cập nhật dữ liệu dùng Stored Procedure: 73](#_Toc8945805)

[V.2.6. Thực thi các biểu thức SQL tùy biến: 84](#_Toc8945806)

[V.3. Tổng kết: 88](#_Toc8945807)

[VI. LINQ to Object: 88](#_Toc8945808)

[VI.1. Các khái niệm trong LINQ to Object: 89](#_Toc8945809)

[VI.1.1. Object: 89](#_Toc8945810)

[VI.2. Xây dựng chương trình với LINQ to Object: 89](#_Toc8945811)

[VI.2.1. LINQ to String: 89](#_Toc8945812)

[VI.2.2. : 89](#_Toc8945813)

[VI.3. Tổng kết: 89](#_Toc8945814)

[VII. LINQ to XML: 89](#_Toc8945815)

[VII.1. Các khái niệm trong LINQ to XML: 90](#_Toc8945816)

[VII.1.1. XML - eXtensible Markup Language 90](#_Toc8945817)

[VII.1.2. Mô hình đối tượng tài liệu - Document Object Model (DOM): 105](#_Toc8945818)

[VII.1.3. Namespace System.Xml và đọc ghi file XML trong C#: 107](#_Toc8945819)

[VII.1.4. LINQ to XML: 111](#_Toc8945820)

[VII.1.5. Namespace System.Xml.Linq: 119](#_Toc8945821)

[VII.2. Xây dựng chương trình với LINQ to XML: 128](#_Toc8945822)

[VII.2.1. Định nghĩa các lớp mô hình dữ liệu: 128](#_Toc8945823)

[VII.2.2. Truy vấn cơ sở dữ liệu: 128](#_Toc8945824)

[VII.2.3. Cập nhật cơ sở dữ liệu: 129](#_Toc8945825)

[VII.2.4. : 129](#_Toc8945826)

[VII.3. Tổng kết: 129](#_Toc8945827)

[VIII. Tổng quan về LinQ to XML: 129](#_Toc8945828)

[IX. LINQ to XML so với DOM (C #): 133](#_Toc8945829)

[X. LINQ to XML so với các công nghệ XML khác: 136](#_Toc8945830)

[XI. Tài liệu tham khảo: 138](#_Toc8945831)

# Giới thiệu về LINQ:

LINQ là viết tắt của từ Language – Integrated Query tạm dịch là truy vấn tích hợp ngôn ngữ là một sự nâng cấp lớn trong C#3.0(phiên bản C# đã được Microsoft giới thiệu cùng với Visual Studio 2008 và .NET Framework 3.5), là cầu nối giữa thế giới của các đối tượng(objects) với thế giới của dữ liệu(data).

Theo truyền thống, các câu truy vấn dữ liệu được thể hiện một cách đơn giản giống như các chuỗi kí tự mà không cần đến kiểm tra kiểu tại thời điểm biên dịch hoặc sự tham gia của trình hỗ trợ trực quan. Hơn nữa bạn cần phải tìm hiểu một ngôn ngữ truy vấn riêng cho mỗi loại dữ liệu nguồn khác nhau như: Cở sở dữ liệu SQL, tài liệu XML, các dịch vụ Web(Web services). LINQ là truy vấn *first-class* xây dựng cho 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 và các toán tử quen thuộc.



# Giới thiệu vài nét về quá trình hình thành và phát triển của LINQ:

## Những ý tưởng ban đầu với ODBC, DAO, RDO và ODBCDirect:

Trong những năm đầu thập niên 90 của thế kỉ 20, Microsoft đã nghĩ đến việc xây dựng một API có thể giúp lập trình viên truy cập đến bất kì CSDL nào. Năm 1992, họ đã thành công khi xây dựng nên ODBC(Open Database Connectivity) để truy vấn tới cơ sở dữ liệu quan hệ SQL. Và sau đó là ODBC ISAM(Indexed Sequential Access Method) với khả năng làm việc với các CSDL không hỗ trợ truy vấn(XML) hay CSDL không tích hợp quan hệ(text, excel, dBase, Paradox files)

Cũng trong thời gian này, các khái niệm như Data Access Object (DAO) và Remote Data Objects (RDO) cũng lần lượt ra đời tạo tiền đề cho Microsoft xây dựng nên ODBCDirect: một chế độ cho phép DAO có thể tham gia truy vấn độc lập vào CSDL sử dụng ODBC

## Thành phần hóa quá trình truy cập dữ liệu (OLE DB):

Trong giai đoạn này, mục tiêu của các kĩ sư Microsoft là xây dựng nên các component truy cập dữ liệu có thể tái sử dụng và và phù hợp với kiến trúc thành phần COM(Component Object Model), và OLE DB(Object Linking and Embedding Database) ra đời.

Là một giao thức *first-class* dựa trên COM để làm việc với dữ liệu, OLE DB được xây dựng để hỗ trợ cho các khái niệm về CSDL thành phần hóa(Federated - Componentized) Database. Mục tiêu của bộ thư viện này là tạo nên một cấu trúc cơ sở chung “Rowset” có thể sử dụng để trình bày các bộ dữ liệu(các hàng) bất kể dữ liệu nguồn là gì và chúng đại diện cho cái gì: một kết quả truy vấn, bảng hoặc bề mặt giao tiếp(Interface), ISAM, text, excel, dữ liệu dạng sổ, email, directory services,... Trên nền tảng của Rowset, chúng ta có thể sử dụng các QueryInterface của COM để xác định bất cứ khi nào các chức năng bổ sung(như cuộn, cập nhật, hoặc điều hướng theo chỉ mục) được hỗ trợ.

Với cách biểu diễn dữ liệu này, Microsoft có thể xây dựng các component chung để làm việc với các kiểu dữ liệu, như một công cụ nhập chung, công cụ kiểm duyệt chung, công cụ truy vấn chung trên cả CSDL quan hệ và CSDL không quan hệ.

Cũng giống như ODBC, OLE DB cũng là một inteface “Cấp thấp” phải sử dụng con trỏ, quản lý bộ nhớ rõ ràng cà cần kiểm soát vòng đời. Và cũng giống như ODBC, nó yêu cầu nhiều gói API riêng biệt để có thể hoạt động với các ngôn ngữ tự động như VB. Vì thế các kĩ sư Microsoft cần tạo ra một định nghĩa mới để sử dụng cho bộ thư viện này, họ đã đảo vị trí các kí tự của DAO và cuối cùng lựa chọn cái tên ADO(ActiveX Data Objects).

ADO áp dụng cùng mô hình kết nối/lệnh/bản ghi của DAO, nhưng sử dụng một công nghệ con trỏ cục bộ tiên tiến hơn của FOX database team. “bản ghi ngoại tuyến”(Disconnected Recordset) của ADO sử dụng các con trỏ chỉ chuyển tiếp qua một luồng kết quả từ CSDL và các thuộc tính trên bản ghi để xác định xem dữ liệu có cuộn được hay cập nhật được hay không, cách li các dữ liệu từ kết quả và các dữ liệu từ kho dữ liệu. Bản ghi ngoại tuyến sử dụng dữ liệu từ truy vấn ban đầu để tạo các lệnh Insert, up date, delete và truyền chúng trở lại kho dữ liệu.

## .NET Framework và C# ra đời, thống nhất các components và API vào ADO:

Tháng 8 năm 1998, một nhóm các kĩ sư Microsoft đã tổ chức một buổi gặp mặt tại Semiahmoo, WA và thống nhất tạo nên BOLT(Beachside Offsite for Lightning Technology). Từ “Lightning” được sử dụng thay thế cho “COM3” và cuối cùng trở thành .NET Framework.

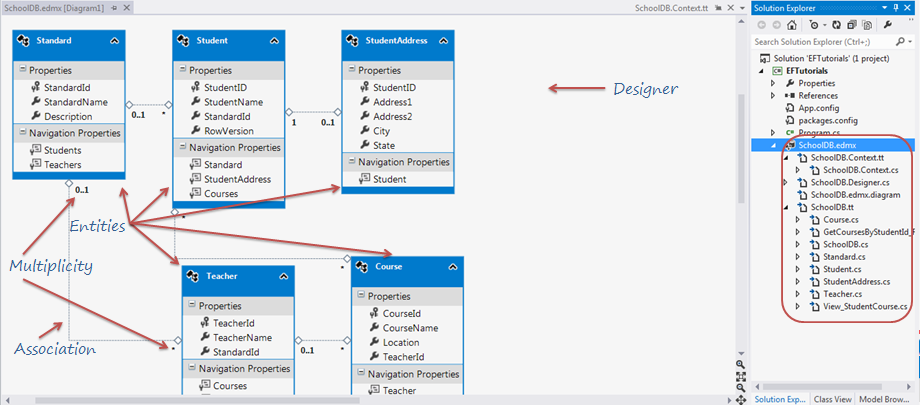
Trong giai đoạn này, các kĩ sư Microsoft đã tìm cách kế thừa nhiều nhất có thể những kiến trúc từ ADO/OLE DB và tái tạo thành ADO.NET. Cùng với sự phổ biến của Internet, disconected programming đã dần trở nên quan trọng và trở nên thông dụng hơn, XML trong giai đoạn này cũng trở thành ngôn ngữ phổ biến để làm việc với dữ liệu. ADO.NET giải quyết những thách thức mới này bằng cách phân tách rõ ràng truy vấn dữ liệu có kết nối(thông qua nhà cung cấp dữ liệu xác định “Data Provider”) với truy vấn dữ liệu không kết nối(thông qua bộ nhớ dữ liệu dùng chung “Dataset”) cùng với một cơ chế rõ ràng và có thể mở rộng “DataAdapter” để kết nối cả hai.

Nhờ thành công từ sức mạnh của sự tách biệt kết nối và không kết nối, việc kế thừa hợp lý từ ADO/OLE DB, và một sự tiến bộ lớn về nền tảng gốc với RAD(rapid application development), COM/DCOM, một .NET Framework gắn kết,.. ADO.NET đã nhận được nhiều sự hưởng ứng từ giới lập trình viên

## Lập trình cho mô hình quan niệm (ADO.NET Entities và LINQ):

1. ADO.NET Entities:

ADO.NET Entities được xây dựng bằng cách thêm khả năng để viết các ứng dụng dựa trên mô hình dữ liệu thực thể(Entity Data Model - EDM). Mô hình dữ liệu thực thể mở rộng mô hình dữ liệu quan hệ với cấu trúc quan hệ thực thể (Entity Relationship - ER) để mô hình hóa quan niệm thế giới thực như kế thừa(nhân khẩu thường trú và nhân khẩu tạm trú là các nhân khẩu), quan hệ(nhân khẩu có tiểu sử) và các thành phần phức tạp( địa chỉ một nhân khẩu bao gồm số nhà, tên xã phường, quận huyện, tỉnh thành phố). Một cấu trúc SQL mở rộng gọi là “Entity SQL” cho phép bạn truy vấn trực tiếp lược đồ quan hệ của mình, tận dụng sự kế thừa, truy cập các thành phần phức tạp và điều hướng các mối quan hệ. Trong nhiều trường hợp, việc xây dựng các khái niệm này thành lược đồ khái niệm và ngôn ngữ truy vấn sẽ giảm bớt sự cần thiết của các phép nối(join), liên hiệp(union), và các truy vấn con phức tạp để đơn giản hóa các truy vấn.



Các lược đồ khái niệm này được thể hiện và truy vấn thông qua một “Entity Client”. Entity Client này là một ADO.NET Data Provider xây dựng các câu truy vấn hướng tới nhà cung cấp dữ liệu nhất định theo hướng nhìn đọc/ghi của client. Các truy vấn và cập nhật được viết dựa trên các khung nhìn khái niệm này được mở rộng bởi Entity Client và được thực hiện dưới dạng các truy vấn của các nhà cung cấp dữ liệu nhất định. Tất cả việc thực hiện truy vấn thực tế được thực hiện trong store(không phải trong client) và các kết quả được tập hợp thành các kết thừa, đa hình, các kết quả với các thành phần lồng nhau và kết hợp. Sự tách biệt này giữa mô hình khái niệm mà ứng dụng hướng đến và lược đồ lưu trữ của CSDL là một khái niệm vô cùng mạnh mẽ mà các kĩ sư Microsoft tin rằng sẽ đơn giản hóa quá trình xây dựng và bảo trì các ứng dụng CSDL.

Trình diễn hướng nhìn Client thông qua ADO.NET Data Provider cho phép Microsoft giữ lại mô hình lập trình ADO.NET, tận dụng các đặc tính ở code, tools và hiểu biết đã có trước đó về ADO.NET. Trên thực tế, Entity Client sử dụng các ADO.NET Data Provider sẵn có (và mở rộng để phục vụ các tiểu chuẩn truy vấn biểu diễn hình cây mới) xây dựng dựa trên một cộng đồng luôn ủng hộ và liên tục tạo các tùy chỉnh cho ADO.NET Data Provider.

1. Đối tượng/Quan hệ (Object/Relational):

Kể từ phiên bản 1.0 của .NET Framework, người dùng đã liên tục tìm kiếm về một giải pháp đối tượng/quan hệ. Microsoft đã có một số nỗ lực tạo ra một số giải pháp O/R cho .NET, đáng kể nhất là “ObjectSpaces” mặc dù không được pháp hành tuy nhiên cũng được công bố dưới dạng “technical preview” tại PDC lúc họ giới thiệu về .NET Framework.

Tương tự hầu hết các giải pháp O/R hiện nay, ObjectSpaces đã cố gắng hỗ trợ một tập ánh xạ và kịch bản phong phú thông qua việc tạo các truy vấn tùy chỉnh. Thêm hỗ trợ cho một kiểu ánh xa kế thừa mới, với ý nghĩa là thêm code cho trình tạo truy vấn để chèn các điều kiện nối (join conditions) cần thiết vào các vị trí phù hợp. Dễ thấy rằng cấu trúc này sẽ khiến cho sự tương tác với các phép nối(join), phép chiếu(projection), liên kết(unions) và vị từ(predicates) được thêm vào trong suốt truy vấn để mô hình hóa cho các cấu trúc tương tự đối tượng khác làm cho code trở nên phức tạp và dễ vỡ hơn. Cố gắng để nắm bắt và cập nhật đối với các truy vấn phức tạp như vậy vừa khó khăn lại vừa dễ khiến cho nó trở nên phức tạp hơn nữa. Tệ nhất ở phương pháp này đó là sự khó khăn trong xác minh truy vấn tạo ra dựa trên các kết hợp này là đúng hay chưa.

ADO.NET Entities có một cách tiếp cận khác. Bằng cách mô hình hóa lược đồ khái niệm thông qua phía client thực hiện truy vấn và cập nhât khung nhìn, Entity Client tận dụng sự đầu tư quan trọng và nghiên cứu đã được thực hiện để tạo ra lý thuyết CSDL quan hệ. Chẳng hạn, việc cập nhật được thực hiện bằng cách áp dụng các kỹ thuật bảo trì khung nhìn để cập nhật khung nhìn để tạo ra một tập hợp các biểu thức delta được kết hợp với các chế độ truy vấn để tạo các biểu thức cập nhật. Kết quả của các khung nhìn truy vấn là xử lý cập nhật đều có thê kết hợp và kiểm chứng.

1. LINQ:

LINQ là viết tắt của cụm từ Language Intergrated Query, LINQ thống nhất các khái niệm truy vấn vào các ngôn ngữ lập trình, cho phép sử dụng code để truy vấn dữ liệu được kiểm tra bởi trình biên dịch và các công cụ dành cho lập trình viên giúp cho lập trình viên viết các câu truy vấn dễ dàng hơn. Điều này, cùng với các mô hình khái niệm cấp cao hơn (như Entities), góp phần giảm bớt tính không tương đồng giữa ứng dụng và dữ liệu.

LINQ được hỗ trợ như một thành phần first-class trong ADO.NET Entity Framework thông qua “LINQ to Entities”. LINQ to Entities là một phần của lớp đối tượng dịch vụ cho phép người dùng viết các truy vấn thông qua các biểu thức ngôn ngữ và các truy vấn tích hợp trong C# và VB như các câu truy vấn SQL. Điều này có nghĩa rằng các khái niệm khung nhìn client tương tự như các mẫu code của ADO.NET hiện có hoặc thông qua việc sử dụng đối tượng dịch vụ bằng cách sử dụng các truy vấn văn bản đặc biệt hoặc truy vấn tích hợp ngôn ngữ.

## Quá trình phát triển của LINQ:

Trong các phiên bản cập nhật kể từ .NET Framework 3.5, Microsoft không trực tiếp cập nhật vào LINQ, họ thường cập nhật các library liên quan tác động đến LINQ, điều này giúp thay đổi và cải thiện LINQ

# Những khái niệm cấu thành nên LINQ:

LINQ mở rộng ngôn ngữ C# bằng cách thêm các biểu thức truy vấn gần giống với các câu lệnh SQL và có thể sử dụng được để trích xuất và xử lý dữ liệu một cách thuận tiện từ các mảng(arrays), các lớp Enumerable, tài liệu XML, các CSDL quan hệ, và các CSDL của bên thứ ba. LINQ còn có thể dùng các biểu thức truy vấn như các câu lệnh chính thức trong bộ thư viện .NET Framework giúp cho việc truy vấn dữ liệu được thực hiện trực tiếp trên code một cách đễ dàng và linh động. Nó cũng định nghĩa một tập hợp các tên phương thức(được gọi là toán tử truy vấn tiêu chuẩn hoặc toán tử chuỗi tiêu chuẩn), cùng với các quy tắc dịch giúp trình biên dịch dịch các biểu thức truy vấn một cách trôi chảy thành các truy vấn trên các tên phương thức đã định nghĩa này, các biểu thức lambda và các kiểu ẩn danh(anonymous types). LINQ có được các khả năng này nhờ được tổng hợp từ nhiều khái niệm hay ho và phức tạp của ngôn ngữ C#, những khái niệm đó bao gồm:

## Phương thức mở rộng - Extension method:

Trong một số trường hợp của việc lập trình, ta muốn thêm method(phương thức) vào cho một class đã được niêm phong(Sealed Class), hoặc class từ các library(thư viện) khác. Với một số ngôn ngữ, điều này là không thể thực hiện được, nhưng với C#, chúng ta có thể dùng Extension method.

Xét ví dụ sau, chúng ta có class Student từ một library khác, không thể sửa code, ta muốn thêm method Print:

public class Student

{

    public string Name { get; set; }

    public int Age { get; set; }

}

Chúng ta tạo 1 extenstion class, class này phải là static class, method cũng phải static, params đầu tiên truyền vào là class cần extention, với từ khóa this.

public static class StudentExtension

{

   public static void Print(this Student student)

   {

      Console.WriteLine(student.ToString());

   }

}

//Sử dụng

var student = new Student();

student.Print();

## Gọi lại(Callback) và Ủy thác(Delegate) (Func, Predicate):

1. Nhắc lại về khái niệm Callback:

Callback la truyền một đoạn code (hàm A) vào một đoạn code khác (hàm B). tới một thời điểm nào đó của hàm B, hàm A này sẽ được gọi tới(Callback)

Ví dụ về Callback trong javascript:

function tangQua(qua) {

   return console.log("Đã tặng " + qua);

}

function oNha(vo, tangQua){

   var qua = "Quà đã nhận";

   tangQua(qua);

}

1. Delegate:

Để sử dụng được Callback thì hàm cần được gọi tới như một param tức là hàm đó cần được lưu như một object và có kiểu dữ liệu trả về. Thử viết lại hàm trên ở C# ta dễ dàng nhận thấy lỗi xảy ra khi tangQua không có kiểu dữ liệu trả về:

public void tangQua(string qua) {

   Console.Write("Da tang " + qua);

}

 //Kiểu dữ liệu cho params "vo" là Person, kiểu dữ liệu cho tangQua là gì???

public void oNha(Person vo, tangQua){

   var qua = "Quà đã nhận";

   tangQua(qua);

}

Ở đây, để truyền function tangQua vào, ta phải sử dụng kiểu dữ liệu Delegate (một Con trỏ hàm - ). Ta khai báo delegate theo cú pháp sau:

public delegate void TangQuaDelegate(string qua);

//delage + kiểu trả về (void) + tên delegate + (tham số truyền vào)

public void tangQua(string qua) {

   Console.Write("Da tang " + qua);

}

public void oNha(Person vo, TangQuaDelegate tangQua)

{

   var qua = "Quà đã nhận";

   tangQua(qua);

}

1. Action, Predicate, Func trong C#:

Action, Predicate, Func(APF) là một cách đơn giản hơn để khai báo delegate:

* Action: Action<T in1, T in2, …>. Action tương đương 1 delegate với kiểu trả về là void, với in1, in2 là các params nhận vào.
* Predicate: Predicate<T in>. Predicate tương đương 1 delegate với kiểu trả về là bool, với in là các param nhận vào. Predicate chỉ có thể nhận vào 1 param duy nhất.
* Func: Func<T in1, T in2, … , T result>. Function tương đương 1 delegate với kiểu trả về do ta khai báo (result), in1, in2 là các params nhận vào. Func bắt buộc phải trả ra giá trị, không thể trả void.

Để dễ hiểu, chhãy tham khảo bảng sau. Đây là bảng so sánh các khai báo bằng delegate, cùng với cách khai báo tương ứng bằng Action, Predicate, Func:

|  |  |  |  |
| --- | --- | --- | --- |
| **DELEGATE** | **ACTION** | **PREDICATE** | **FUNC** |
| delegate void VoidDelegate(int input1, bool input2) | Action<int, bool> |  |  |
| delegate bool BoolDelegate(int input1) |  | Predicate<int> | Func<int, bool> |
| delegate int intDelegate(bool input2) |  |  | Func<bool, int> |
| delegate void HelloWorldDelegate() | Action |  |  |
| delegate bool HelloWorldBoolDelegate() |  | Predicate | Func<bool> |

Đoạn code đầu có thể được viết lại ngắn gọn như sau:

public void tangQua(string qua) {

   Console.Write("Da tang " + qua);

}

public void oNha(Person vo, Action tangQua)

{

   var qua = "Quà đã nhận";

   tangQua(qua);

}

APF kết hợp với Lambda expression và một vài thư viện khác đã góp phần tạo nên sự mạnh mẽ của LINQ

## Biểu thức Lambda - Lambda Expression:

Từ .NET 2.0, Microsoft thêm vào một khái niệm đó là Anonymous Function để hỗ trợ cho cách viết Delegate:

public delegate void TangQuaDelegate(string qua);

TangQuaDelegate dlg =

          delegate(string qua) { Console.WriteLine("Tang quà" + qua); };

Nhận thấy sự rườm rà vẫn còn tồn tại trong cách viết này, sang đến phiên bản .NET 3.0 Microsoft cho ra mắt Lambda Expression. Ta cố thể hiểu Lambda Expression là một cách viết Anonymous Function ngắn gọn hơn:

//Cách cũ

TangQuaDelegate dlg = delegate(string qua) { Console.WriteLine("Tặng quà" + qua); };

//Dùng lambda expression

TangQuaDelegate lamdaDlg = (qua) => { Console.WriteLine("Tặng quà: " + qua); }

//Câu lệnh đầy đủ của lambda expression.

//Dấu "=>" gọi là go-to

(parameters) => { statement }



Dưới đây là một số quy tắc viết Lambda Expression:

**//1. Có thể bỏ qua kiểu dữ liệu của parameter truyền vào**

(string qua) => {Console.WriteLine("Tặng quà: " + qua);}

(qua) => {Console.WriteLine("Tặng quà: " + qua);}

**//2. Nếu không có parameter, bỏ dấu () trống**

() => {Console.WriteLine("Hello");}

**//3. Nếu chỉ có 1 parameter, có thể bỏ luôn dấu ()**

(x) => {Console.WriteLine("Hello " + x);}

x => {Console.WriteLine("Hello " + x);}

**//4. Nếu có nhiều parameter, ngăn cách bằng dấu phẩy**

(x, y) => {Console.WriteLine("Hello " + x + y);}

**//5. Nếu anonymous function chỉ có 1 câu lệnh, có thể bỏ dấu {}**

x => { Console.WriteLine("Hello " + x); }

x => Console.WriteLine("Hello " + x)

**//6. Nếu chỉ return 1 giá trị, có thể bỏ chữ return.**

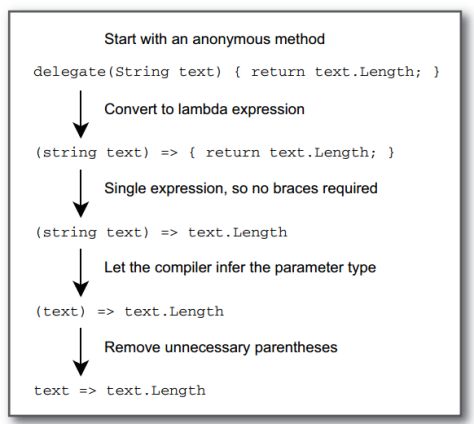
//4 lambda expression sau tương đương nhau

(x) => { return x > 4; }

x => { return x > 4; }

x => return x > 4

x => x > 4



*Các bước để rút gọn một Anonymous Function thành một Lambda Expression*

Lambda Expression kết hợp với APF tạo nên các hàm LINQ(Where(), Select(), First(),…). Ta sẽ xét ví dụ sau về hàm Where() để hiểu rõ hơn:

var studentList = new List<Student>();

//Sử dụng lambda expression

var students = studentList.Where(stu => stu.Age > 20);

//Nếu không có Lambda Expression, ta phải sử dụng Anonymous Function

var student = studentList.Where(new delegate(Student stu) { return stu.Age > 20; });

//Hoặc tệ hơn

public bool FindStudentWithAge(Student stu) { return stu.Age > 20; }

var student = studentList.Where(FindStudentWithAge);

Ở Ví dụ trên các tham số đầu vào của hàm Where là một Delegate Func, Lambda Expression cho phép chúng ta khai báo một cách ngắn gọn và dễ hiểu Func này giúp câu truy vấn LINQ trở nên ngắn gọn hơn.

## Mẫu - Generic:

Generic tham gia vào 50% những dòng code LINQ mà chúng ta viết

1. Generic là gì:

Rất nhiều người trong chúng ta sử dụng Generic mà không biết rằng đó là Generic, xét ví dụ về tạo một danh sách các học sinh như sau:

//Một list chứa các object là Student

List<Student> students = new List<Student>();

students.Add(new Student()); //Code đúng

students.Add(new Car()); //Compile lỗi

//Lấy học sinh đầu tiên.

//Compiler tự hiểu kết quả là Student

Student first = students.First();

Generic trong ví dụ trên chính là phần trong 2 dấu ngoặc nhọn <>

Để dễ hiểu, ta hãy quay lại thòi .NET 1.0, khi Generic chưa xuất hiện:

//Không có generic

//Một list chứa các object

List students = new List();

students.Add(new Student()); //Compile bình thường

students.Add(new Car()); //Compile bình thường

//Lấy object đầu tiên, phải ép kiểu sang Student

Student first = (Student)students.First();

Không có generic, compiler không thể check lỗi lúc compile. Do đó, ta có thể thêm 1 object Car và list gồm các object Student. Khi lấy 1 phần tử ra, ta cũng phải ép kiểu, vì compiler chỉ hiểu nó là 1 object. Vì những lý do đó, generic đã được thêm vào ở .NET 2.0. Tác dụng của generic:

* Giúp tái sử dụng code. Ví dụ: Ta chỉ cần viết class List<T>, T ở đây có thể là bất kì class gì.
* Hỗ trợ compiler bắt lỗi trong quá trình compiler (Hạn chế được tình trạng như dòng new Car()).
* Không còn phải ép kiểu từ object.

1. Sử dụng Generic:

Generic được ứng dụng nhiều trong các class List, Dictionary và LINQ,… sử dụng Generic cho LINQ giúp chúng ta có thể tái sử dụng hàm cho nhiều kiểu dữ liệu, xét ví dụ sau về hàm swap:

Nếu không có Generic:

public void swap (int ref a, int ref b);

public void swap (double ref a, double ref b);

public void swap (bool ref a, bool ref b);

public void swap (object ref a, object ref b);

.........

//Nếu muốn swap 2 student, phải viết thêm hàm

public void swap (Student ref a, Student ref b);

Với sức mạnh của Generic, ta chỉ cần viết 1 và chỉ 1 method duy nhất như sau:

//Dấu <T> để biết method là generic method

public static void Swap<T>(ref T a, ref T b)

{

  T temp = b;

  b = a;

  a = temp;

}

//Sử dụng hàm này như sau:

int firstInt = 1;

int secondInt = 2;

Swap<int>(ref firstInt, ref secondInt);

double firstDouble = 1.5;

double secondDouble = 2.5;

//Có thể bỏ qua dấu <> vì compiler tự hiểu kiểu double

//Dựa trên tham số ta truyền vào

Swap(ref firstDouble, ref secondDouble);

Hoặc xét một ví dụ hàm với giá trị trả về thuộc kiểu chưa xác định:

public static T getRandomElement<T>(List<T> list)

{

Random ran = new Random();

int randomIndex = ran.Next(list.Count - 1);

return list[randomIndex];

}

//Sử dụng

List<int> intList = new List<int> {1,2,3,4,5,6 };

List<double> doubleList = new List<double> {0.4, 0.6,0.8,4.5,0.2 };

var randomInt = getRandomElement(intList);

var randomDouble = getRandomElement(doubleList);

Generic Method kết hợp với Extension Method góp phần giúp cho các truy vấn LINQ trở nên đa dạng và nhất quán hơn.

## IEnumerable, IQueryable và Yield:

1. IEnumerable:

IEnumerable là một mảng read-only nằm trong namespace **System.Collections** với các tính chất sau:

* + - * + Enumerable có thể duyệt các phần tử chỉ 1 chiều tiến lên, nó không thể duyệt ngược lại giữa các phần tử.
        + IEnumerable hoạt động tốt nhất khi truy vấn từ một collection in-memory tức là trong bộ nhớ RAM như List, Array…
        + Khi truy vấn dữ liệu từ database, IEnumerable thực thi câu lệnh select trên server sau đó tải toàn bộ dữ liệu về client rồi mới lọc dữ liệu.
        + IEnumerable phù hợp với Linq to Object và Linq to XML
        + IEnumerable không hỗ trợ custom query
        + IEnumerable không hỗ trợ lazy loading(không load khi khởi tạo, chỉ load khi có yêu cầu) vì thế không phù hợp với trường hợp phân trang.

Ví dụ về IEnumerable:

MyDataContext dc = new MyDataContext ();

IEnumerable<Student> list = dc. Student s.Where(p => p.Name.StartsWith("S"));

list = list.Take< Student >(10);

Câu lệnh của đoạn code sẽ được generate sang SQL như sau:

SELECT [t0].[ID], [t0].[Name], [t0].[Class] FROM [Student] AS [t0] WHERE [t0].[Name] LIKE @p0

Chú ý là trong câu lệnh được tạo ra, sẽ không có “TOP 10” vì IEnumerable sẽ lọc các bản ghi ở dưới client. Nên toàn bộ bản ghi trước khi load ra top 10 sẽ được tải về client.

1. IQueryable:

IQueryable nằm trong namespace **System.Linq** được tạo ra để khắc phục các thiếu sót của IEnumerable, với các tính chất mới như sau:

* + - * + IQueryable cũng chỉ có thể di chuyển 1 chiều tiến lên trong collection, nó không thể move back lại.
        + IQueryable hoạt động tốt nhất với những truy vấn dữ liệu out-memory như là database.
        + Khi truy vấn, IQueryable thực thi câu lệnh truy vấn và lọc dữ liệu trực tiếp trên Server.
        + IQueryable phù hợp cho Linq to SQL
        + IQueryable hỗ trợ custom query sử dụng phương thức CreateQuery và Execute.
        + IQueryable hỗ trợ lazy loading. Vì thế nó phù hợp cho trường hợp phân trang.

Ví dụ về IQueryable:

MyDataContext dc = new MyDataContext ();

IQueryable<Student> list = dc. Student.Where(p => p.Name.StartsWith("S"));

list = list.Take<Student>(10);

Câu lệnh sẽ được Generate như sau:

SELECT TOP 10 [t0].[ID], [t0].[Name], [t0].[Class] FROM [Student] AS [t0] WHERE [t0].[Name] LIKE @p0

Chú ý: vì IQueryable thực thi câu lệnh và lọc dữ liệu trực tiếp trên server vì thế câu lệnh được tạo ra có thêm chữ “TOP 10”. Cũng vì lí do này, câu lệnh IQueryable sẽ không thực thi ngay khi khởi tạo mà sẽ nó đợi cho đến khi ToList() được gọi hoặc được đưa vào duyệt thì mới thực thi.

1. Yield:

Ta xét trường hợp sau, đọc danh sách học sinh từ file:

public List<Student> ReadStudentsFromFile(string fileName)

{

  string[] lines = File.ReadAllLines(fileName);

  //Tạo một list trống

  List<Student> result = new List<Student>();

  foreach (var line in lines)

  {

    Student student = ParseTextToStudent(line);

    result.Add(student); //Thêm student vào list

  }

  return result; // Trả list ra

}

var students = ReadStudentsFromFile("students.txt");

foreach(var student in students) {};

Đoạn code này không có gì sai. Tuy nhiên ta thấy việc tạo list, thêm phần tử vào list và trả về list có thể được rút gọn với từ khóa Yield như sau:

//Đổi kiểu trả về là IEnumerable

public IEnumerable<Student> ReadStudentsFromFile(string fileName)

 {

   string[] lines = File.ReadAllLines(fileName);

   foreach (var line in lines)

   {

      Student student = ParseTextToStudent(line);

      yield return student; //YIELD

   }

 }

//Dùng như cũ

var students = ReadStudentsFromFile("students.txt");

foreach(var student in students) {};

Việc sử dụng từ khóa Yield ở đoạn code trên không chỉ giúp số dòng code giảm đi mà còn mang lại rất nhiều lợi ích khác.

Chúng ta đều biết điều cơ bản nhất khi viết 1 method: Từ khóa return sẽ kết thúc method, trả ra kết quả, ko chạy thêm bất kì câu lệnh gì phía sau:

public int GetNumber() { return 5; }

Console.WriteLine(GetNumber());

Vậy xét trường hợp nếu ta dùng yield return 3 lần thì sao:

public IEnumerable<int> GetNumber()

{

yield return 5;

yield return 10;

yield return 15;

}

foreach(int i in GetNumber())Console.WriteLine(i);//5 10 15

Ta có thể hiểu luồng chạy của chương trình như sau:

1. Khi gọi method GetNumber, lấy phần từ đầu tiên, chương trình chạy tới dòng lệnh số 3, lấy ra kết quả là 5, in ra console.
2. Duyệt tiếp phần từ tiếp theo, chương trình chạy vào dòng lệnh số 4, lấy kết quả 10, in ra màn hình.
3. Tương tự với phần tử cuối cùng, sau khi in ra, chương trình kết thúc.

Xét với trường hợp 2 method Student ở đầu:

* + - * + Ở method đầu, ta trả về kết quả sau khi đã chạy hết hàm for, đưa kết quả vào trong 1 list mới, hàm ReadStudentsFromFile kết thúc.
        + Ở method thứ 2, kết quả được trả về ngay sau khi parse được student đầu tiên, với mỗi vòng lặp tiếp theo, chương trình sẽ chạy tiếp vào method YieldReadStudentsFromFile, lấy kết quả ra dần dần.

Sau khi hiểu bản chất, ta có thể ứng dụng yield vào các trường hợp như sau:

* + - * + Cần method trả về là một danh sách read-only, chỉ đọc, không được thêm xóa sửa.
        + Như trường hợp trên, giả sử ta có 50 dòng, hàm ParseTextToStudent tốn 1s 1 lần. Với cách cũ, khi gọi hàm ReadStudentsFromFile, ta phải đợi 50s. Với hàm YieldReadStudentsFromFile, hàm ParseTextToStudent chỉ được chạy mỗi khi ta đọc thông tin của học sinh, do đó tăng performance lên rất nhiều (Nếu ta chỉ lấy 5 học sinh đầu chỉ cần đợi 5s).
        + Trong một số trường hợp, danh sách trả về có vô hạn phần tử, hoăc lấy toàn bộ phần tử rất mất thời gian, ta phải sử dụng yield để giải quyết.

Thực tế khi dùng hàm yield, C# sẽ compile method đã viết lại thành 1 state machine, implement các method Next, Current, Max,… của Ienumerator. Chi tiết tại: http://coding.abel.nu/2011/12/return-ienumerable-with-yield-return/

# Cấu trúc và đặc tính của LINQ:

## Cấu trúc cơ bản của LINQ và biểu thức LINQ:

Namespace **System.Linq**;

Namespace **System.Data.Linq**;

Thực tế LINQ là một danh sách các Extension method, thêm một số method cho Interface IEnumerable (các class List, Dbset, ArrayList, Dictionary,… đều kế thừa từ IEnumerable do đó chúng cũng có thể gọi Linq method).

Linq cơ bản bao gồm 3 signature: **Where, Select và First.**

public static class Enumerable{

   public static IEnumerable<TSource> **Where**<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);

   public static IEnumerable<TResult> **Select**<TSource, TResult>(this IEnumerable<TSource> source, Func<TSource, TResult> selector);

   public static TSource **First**<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate);

}

Các method này nhận đầu vào là 1 delegate Func<> với tham số thuộc kiểu Generic(Tsource, TResult) do đó ta có thể khai báo kiểu gì cũng được.

Chúng ta cũng có thể tự tạo ra được Linq method cho riêng mình bằng cách viết Extension method cho IEnumerable. Với signature tương tự như Linq, ví dụ:

public static class MyLinq

{

  //Extension method

  public static IEnumerable<TSource> MyWhere<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate)

  {

    //Loop toàn bộ cái item trong danh sách truyền vào

    foreach (var item in source)

    {

      // Callback lại hàm đã truyền vào, hàm này trả giá trị boolean

      // Nếu hàm callback return true, đưa item đó vào IENumerable kết quả

      // Xem lại vài IEnumerable và yield

      if (predicate(item))  yield return item;

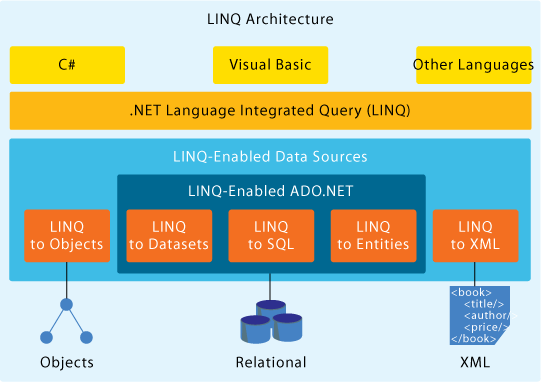
    }

  }

}

Nhờ đặc tính này cũng như một số đặc tính mềm dẻo khác của LINQ, rất nhiều thư viện LINQ mở rông đã được xây dựng (LINQ to JSON: <https://www.newtonsoft.com/json/help/html/LINQtoJSON.htm>, LINQ for MongoDB: <https://mongodb.github.io/mongo-csharp-driver/1.11/linq/>, GraphQL to Linq.IQueryable: <https://github.com/ckimes89/graphql-net>,...)

Trong một truy vấn LINQ, bạn phải luôn luôn làm việc với các đối tượng. Bạn sử dụng giống như một truy vấn cơ bản và mã hóa, chuyển đổi dữ liệu sang 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 thư viện LINQ to phép.

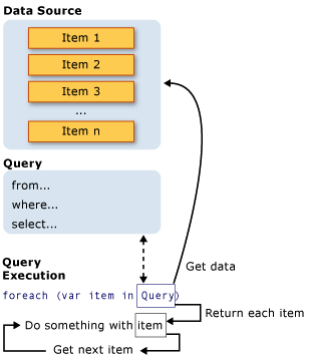


### Ba phần của một biểu thức LINQ:

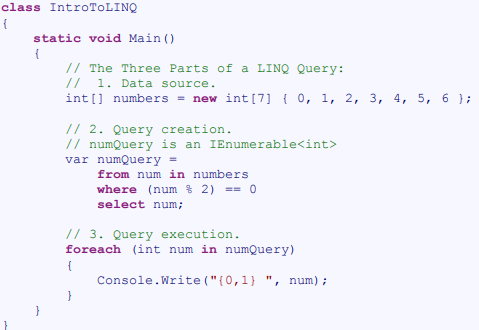
Tất cả các biểu thức LINQ làm việc theo ba thao tác:

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

Minh họa sau đây cho thấy các hoạt động truy vấn tìm kiếm được hoàn tất. Trong LINQ các truy vấn thực hiện 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.



Ví dụ sử dụng một mảng số nguyên như là một dữ liệu nguồn (khái niệm này có thể áp dụng cho tất cả các dữ liệu nguồn). Ví dụ này sẽ được sử dụng trong suốt phần còn lại của chủ đề này.



### Các dữ liệu nguồn:

Trong ví dụ trước vì dữ liệu là một mảng, nó hoàn toàn có các điểm chung của IEnumerable. Do đó chúng ta có thể thực hiện truy vấn LINQ lên nó. Một truy vấn được thực hiện trong một câu lệnh foreach và foreach yêu cầu đầu vào là IEnumerable hay IEnumerable<T>. Các kiểu có hỗ trợ IEnumerable<T> hoặc các Interface 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 LINQ. Nếu một CSDL chưa được định nghĩa trong bộ nhớ như là một queryable, một nhà cung cấp LINQ phải xay dựng nó. Ví dụ, LINQ to XML giúp đưa dữ liệu XML vào một queryable Xelement

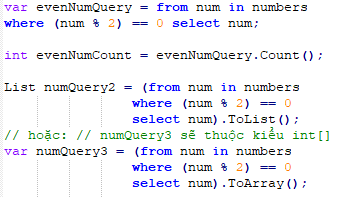
Với LINQ to SQL trước tiên bạn phải thiết kế một đối tượng quan hệ, công việc này được làm thủ công hoặc bằng cách sử dụng trình thiết kế đối tượng quan hệ(O/R Designer).

### Truy vấn và thực thi 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**(Vị trí của các mệnh đề có đôi chút thay đổi so với trong SQL).

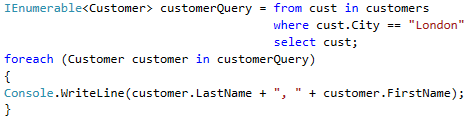
Mệnh đề **from** dùng để xác định dữ liệu nguồn, mệnh đề **where** dùng để lọc dữ liệu, mệnh đề **select** dùng để chọn ra những phần tử được trả về

* + - 1. Thực thi truy vấn:
         * Cũng giống như trạng thái trước, biến truy vấn chỉ chứa các lệnh truy vấn. Sự thực thi của các truy vấn sẽ hoãn lại đến khi bạn gọi tới biến truy vấn trong câu lệnh foreach. Cái này làm cơ sở đệ quy cho hoãn thực thi(Lazy Loading).
         * Câu lệnh foreach là nơi các kết quả truy vấn được trả về, ở ví dụ trên, biến num được lặp và nắm giữ mỗi giá trị trong trình tự trả về.
         * các biến truy vấn tự nó không bao giờ chứa kết quả truy vấn, bạn có thể thực hiện nó nhiều lần nếu muốn
      2. Thực thi bắt buộc tức thời:
         * Là truy vấn thực hiện kết hợp các chức năng trên một loạt các phần tử nguồn đầu tiên và phải lặp đi lặp lại trên danh sách các nhần tử. Ví dụ như các truy vấn Count, Max, Average, và First.
         * Những thực thi này không có một câu lệnh foreach nào rõ ràng bởi vì các truy vấn tự nó phải sử dụng foreach để trả về một kết quả.



### Các biến IEnumerable có trong các câu truy vấn LINQ:

Các biến kết quả trong câu truy vấn LINQ sẽ có kiểu IEnumerable<T> hoặc có kiểu IQueryable<T>. Ví dụ bạn viết truy vấn với biến kiểu IEnumerable<Customer>, nó đơn giản là các thức truy vấn, khi nó được thực hiện sẽ tạo ra một chuỗi gồm không hoặc nhiều đối tượng Customer.



## Các hoạt động truy vấn cơ bản:

Cấu trúc các truy vấn của LINQ chủ yếu tương tự như ở các ngôn ngữ truy vấn như SQL hoặc XQuery

### Truy xuất dữ liệu nguồn:

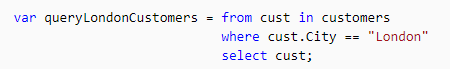
Trong một truy vấn LINQ, bước đầu tiên là xác định dữ liệu nguồn. Trong C# cũng như trong hầu hết các ngôn ngữ lập trình một biến phải được khai báo trước khi nó có thể được sử dụng. Trong một truy vấn LINQ, mệnh đề from đứng đầu tiên để giới thiệu các dữ liệu nguồn (customer) và các phạm vi biến (cust).



Phạm vi biến giống như các biến lặp trong một vòng lặp foreach, ngoại trừ không có một vòng lặp thực sự nào xảy ra trong một biểu thức truy vấn. Khi truy vấn được thực thi, phạm vi biến sẽ là một tham chiếu lần lượt đến các các phần tử trong mỗi customers. Bởi vì trình biên dịch có thể phỏng đoán kiểu của cust, bạn không cần phải xác định nó trước. Ngoài ra phạm vi các biến cũng có thể được khai báo bởi một mệnh đề let.

### Lọc (Filtering):

Hoạt động truy vấn dữ liệu thông dụng nhất là áp dụng một bộ lọc vào biểu thức logic Boolean. Các bộ lọc giúp các truy vấn trả về những phần tử mà qua nó biểu thức trả về kết quả true. Kết quả được trình bày bằng cách sử dụng mệnh đề where. Các bộ lọc có tác dụng xác định các yếu tố đó để loại trừ liên tục từ dữ liệu nguồn. Trong ví dụ sau, sẽ chỉ trả về những khách hàng có địa chỉ ở London



Khi truy vấn, chúng ta có thể sử dụng các biểu thức tương tự như trong C# với các biểu thức logic AND và OR để áp dụng cho bộ lọc có nhiều điều kiện 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 code sau đây:

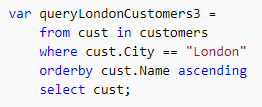


Để trả về khách hàng có địa chỉ ở London hay Paris, bạn sẽ viết mã sau:



### Sắp xếp (Ordering):

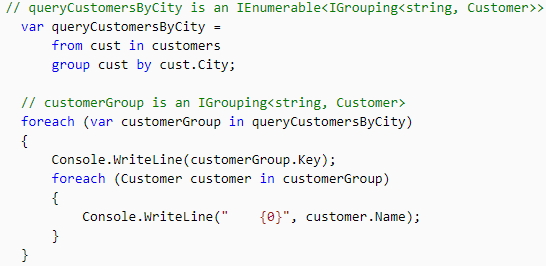
Mệnh đề orderby sẽ giúp các phần tử trong chuỗi trả về để được sắp xếp theo loại và so sánh với kiểu của dữ liệu được khác định. Ví dụ, truy vấn sau đây được viết thêm phân loại các kết quả dựa trên thuộc tính Name. Bởi vì thuộc tính Name là một chuỗi, mặc định nó sẽ so sánh và thực hiện sắp xếp theo thứ tự chữ cái từ A đến Z.



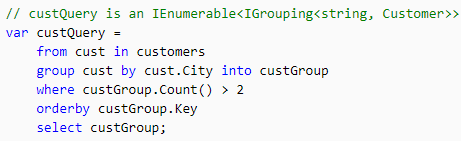
Để các kết quả trong thứ tự đảo ngược trật tự, từ Z to A, sử dụng mệnh đề orderby ...descending.

### Gom nhóm (Grouping):

Mệnh đề group cho phép bạn nhóm các kết quả lại 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.

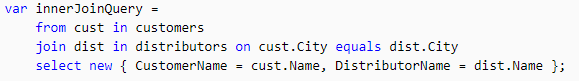


Khi bạn kết thúc một truy vấn với mệnh đề group, các kết quả của bạn sẽ sẽ được trả về một danh sách các danh sách. Mỗi phần tử trong danh sách là một đối tượng trong đó có một Key và danh sách các phần tử được gom nhóm bởi key này. Khi bạn duyệt qua một truy vấn mà kết quả là một nhóm có thứ tự, bạn cần phải sử dụng một vòng lặp foreach. Nếu bạn muốn tham khảo các kết quả thực thi của việc gom nhóm, bạn có thể sử dụng từ khóa into để tạo ra một định danh có thể truy vấn được. Dưới đây là những truy vấn trả về chỉ những nhóm có chứa nhiều hơn hai khách hàng



### Kết (Joining):

Thao tác Join tạo ra sự kết hợp giữa nhiều bảng không được mô hình hóa rõ ràng trong các dữ liệu nguồn. Ví dụ, bạn có thể thực hiện một thao tác để tìm tất cả các khách hàng và nhà phân phối ở cùng một địa điểm. Trong LINQ mệnh đề Join luôn luôn tham gia các hoạt động trên tập đối tượng thay vì các bảng CSDL.



*Trong LINQ bạn không nhất thiết phải sử dụng mệnh đề Join thường xuyên như trong SQL bởi vì các khóa ngoại LINQ miêu tả trong mô hình như là một tập chứa danh sách các đối tượng liên quan. Ví dụ, một đối tượng Customer có chứa một tập các đối tượng Order. Thay vì sử dụng join, bạn có thể truy cập tập dữ liệu bằng dấu chấm*



### Lựa chọn (Selection/Projection):

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

**Bảng các toán tử thông dụng của LINQ:**

|  |  |  |
| --- | --- | --- |
| Filtering | Where, OfType | |
| Join | Join, GroupJoin | |
| Projection | SelectMany |
| Sorting | OrserBy, OrderByDescending, ThenBy, ThenByDescending, Reverse |
| Grouping | GroupBy, ToLookup |
| Conversion | AsEnumerable, AsQueryable, Cast, OfType, ToArray, ToDictionary, ToList, ToLookup |
| Concatenation | Concat |
| Aggretion | Aggregate, Average, Count, LonCount, Max, Min, Sum |
| Quantifier | All, Any, Contains |
| Partition | Skip, SkipWhile, Take, TakeWhile |
| Generation | DefaultIfEmpty, Empty, Range, Repeat |
| Set | Distinct, Except, Intersect, Union |
| Equality | SequenceEqual |
| Element | ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault, DefaultIfEmpty |

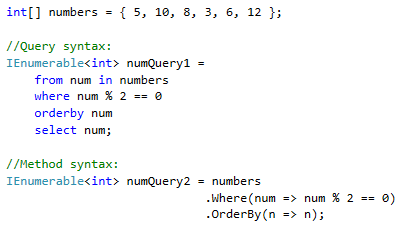
## Chuyển đổi dữ liệu với LINQ:

LINQ không chỉ là truy xuất dữ liệu. LINQ còn là một công cụ mạnh mẽ cho việc biến đổi dữ liệu. Bằng cách sử dụng một truy vấn LINQ, bạn có thể sử dụng một Danh sách dữ liệu nguồn vào và sửa đổi nó bằng nhiều cách để tạo ra một danh sách mới. Bạn có thể sửa đổi trình tự của nó mà không sửa đổi các phần tử bên trong bằng cách phân loại và gom nhóm. Tính năng mạnh mẽ nhất trong của các câu truy vấn LINQ là khả năng tạo ra kiểu mới(sử dụng Anonymous Type). Tính năng này được áp dụng nhiều nhất trong mệnh đề select.

## Cú pháp truy vấn (Query Syntax) và cú pháp phương thức (Method Syntax):

Hầu hết các truy vấn LINQ thông dụng đều được viết bằng cách sử dụng cú pháp truy vấn(Query Syntax) theo LINQ. Tuy nhiên, cú pháp truy vấn phải được dịch thành các lệnh gọi phương thức dành cho .NET common language runtime (CLR) khi code được biên dịch. Các lệnh gọi phương thức này sẽ sử dụng các toán tử truy vấn tiêu chuẩn, các toán tử này có các tên Where, Select, GroupBy, Join, Max và Average. Bạn có thể gọi chúng trực tiếp bằng cách sử dụng cú pháp phương thức(Method Syntax) thay vì cú pháp truy vấn.

Query Syntax và Method Syntax giống hệt nhau về mặt ngữ nghĩa, nhưng nhiều người cho rằng Query Syntax đơn giản hơn và dễ đọc hơn. Một số truy vấn phải được thể hiện dưới dạng các lệnh gọi phương thức. Ví dụ: bạn phải sử dụng lệnh gọi phương thức để thể hiện một truy vấn truy xuất số lượng phần tử phù hợp với một điều kiện nhất định. Bạn cũng phải sử dụng lệnh gọi phương thức cho truy vấn truy xuất phần tử có giá trị tối đa trong chuỗi nguồn. Tài liệu tham khảo cho các toán tử truy vấn tiêu chuẩn trong namespace **System.Linq** thường sử dụng Query Syntax. Do đó, ngay cả khi bắt đầu viết các truy vấn LINQ, việc làm quen với cách sử dụng Query Syntax trong các truy vấn và trong các biểu thức truy vấn là rất hữu ích.



## LINQ và Kiểu mẫu (Generic Type):

Như đã nêu ở mục III.4, Các câu truy vấn LINQ được dựa trên các kiểu dạng chung hay kiểu mẫu(Generic type), đã được giới thiệu trong phiên bản 2.0 của. NET Framework. Bạn không cần thiết phải có các kiến thức chuyên sâu về các Generic Type này trước khi bắt đầu viết các truy vấn. Tuy nhiên, bạn nên hiểu rõ hai khái niệm cơ bản:

1. Khi bạn tạo một thể hiện từ danh sách kiểu Generic như List<T>, bạn thay thế "T" với các loại đối tượng trong List 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 các Generic Type cung cấp nhiều lợi ích cho một tập hợp khi xem các phần tử của chúng như đối tượng. Nếu bạn cố gắng để thêm một Customer vào trong một List<string>, bạn sẽ nhận được một lỗi compile. Sử dụng các tập hợp có chung đặc điểm rất dễ dàng và tiện lợi vì bạn không cần phải thực hiện ép kiểu trong thời gian chạy chương trình(run-time type-casting).
2. IEnumerable<T> là một Interface cho phép tạo các lớp Generic Collection có thể liệt kê được bằng cách sử dụng câu lệnh foreach. Các lớp Generic Collection hỗ trợ IEnumerable<T> tương tự như cách các lớp non-generic collection như ArrayList hỗ trợ IEnumerable.

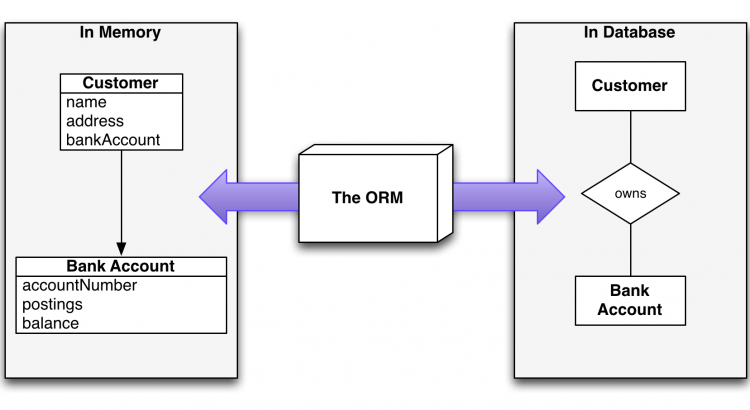
## Mô hình đối tượng quan hệ (Object Relational Mapping) và thiết kế đối tượng quan hệ (Object Relational Designer):

### Mô hình đối tượng quan hệ (ORM):

#### Tổng quan:

Object Relational Mapping hay ORM là tên gọi chỉ việc ánh xạ các bản ghi dữ liệu trong hệ quản trị cơ sở dữ liệu sang dạng đối tượng mà mã nguồn định nghĩa trong class. Là một khái niệm phổ biến, được cài đặt trong tất cả các loại ngôn ngữ hiện đại ngày nay như: java, php, node.js, swift, C# … Do đó, chúng ta có thể dễ dàng có thể cài đặt ORM hoặc sử dụng các thư viện mã nguồn mở về ORM trong bất cứ dự án nào mình thích.

Để cài đặt kỹ thuật này đơn giản là chúng ta cài đặt một lớp thực hiện việc kết nối CSDL và các chức năng tương tác với CSDL như Select, Insert, Delete, Update và các tương tác trên các đối tượng dữ liệu có quan hệ với nhau.



object relational mapping tự động hóa chuyển đổi từ object sang table và ngược lại từ table sang object rất nhanh chóng giúp giảm thời gian cũng như chi phí phát triển. Object relational mapping cũng cần ít code hơn sotre procedures. Object relational mapping có khả năng thay thế số lượng lớn store procedudres cần phát triển. Bên cạnh đó, ORM cũng giúp tăng tốc thực thi của hệ thống.

#### Những ưu điểm và nhược điểm của ORM:

* Ưu điểm:
  + **Portable – tính năng động:** ORM được sử dụng để bạn viết cấu trúc của bạn một lần và lớp ORM sẽ xử lý câu lệnh cuối cùng phù hợp với DBMS được cấu hình. Đây là một lợi thế tuyệt vời khi thao tác truy xuất dữ liệu đơn giản như giới hạn được thêm vào dưới dạng ‘limit 0,100’ ở cuối câu lệnh Select trong MySQL, trong khi đó với cách truy xuất thông thường bạn phải viết là ‘Select Top 100 From Table’ trong MS SQL.
  + **Nesting Of Data – truy xuất lồng dữ liệu:** trong trường hợp database bạn có nhiều bảng và các bảng này liên hệ rối rắm phức tạp về dữ liệu thì ORM sẽ tự động lấy dữ liệu cho bạn một cách đơn giản (ở đây đang nói tới việc đơn giản trong truy xuất dữ liệu, không bàn về vấn đề tối ưu truy xuất)
  + **Single Language – không cần biết SQL:** thật vậy với nguyên lý thiết kế là ánh xạ toàn bộ dữ liệu lấy được từ DBMS sang bộ nhớ nên việc thao tác truy suất bây giờ chỉ phụ thuộc vào ngôn ngữ lập trình bạn đang xử dụng, bạn chẳng cần quan tâm phía đằng sau của ORM sẽ làm gì sinh ra mã SQL như thế nào khi truy xuất SQL, và kết quả là chúng ta chỉ cần nhuần nhuyễn ngôn ngữ lập trình đang dùng.
  + **Adding is like modifying – thêm sửa dữ liệu là như nhau:** đối với ORM, nó không phân biệt giữa thêm mới và cập nhật mọi tác vụ có liên quan đến sửa đổi hay chèn dữ liệu đều được xem là định nghĩa thêm mới, hai tác vụ này được xem như là một.
* Nhược điểm:
  + **Slow – chậm:** trong tất cả các bài đánh giá và so sánh về tốc độ truy suất dữ liệu bạn sẽ thấy ORM dù cho được tối ưu đến đâu cũng sẽ không bao giờ nhanh hơn được so với truy vấn trực tiếp từ SQL.
  + **Tunning – khó điều chỉnh:** Tối ưu hóa các câu lệnh SQL là điều nên làm đối với các dữ liệu lớn, tuy nhiên đối với ORM quá trình này rất khó khăn và phức tạp, bởi vì bạn phải tuân theo thiết kế của ORM và ưu điểm mà nó mang lại là hạn chế và cô lập quá trình thao tác dữ liệu trực tiếp lên DBMS.
  + **Complex Queries – truy vấn phức tạp:** mặc dù được thiết kế để đơn giản hóa tuy nhiên một vài ORM thao tác dữ liệu rất rối rắm, các đoạn code rất phức tạp và khó hiểu hơn nhiều lần so với SQL thuần.
  + **Studing – khó học:** Đa phần các hệ thống với các tác vụ nặng và phức tạp thư viện ORM thuần túy được cung cấp không đảm bảo về hiệu năng, và bạn cùng các thành viên trong team chắc chắn phải nghiên cứu từng dòng code để tối ưu cho hệ thống của mình.

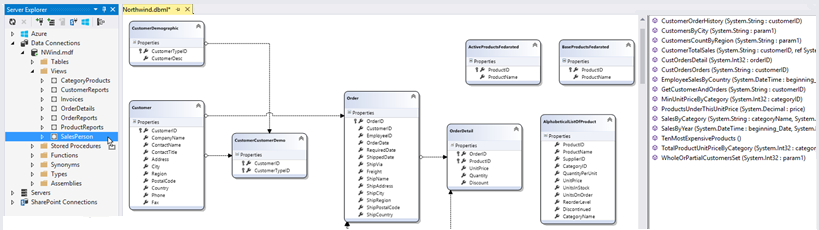
### Thiết kế đối tượng quan hệ (O/R desgner):

#### Tổng quan:

Thiết kế đối tượng quan hệ (O/R Designer) cung cấp một 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 kiểu DataContext rõ ràng để gửi và nhận dữ liệu giữa các lớp thực thể và cơ sở dữ liệu.

O/R Designer cũng cung cấp khả năng ánh xạ các thủ tục lưu trữ và các hàm để thực hiện các phương thức trong DataContext trả về các dữ liệu và các lớp thực thể. Ngoài ra, O/R Designer cung cấp khả năng thiết kế thừa kế các mối quan hệ giữa các lớp thực thể.

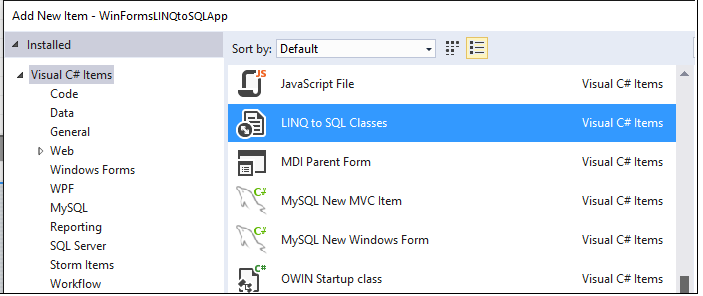
Giao diện thiết kế O/R Designer gồm hai khu vực riêng biệt: các thực thể(Entity) trong của sổ bên trái, và các phương thức(Method) trong cửa sổ bên phải. Cửa sổ các thực thể chính là giao diện thiết kế lớp thực thể, các liên kết, và các bậc kế thừa. Các phương thức trong của sổ bên phải là giao diện hiển thị DataContext các phương thức ánh xạ để lưu trữ các thủ tục và hàm.



#### Phương pháp thiết kế đối tượng quan hệ:

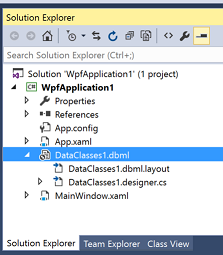
Việc xây dựng đối tượng quan hệ trong LINQ bao gồm ba thao tác:

1. Chọn Project > Add new item. Tại màn hình khởi tạo lựa chọn Data -> LINQ to SQL Classes, Đặt tên và nhấp chọn Add



1. Visual Studio sẽ tạo một file .dbml và thêm nó vào solution. Đây là file ánh xạ XML và các mã liên quan.

Lưu ý: Tên của biến DataContext được sử dụng cho việc truy vấn sau này sẽ có cấu trúc là: [tên\_file]+DataContext



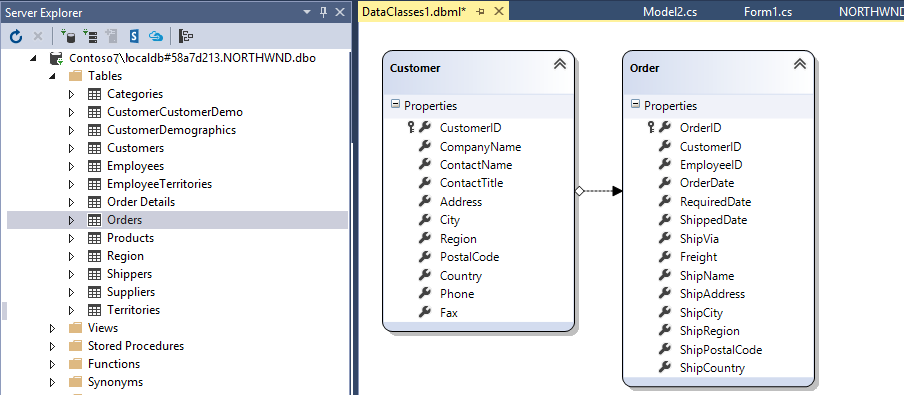
1. Khi bạn mở tệp .dbml, Visual Studio hiển thị giao diện trình thiết kế O/R cho phép bạn tạo mô hình một cách trực quan:
   * 1. Cấu hình và khởi tạo DataContext:

Sau khi bạn thêm một lớp LINQ to SQL cho một Project và mở thiết kế O/R, giao diện sẽ trống rỗng, DataContext sẵn sàng để được cấu hình. các DataContext được cấu hình sẽ kết nối với các thông tin được cung cấp bởi các thực thể 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 thực thể đầu tiên được kéo vào bản thiết kế.

* + 1. Tạo các lớp thực thể và kết hợp chúng với bảng dữ liệu:

Bạn có thể tạo các lớp thực thể được ánh xạ từ các bảng và các view bằng cách kéo thả các Tables và các Views từ Server Explorer/Explorer Database lên các bản thiết kế O/R. Như đã nói trong phần trước, DataContext được cấu hình để kết nối với các thông tin được cung cấp bởi các thực thể đầu tiên được kéo thả vào bản thiết kế. *Nếu một thực thể sử dụng một kết nối khác được thêm vào bản thiết kế O/R, bạn có thể thay đổi kết nối thủ công cho thực thể này trong DataContext, O/R designer cho phép tạo thiết kế đối tượng quan hệ từ nhiều CSDL khác nhau.*

Mệnh đề **from in** dùng để xác định dữ liệu nguồn, mệnh đề **where** dùng để lọc dữ liệu, mệnh đề **select** dùng để chọn ra những phần tử được trả về



* + 1. Tạo phương thức DataContext để chứa các thủ tục và hàm:

Bạn có thể tạo DataContext chứa các phương thức mà gọi (được ánh xạ tới) các thủ tục(StoredProcedure) và các hàm(Function) lưu trữ của CSDL bằng cách kéo chúng từ Server Explorer/Explorer Database lên các O/R Designer. Các thủ tục lưu trữ và các hàm này sẽ được sử dụng như các phương thức của DataContext.

* + 1. Cấu trúc DataContext để chứa các thủ tục lưu trữ dữ liệu giữa các lớp thực thể và cơ sở dữ liệu:

Như đã nêu trên, bạn có thể thêm các thủ tục lưu trữ(StoredProcedure) và các hàm(Function) và sử dụng chúng như một phương thức của DataContext. Ngoài ra, bạn cũng có thể chỉ định các thủ tục lưu trữ này để làm phương thức mặc định cho LINQ to SQL để thực hiện hành động insert, update, và delete.

### Các đặc tính của Thiết Kế Đối Tượng Quan Hệ:

Khóa: Các khóa chính và khóa ngoại trong O/R Designer đều có chức năng như các khóa chính và khóa ngoại trong CSDL(SQL). Đặc biệt, khi truy vấn đối tượng dữ liệu có tồn tại khóa ngoại, O/R Designer còn hỗ trợ truy vấn đến dữ liệu bảng chứa thông tin của khóa ngoại đó.

Hệ thống các quan hệ (Relationship): Giống như các đối tượng khác, các lớp LINQ to SQL có thể sử dụng được tính kế thừa và chuyển hóa từ các lớp khác. Trong một CSDL, các mối quan hệ thừa kế được tạo ra trong một số bảng. O/R Designer hỗ trợ các khái niệm về đơn(bảng) kế thừa như trong các hệ thống quan hệ của ngôn ngữ lập trình C#. Liên hệ (Association) giữa các thực thể trong LINQ to SQL cũng tương tự như mối quan hệ giữa các bảng trong CSDL, bạn có thể sử dụng hộp thoại Asociation Editor để tạo các quan hệ này.

Các truy vấn LINQ to SQL: Các lớp thực thể tạo bởi O/R Designer được thiết kế để sử dụng với các truy vấn LINQ.

Phân tách lớp DataContext và lớp thực thể(Entity) vào các namespace khác nhau: O/R Designer cung cấp các lựa chọn **Context Namespace** và **Entity Namespace** trên DataContext. Các thuộc tính này xác định namespace mà DataContext và Entity class được tạo. Thông thường lựa chọn này sẽ rỗng và DataContext, Entity class sẽ được tạo ở namespace của phần mềm. Để tạo code vào namespace khác với namespace của phần mềm, bạn có thể nhập giá trị vào Properties của Context Namespace và/hoặc Entity Namespace

# LINQ to SQL:

using **System.Linq**; **System.Data.Linq**; **System.ComponentModel**;

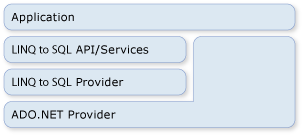
LINQ to SQL là một thành phần của .NET Framework (3.5 trở lên) cung cấp cơ sở hạ tầng thời gian thực để giúp quản lý các dữ liệu quan hệ như các đối tượng.

*Chú ý*: Các dữ liệu quan hệ sẽ có giao diện như là một danh sách của các bảng 2 chiều (các quan hệ hoặc các flat file), nơi mà các cột bảng liên quan đến nhau. Để sử dụng LINQ to 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 to 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 lập trình viên. Khi các ứng dụng khởi chạy, LINQ to SQL dịch các truy vấn LINQ trong các mô hình đối tượng thành các truy vấn SQL và gửi chúng đến CSDL để thực thi. Khi cơ sở dữ liệu trả về kết quả, LINQ to SQL chuyển chúng trở lại thành các đối tượng để có thể sử dụng cho ngôn ngữ lập trình. LINQ to SQL hỗ trợ đầy đủ transaction, view và các stored procedure (SP).

Các lập trình viên sử dụng Visual Studio thông thường sẽ sử dụng các Object/Relational Designer (thết kế đối tượng quan hệ), thứ có thể cung cấp giao diện người dùng giúp họ thực thi nhiều tính năng của LINQ to SQL.

LINQ to SQL là một phần của dòng công nghệ ADO.NET. Nó được xây dựng dựa trên các dịch vụ được cung cấp bởi mô hình ADO.NET provider. Do vậy, bạn có thể phối hợp code LINQ to SQL với các ADO.NET application sẵn có và sử dụng các ADO.NET Provider cho LINQ to SQL. Ví dụ minh họa sau cung cấp cái nhìn cơ sở về các mối quan hệ

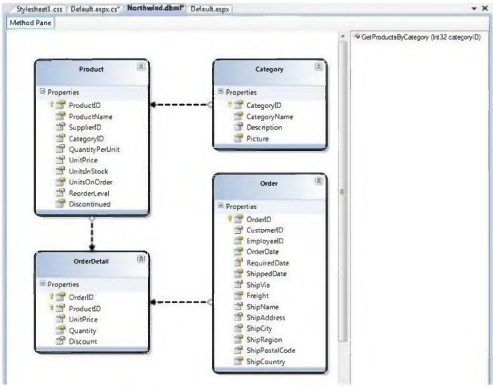


## Các Khái niệm trong LINQ to SQL:

### Mô hình hóa CSDL dùng LINQ to SQL(dùng thiết kế đối tượng quan hệ):

Như đã được giới thiệu ở mục VIII, từ phiên bản 2008 Visual studio đã tích hợp thêm một trình thiết kế LINQ to SQL như một công cụ dễ dàng cho việc mô hình hóa một cách trực quan các CSDL dùng LINQ to SQL.

Bằng cách dùng trình thiết kế LINQ to SQL, tôi có thề dễ dàng tạo một mô hình cho CSDL mẫu “Northwind” giống như dưới đây:



Mô hình LINQ to SQL ở trên định nghĩa bốn lớp thực thế: Product, Category, Order và OrderDetail. Các thuộc tính của mỗi lớp ánh xạ vào các cột của bảng tương ứng trong CSDL. Mỗi instance của một lớp biểu diễn một dòng trong bảng dữ liệu.

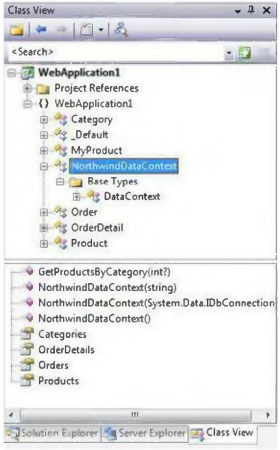
Các mũi tên giữa bốn lớp thực thể trên biểu diễn quan hệ giữa các thực thế khác nhau, chúng được tạo ra dựa trên các mối quan hệ primary-key/foreign-key trong CSDL. Hướng của mũi tên chỉ ra mối quan hệ là một - một hay một - nhiều. Các thuộc tính tương ứng sẽ được thêm vào các lóp thực thể trong các trường hợp này. Lấy ví dụ, lớp Category ở trên có một mối quan hệ một nhiều với lớp Product, điều này có nghĩa nó sẽ có một thuộc tính “Categories” là một tập hợp các đối tượng Product trong Category này. Lớp Product cũng sẽ có một thuộc tính “Category” chỉ đến đối tượng “Category” chứa Product này bên trong.

Bảng các phương thức bên tay phái bên trong trình thiết kế LINQ to SQL ớ trên chứa một danh sách các SP để tương tác với mô hình dữ liệu của chúng ta. Trong ví dụ trên tôi đã thêm một thủ tục có tên “GetProductsByCategory”. Nó nhận vào một categorylD và trả về một chuồi các Product. Chúng ta sẽ xem bằng cách nào có thế gọi được thủ tục này trong các đoạn code bên dưới.

### Lớp DataContext

Khi bạn bấm nút “Save” bên trong màn hình thiết kế LINQ to SQL, Visual Studio sẽ lưu các lớp .NET biểu diễn các thực thế và quan hệ bên trong CSDL mà chúng ta vừa mô hình hóa. Cứ mồi một file LINQ to SQL chúng ta thêm vào solution, một lớp DataContext sẽ được tạo ra, nó sẽ được dùng khi cần truy vấn hay cập nhật lại các thay đối. Lớp DataContext được tạo sẽ có các thuộc tính để biểu diễn mối bảng được mô hình hóa từ CSDL, cũng như các phương thức cho mỗi SP mà chúng ta đã thêm vào.

Lấy ví dụ, dưới đây là lớp NorthwindDataContext được sinh ra dựa trên mô hình chúng ta tạo ra ở trên:

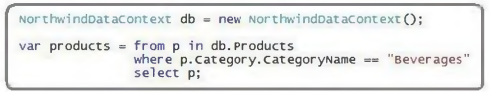


### Các câu lệnh làm việc trong LINQ to SQL:

Một khi đã mô hình hóa CSDL dùng trình thiết kế LINQ to SQL, chúng ta có thế dễ dàng viết các đoạn lệnh đế làm việc với nó. Dưới đây là một vài ví dụ về các thao tác chung khi xử lý dữ liệu:

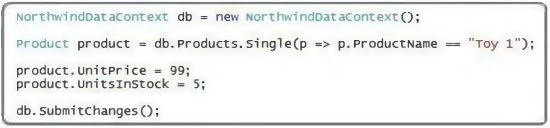
1. Truy vấn dữ liệu từ CSDL:

Đoạn lệnh dưới đây dùng cú pháp LINQ để lấy về một tập Enum erable các đối tượng Product. Các sán phấm được lấy ra phải thuộc phân loại “Beverages”:



1. Cập nhật dữ liệu vào CSDL:

Đoạn lệnh dưới đây cho thấy cách lấy một sản phấm, cập nhật lại giá tiền và lưu lại CSDL.



1. Chèn thêm dữ liệu mới vào CSDL:

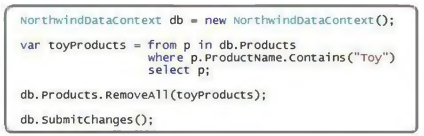
Đoạn code dưới đây biểu diễn cách tạo một phân loại mới, và tạo hai sản phấm mới và đưa chúng vào trong phân loại đã tạo. Cá ba sau đó sẽ được đưa vào cơ sở dữ liệu. Chú ỷ rằng tôi không cần phái tự quán lý các mối quan hệ primary key/foreign key, thay vào đó, tôi chỉ đơn giản thêm các đối tượng Product vào tập hợp Products của đối tượng category, và rồi thêm đối tượng category vào tập hợp Categories của DataContext, LINQ to SQL sẽ biết cách thiết lập các giá trị primary key/foreign key một cách thích hợp.

(Add đã được thay đối bằng InsertOnSubmit trong phiên bản hiện tại)



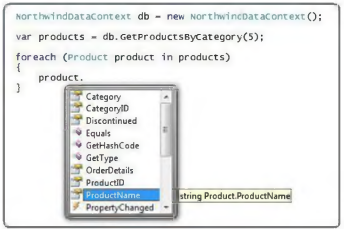
1. Xóa dữ liệu ở CSDL:

Đoạn code sau đây sẽ biểu diễn cách xóa tất cả các sản phẩm Ty ra khỏi CSDL: (RemoveAll đã được thay đổi bằng DeleteAllOnSubmit trong phiên bản hiện tại)



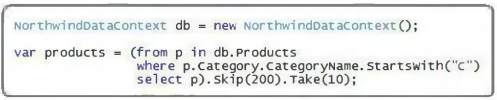
1. Gọi một thủ tục:

Đoạn code dưới đây biểu diễn cách lấy các thực thể Product mà không dùng cú pháp gọi đển thủ tục “GetProductsByCategory” chúng ta đã thêm vào trước đó. Một khi đã lấy được kết quả, chúng ta có thể cập nhật/xóa và sau đó gọi db.SubmitChanges() để cập nhật các thay đổi trở lại CSDL.



1. Thực hiện truy vấn có phân trang:

Đoạn code dưới đây biểu diễn cách phân trang trên server như một phần của câu truy vấn LINQ. Bằng cách dùng các toán tử Skipo và Take(), chúng ta sẽ chỉ trả về 10 dòng tò CSDL - bắt đầu từ dòng 200.



1. Tự tạo một database với DataContext.CreateDatabase:

Trong LINQ to SQL, các đối tượng được ánh xạ vào một CSDL quan hệ. Sự ánh xạ được kích hoạt bằng cách ánh xạ các đặc tính tượng trưng hoặc sử dụng các tệp ánh xạ ngoài để mô tả cấu trúc của CSDL quan hệ. Cả 2 phương pháp này đều sẽ cung cấp đầy đủ thông tin về CSDL quan hệ để ta có thể tạo ra một phiên bản của CSDL sử dụng phương thức DataContext.CreateDatabase.

Phương thức DataContext.CreateDatabase tạo một bản sao của cơ sở dữ liệu nằm trong phạm vi thông tin được mã hóa trong mô hình đối tượng. Ánh xạ thuộc tính và phương thức tuy nhiên có thể sẽ không mã hóa mọi thứ về cấu trúc của cơ sở dữ liệu hiện có. Thông tin ánh xạ sẽ không thể bao gồm nội dung của các hàm do người dùng xác định, các hàm, thủ tục lưu trữ, trigger hoặc khả năng kiểm tra các ràng buộc.

Chúng ta có thể sử dụng phương thức DataContext.CreateDatabase trong mọi trường hợp, đặc biệt là nếu phía cung cấp dữ liệu sử dụng Microsoft SQL Server 2008 trở lên. Các lý do điển hình bao gồm:

* Chúng ta đang xây dựng một ứng dụng có thể tự động cài đặt trên hệ thống của khách hàng.
* Chúng ta đang xây dựng một ứng dụng khách (Client application) cần cơ sở dữ liệu cục bộ để lưu các trạng thái ngoại tuyến của nó.

Bạn cũng có thể sử dụng phương thức DataContext.CreateDatabase với SQL Server bằng cách sử dụng tệp .mdf hoặc tên thay thế, tùy theo chuỗi kết nối của bạn. LINQ to SQL sử dụng chuỗi kết nối để xác định CSDL nguồn và CSDL nào được tạo.

Ví dụ: tạo ra một cơ sở dữ liệu mới có tên MyDVDs.

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;

}

Chúng ta cũng có thể sử dụng mô hình đối tượng để tạo CSDL như sau:

public void CreateDatabase()

{

MyDVDs db = new MyDVDs("c:\\mydvds.mdf");

db.CreateDatabase();

}

Khi ta xây dựng một ứng dụng có thể tự động cài đặt lên hệ thống của khách hàng, cần phải kiểm tra xem liệu CSDL của ứng dụng đã tồn tại hay chưa và xóa nó trước khi tạo một CSDL mới. Lớp DataContext cung cấp các phương thức DatabaseExists và DeleteDatabase để giúp chúng ta trong quá trình này.

public void CreateDatabase2()

{

MyDVDs db = new MyDVDs(@"c:\mydvds.mdf");

if (db.DatabaseExists())

{

Console.WriteLine("Deleting old database...");

db.DeleteDatabase();

}

db.CreateDatabase();

}

## Xây dựng chương trình với LINQ to SQL:

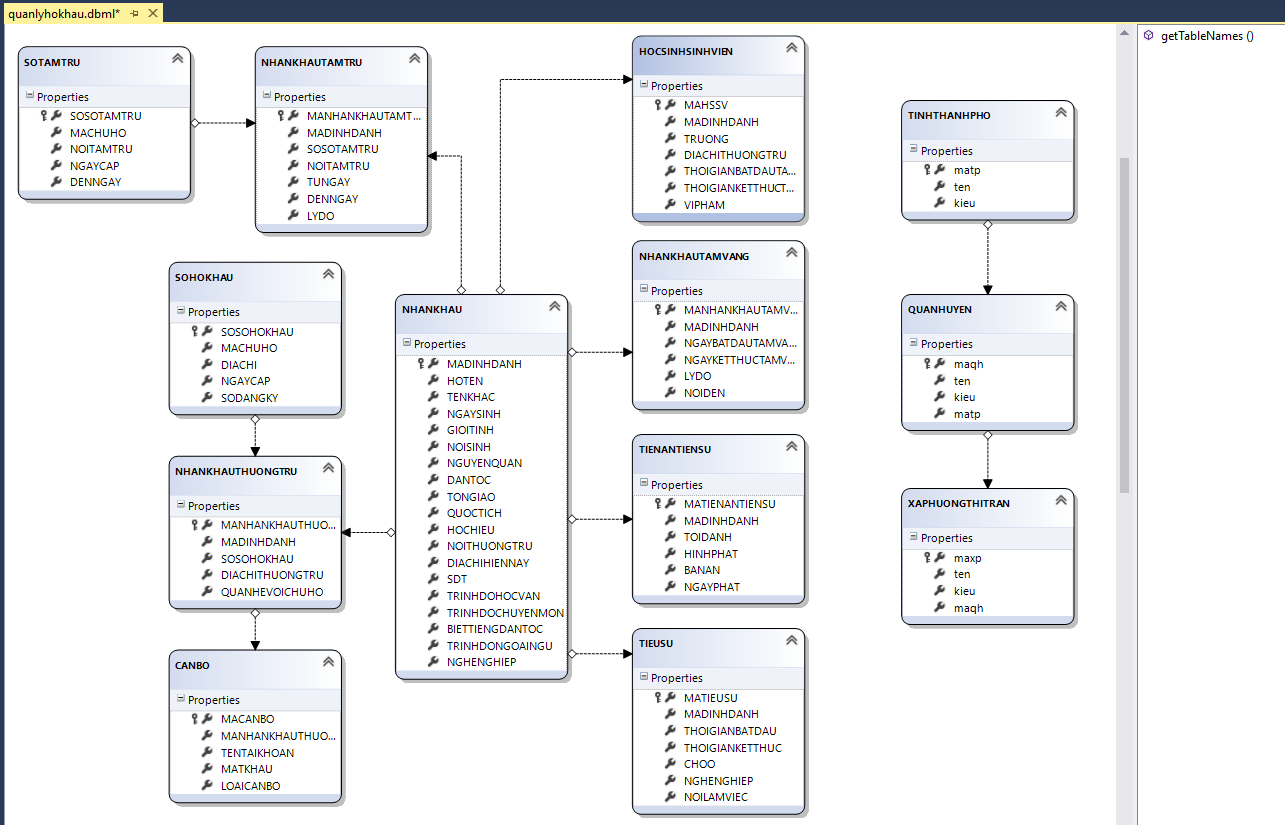
### Định nghĩa các lớp mô hình dữ liệu:

Trong phần mở đầu, chúng ta đã thảo luận về “LINQ to SQL là gì” và cung cấp một cái nhìn cơ bản về những trường hợp chúng ta có thể sử dụng nó.

Trong phần V.2 cũng đã cung cấp các đoạn code mẫu để biểu diễn cách xử lý dữ liệu dùng LINQ to SQL, bao gồm:

* Cách truy vấn dữ liệu
* Các cập nhật dữ liệu
* Cách chèn và tạo quan hệ các dòng trong một CSDL
* Cách xóa các dòng trong một CSDL
* Cách gọi một thủ tục
* Cách lấy dừ liệu và phân trang trên server

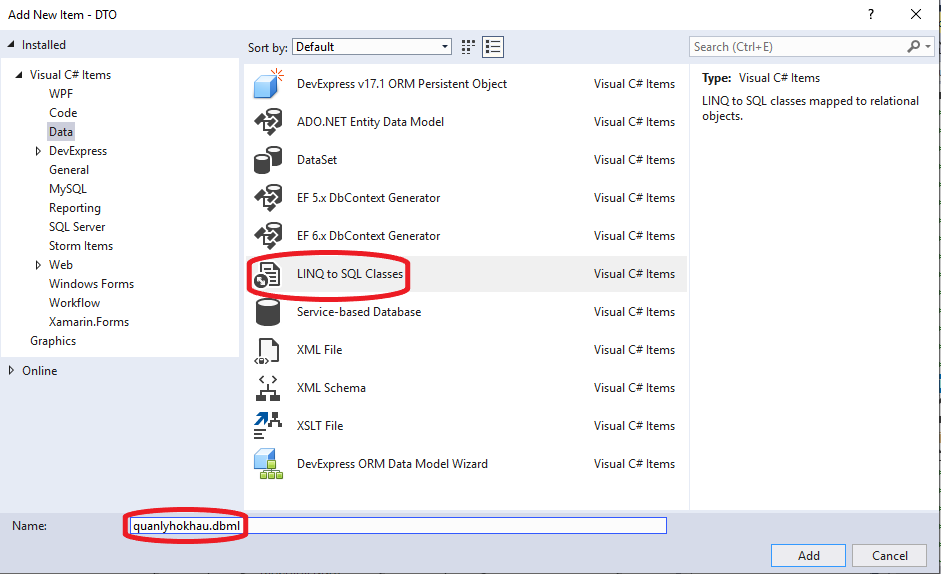
Chúng ta đã thực hiện tất cả các thao tác dữ liệu đó bằng cách dùng một mô hình dữ liệu LINQ to SQL giống như dưới đây:



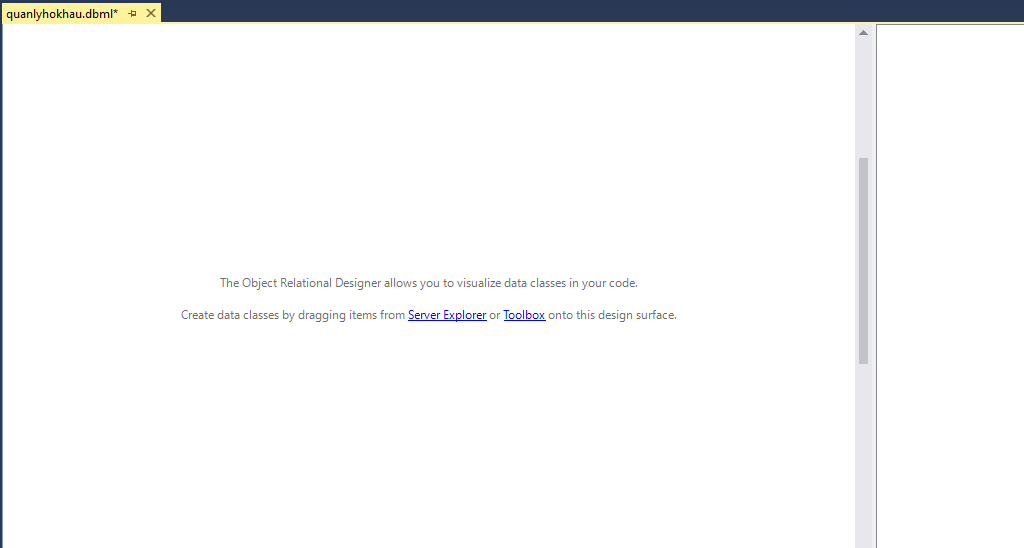
Trong bài này, Chúng sẽ đi vào chi tiết cách tạo ra một mô hình dữ liệu LINQ to SQL giống như trên. Lưu ý: kể từ phần này chúng ta sẽ làm việc với project “Quản lý hộ khẩu” với cơ sở dữ liệu “qlhk” xây dựng trên .NET Framework 4.7 và Visual Studio 2017

1. Tạo ra một mô hình dữ liệu LINQ to SQL:

Chúng ta có thêm một mô hình dữ liệu LINQ to SQL vào một project C# WinForm (hoặc ASP.NET) bằng cách dùng tùy chọn “Add New Item” bên trong Visual Studio và chọn “LINQ to SQL Classes”:



Việc chọn mục “LINQ to SQL” sẽ khởi chạy LINQ to SQL designer (O/R Designer), và cho phép ta mô hình hóa các lớp mà nó biếu diễn một CSDL quan hệ. Nó cũng sẽ tạo ra một lớp kiểu “DataContext”, trong đó có các thuộc tính đế biểu diễn mồi bảng mà chúng ta mô hình hóa trong CSDL, cũng như các phương thức cho mỗi Stored Procedure mà chúng ta mô hình hóa. Như đã được mô tả trong phần V.1.1, 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. Dưới đây là ảnh chụp màn hình của một của số thiết kế LINQ to SQL, và cũng là cái mà ta sẽ thấy ngay khi tạo ra một mô hình dừ liệu LINQ to SQL:



1. Các lớp thực thể:

LINQ to SQL cho phép ta mô hình hóa các lớp ánh xạ từ CSDL. Các lớp này thường được gọi 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 đối tượng CSDL. Các thuộc tính của các lớp thông thường ánh xạ vào các cột trong bảng. Mỗi instance của một lớp thực thể biếu diễn một dòng trong bảng.

Các lớp thực thế trong LINQ to SQL không cần phải kế thừa từ một lớp đặc biệt nào khác, điều đó cho phép ta có thể cho chúng thừa kế từ bất cứ đối tượng nào mình muốn. Tất cả các lớp được tạo ra dùng O/R Designer đều được định nghĩa như “partial class'’ - có nghĩa là ta có thể viết code để thêm các thuộc tính, phương thức và sự kiên cho chúng.

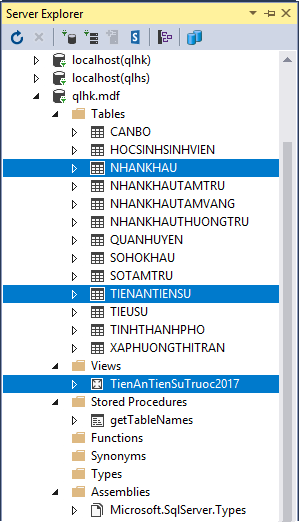
Không giống như chức năng DataSet/TableAdapter có trong vs 2005, khi dùng O/R Designer, chúng ta không cần chỉ ra câu truy vấn SQL được dùng đế tạo ra mô hình và lớp truy xuất dữ liệu.

Thay vào đó, ta tập trung chủ yếu vào việc định nghĩa các lớp thực thể, cách chúng ánh xạ vào CSDL, và mối quan hệ giữa chúng. Trình LINQ to SQL cụ thể mà chúng ta dùng sẽ đảm bảo việc sinh ra các lệnh SQL thích hợp vào lúc khởi chạy khi ta tương tác và làm việc với các thực thể dữ liệu. Trong thực tế, chúng ta vẫn có thế dùng cú pháp truy vấn LINQ để chỉ ra cách mình muốn truy vấn dữ liệu.

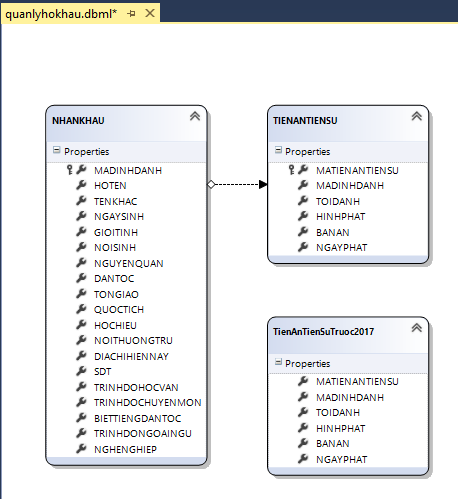
1. Tạo các lớp thực thể từ CSDL:

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

Cách 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 chúng ta muốn mô hình hóa, và kéo thả chúng lên trên của số O/R Designer.



Khi chúng ta thêm 2 bảng (NHANKHAU và TIENANTIENSU) và 1 view (TienAnTienSuTruoc2017) từ CSDL “qlhk” vào cửa số O/R Designer, ta sẽ có thêm 3 lớp thực thể được tạo ra một cách tự động:

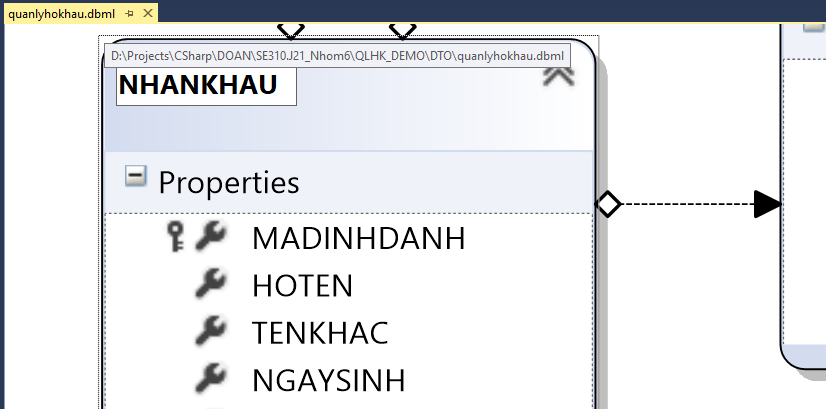


Dùng các lớp mô hình hóa dữ liệu ở trên, ta có thể chạy tất cả các đoạn lệnh mẫu đã một tả trong phần V.1.3. Chúng ta có thể thực hiện được các thao tác query, insert, update, delete và phân trang mà không cần thêm bất kì đoạn code nào khác.

1. Cách đặt tên và ngữ pháp số nhiều:

LINQ to SQL có khả năng tự động chuyển tên bảng và cột thành dạng số nhiều khi tạo các lớp thực thể. Lấy ví dụ: Bảng “NHANKHAU” trong ví dụ của chúng ta tạo ra lớp “NHANKHAUs”, cũng như bảng “TIENANTIENSU” tạo ra lóp “TIENANTIENSUs”. Cách đặt tên này giúp mô hình thống nhất với quy ước đặt tên trong .NET.

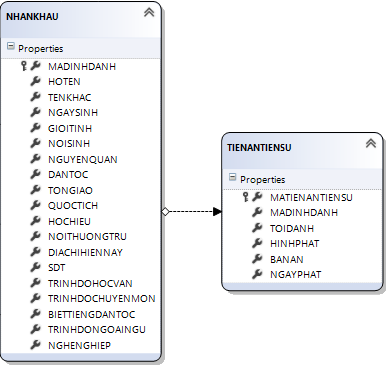
Nếu không thích tên lớp hay tên thuộc tính do trình designer sinh ra, ta vẫn có thể sửa lại thành bất cứ tên nào mình thích. Bạn có thể làm điều này bằng cách chỉnh sửa tên thực thể/thuộc tính bên trong trình thiết kế hoặc thông qua bảng thuộc tính.



Khả năng đặt tên cho các thực thể/thuộc tính/quan hệ khác với tên trong CSDL rất hữu dụng trong một số trường hợp, ví dụ:

1. Khi tên bảng/cột trong CSDL bị thay đối. Bởi vì mô hình thực thể có thể có tên khác với tên trong CSDL, do vậy ta chỉ cần cập nhật lại các quy tắc ánh xạ mà không cần cập nhật chương trình hoặc các lệnh truy vấn để có thể dùng được tên mới.
2. Khi các thành phần bên trong CSDL được đặt tên không rõ ràng. Ví dụ: thay vì dùng “au\_lname” và “au\_fname” cho các tên thuộc tính của một lớp thực thể, ta có thế đặt tên chúng thành “LastName” và “FirstName” trong lớp thực thế và viết các lệnh để dùng với nó (mà không cần đổi tên các cột trong CSDL).
3. Quan hệ giữa các thực thể:

Khi ta kéo thả các đối tượng từ Server Explorer lên trên cửa sổ O/R 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 ta thêm cả hai bảng NHANKHAU và TIENANTIENSU từ qlhk lên trên cửa số LINQ to SQL, ta có thể thấy mối quan hệ một nhiều giữa chúng (được biểu diễn bằng một mũi tên trên của sổ designer):



Mối quan hệ trên sẽ làm lớp thực thể TIENANTIENSU có thêm một thuộc tính là NHANKHAU, mà chúng ta có thể dùng để truy cập vào danh sách thực thể NHANKHAU của một TIENANTIENSU. Nó cũng làm lớp NHANKHAU có thêm thuộc tính “TIENANTIENSUs”, đây là một tập hợp cho phép ta lấy ra tất cả các TIENANTIENSU có trong NHANKHAU đó.

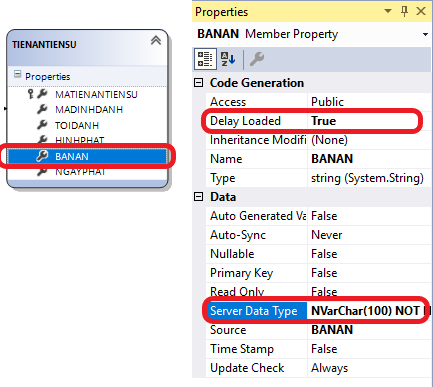


Nếu không thích cách mà trình thiết kế đã mô hình hóa hoặc đặt tên, chúng ta hoàn toàn có thế chỉnh sửa lại. Chỉ cần click lên mũi tên chỉ ra quan hệ trên của số soạn thảo và truy cập vào các thuộc tính của nó thông qua bảng thuộc tính để đổi tên, chỉnh sửa hoặc thậm chí xóa nó.

1. Delay/Lazy loading:

LINQ to SQL cho phép chỉ ra các thuộc tính của một thực thể sẽ được lấy về trước(prefetch) hay chỉ được lấy khi người dùng lần đầu truy cập (gọi là delaỵ/lazy loading). Chúng ta có thế tùy biến các quy tắc prefetch/lazy load cho các thuộc tính trong thực thể bằng cách chọn thuộc tính hay quan hệ đó, và đặt lại giá trị cho thuộc tính “Delay Loaded” thành true hoặc false.

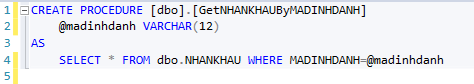
Ví dụ, cấu hình thuộc tính BANAN để nó chỉ được nạp khi dùng đến bằng cách đặt thuộc tính Delay Loaded thành true:



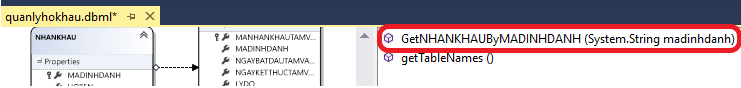
Ghi chú: Thay vì cấu hình prefetch/delay load trên các thực thế, chúng ta cũng có thế đặt lại thông qua các lệnh khi thực hiện các câu truy vấn LINQ trên lớp thực thể đó.

1. Dùng các StoredProcedure:

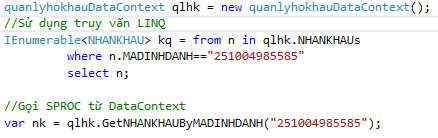
LINQ to SQL cho phép ta có thế mô hình hóa các thủ tục lưu trữ như là các phương thức trong lớp DataContext. Ví dụ, cho rằng chúng ta đã định nghĩa một thủ tục đơn giản có tên SPROC như dưới đây để lấy về các thông tin nhân khẩu dựa trên một mã định danh:



Dùng Server Explorer trong VS để kéo/thả thủ tục SPROC lên trên cửa sổ soạn thảo LINQ to SQL để có thể thêm một phương thức cho phép gọi SPROC. Nếu ta thả SPROC lên trên thực thể “NHANKHAU”, O/R Designer sẽ khai báo SPROC để trả về một tập kết quả có kiểu IEnumerable:



Sau đó ta có thể dùng cú pháp LINQ to SQL hoặc gọi thẳng phương thức này để lấy dữ liệu từ CSDL:



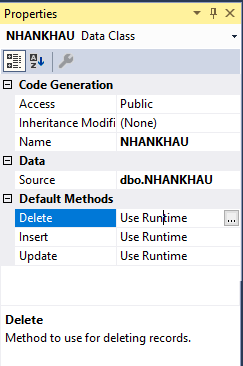
1. Dùng SPROCS để cập nhật/xóa, thêm dữ liệu:

Mặc định LINQ to SQL sẽ tự động tạo ra các biếu thức SQL phù hợp mỗi khi chúng ta muốn cập nhật/xóa/thêm dữ liệu. Ví dụ, nếu bạn viết mã LINQ to SQL như dưới đây để cập nhật một số giá trị trên một thực thể “Product”:



Mặc nhiên, LINQ to SQL sẽ tạo và thực thi lệnh UPDATE tương ứng khi ta xác nhận thay đối (SubmitChanges).

Chúng ta cũng có thể định nghĩa và dùng các thủ tục INSERT, UPDATE, DELETE nếu muốn. Để làm điều đó, ta cần click vào một lớp thực thể trong cửa sổ O/R Designer và trong bảng thuộc tính, nhấn chuột vào nút “ ...” tìm các giá trị Delete/Insert/Update, và chọn SPROC mà ta đã định nghĩa.



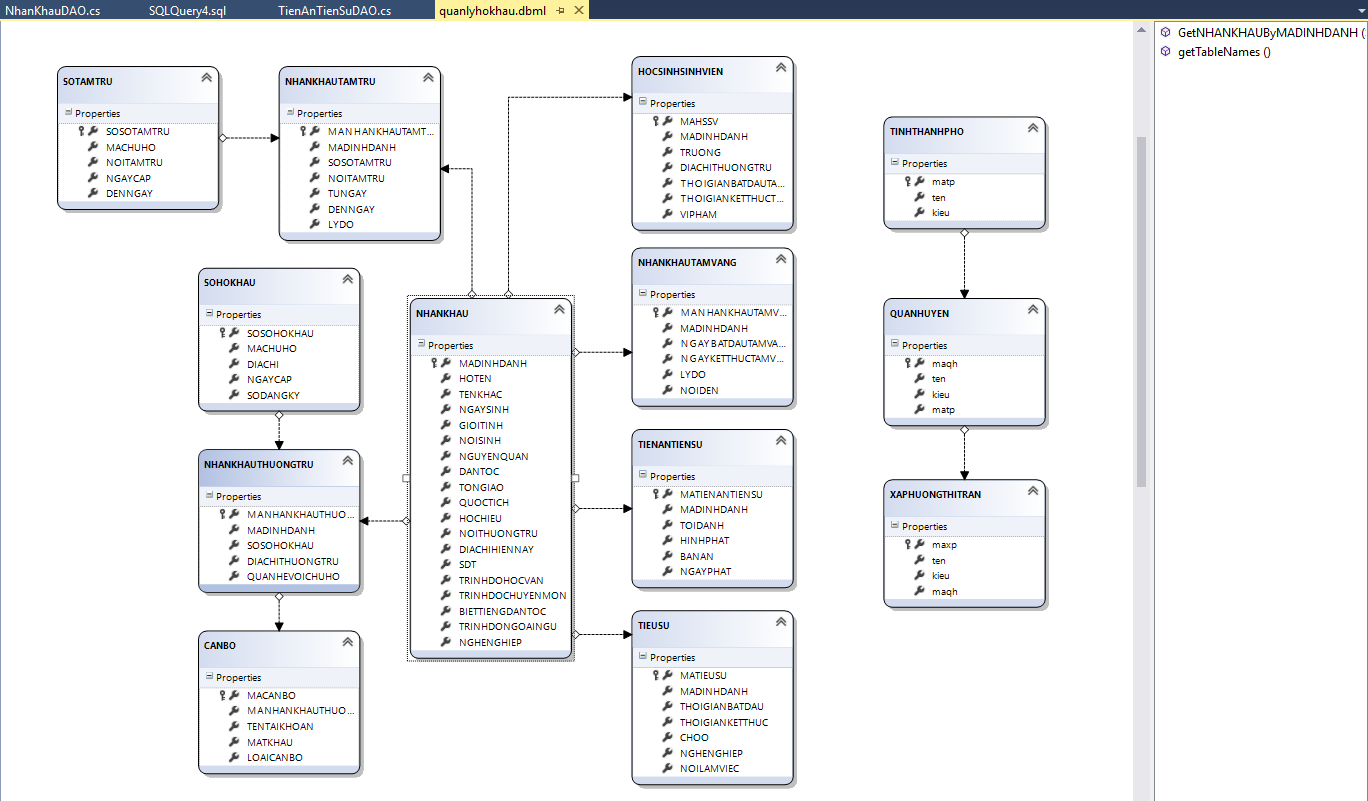
Có một điều thú vị là những thay đổi trên hoàn toàn được thực hiện ở lớp ánh xạ LINQ to SQL - Có nghĩa là tất cả những đoạn code ta đã viết trước đó đều tiếp tục làm việc mà không cần thay đổi bất cứ điều gì. Điều này giúp ta tránh phải thay đổi lại code ngay cả nếu sau này muốn dùng một hàm SPROC để tối ưu thêm/xóa/sửa.

### Truy vấn Cơ sở dữ liệu:

LINQ to SQL là một bộ khung (Framework) có sẵn cho ORM. Nó cho phép chúng ta dễ dàng mô hình hóa các CSQL quan hệ dùng các lớp .NET. Để từ đó, chúng ta có thể dễ dàng truy vấn dữ liệu sử dụng các biểu thức LINQ cũng như có thể Thêm/Sửa/Xóa dữ liệu từ CSDL.

1. Mô hình hóa CSDL QLHK dùng LINQ to SQL:

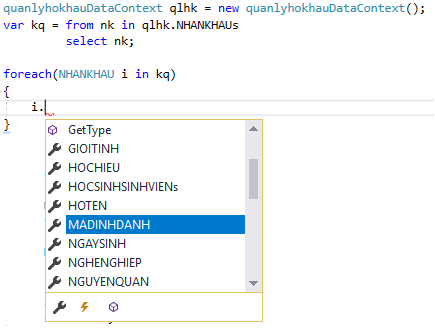
Trong phần V.2.1, chúng ta đã đi qua các bước để tạo một mô hình các lớp LINQ to SQL bằng cách dùng trình LINQ to SQL classes có sẵn trong VS. Dưới đây là mô hình CSDL mẫu qlhk:



1. Lấy các NHANKHAU:

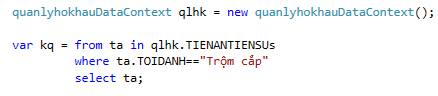
Một khi đã định nghĩa mô hình dữ liệu như trên, chúng ta có thể dễ dàng truy vấn và lấy dữ liệu từ CSDL. LINQ to SQL cho phép làm điều này bằng cách viết các câu truy vấn dùng các cú pháp và biểu thức LINQ với lớp quanlyhokhauDataContext mà chúng ta đã tạo.

Ví dụ, để lấy và duyệt một tập các đối tượng NHANKHAU, ta có thể viết code như sau:

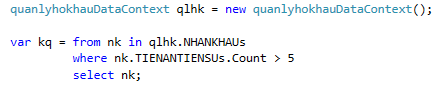


Trong câu truy vấn trên, mệnh đề “where” đã được dùng trong cú pháp LINQ đế xác đỉnh các tiền án trong một NHANKHAU cho trước. Sử dụng MADINHDANH của NHANKHAU để thực hiện lọc ra các dữ liệu mong muốn.

Một trong những điểm hay là ta có rất nhiều lựa chọn, rất nhiều cách để tùy biến câu lệnh, và ta có thể nắm bắt ưu điểm của mối quan hệ giữa các thực thể mà chúng ta đã tạo khi mô hình hóa các lớp để làm cho câu lệnh phong phú và tự nhiên hơn. Ví dụ, chúng ta có thể sửa lại câu truy vấn để lọc ra các dòng theo HOTEN thay vì MADINHDANH bằng cách viết câu lệnh LINQ như sau:



Chú ý thuộc tính “NHANKHAU” trên mỗi đối tượng TIENANTIENSU được dùng để lọc theo HOTEN của NHANKHAU chứa TIENANTIENSU đó. Thuộc tính này được tự động tạo ra bởi LINQ to SQL vì chúng ta đã mô hình hóa các lớp NHANKHAU và TIENANTIENSU như một mối quan hệ một-nhiều (quan hệ với khóa MADINHDANH). Một ví dụ khác về cách dùng quan hệ trong mô hình dữ liệu bên trong các câu truy vấn, chúng ta có thể viết câu lệnh LINQ như dưới đây đế lấy về chỉ những NHANKHAU có 5 hoặc hơn TIENANTIENSU:

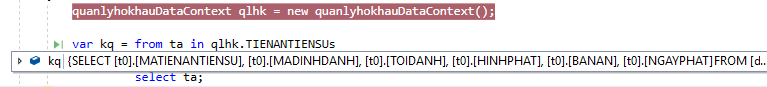


Chú ý cách chúng ta đã dùng tập hợp “TIENANTIENSUs” mà LINQ to SQL đã tạo trên mỗi lớp NHANKHAU (nhờ vào mối quan hệ một-nhiều mà chúng ta đã mô hình hóa trước đó)

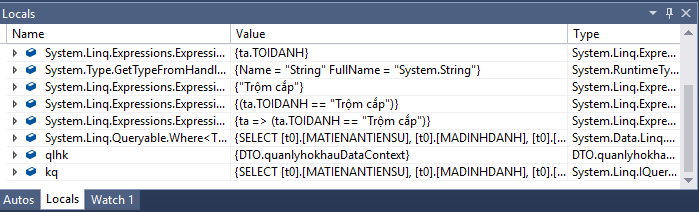
1. Trực quan hóa các câu truy vấn LINQ to SQL trong trình gỡ lỗi:

Các trình ORM như O/R Designer trong LINQ to SQL tạo ra và thực thi các câu lệnh SQL một cách tự động mỗi khi ta thực hiện một câu truy vấn hay cập nhật mô hình đối tượng của nó.

Vậy, “Câu lệnh SQL thực sự được thực thi là gì?”. Một điều thực sự thú vị về LINQ to SQL là nó cho phép ta xem câu lệnh SQL thực sự được thực thi rất dễ dàng khi ta chạy ứng dụng trong chế độ debug. Bắt đầu từ bản Beta2 của VS 2008, chúng ta có thể dùng một LINQ to SQL visualizer plug-in đế xem (và kiểm tra) bất kỳ câu lệnh truy vấn LINQ to SQL nào. Chỉ cần đặt một breakpoint và di chuột lên trên một câu lệnh LINQ to SQL, sau đó nhấn vào biếu tượng chiếc kính lúp để xem giá trị của câu lệnh một cách trực quan:



Một cửa sổ debug cho phép ta xem một cách chính xác câu lệnh LINQ to SQL mà LINQ to SQL sẽ đùng để lấy về các đối tượng TIENANTIENSU:



Ta có thể dễ dàng copy câu truy vấn trên cửa sổ này, sau đó chạy câu lệnh SQL trực tiếp trong trình hỗ trợ truy vấn SQL của VS và xem một cách chính xác dữ liệu được trả về:



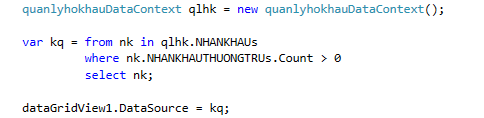
Điều này rõ ràng làm cho việc xem những gì LINQ to SQL thực hiện trở nên cực kỳ dễ dàng. Thậm chí, chúng ta có thể thay thế câu SQL mà LINQ to SQL thực thi nếu muốn - mặc dù đễ thấy rằng câu lệnh mặc định mà LINQ to SQL thực thi là thực sự tốt.

1. Ghép nối các câu truy vấn LINQ to SQL vào các control C#:

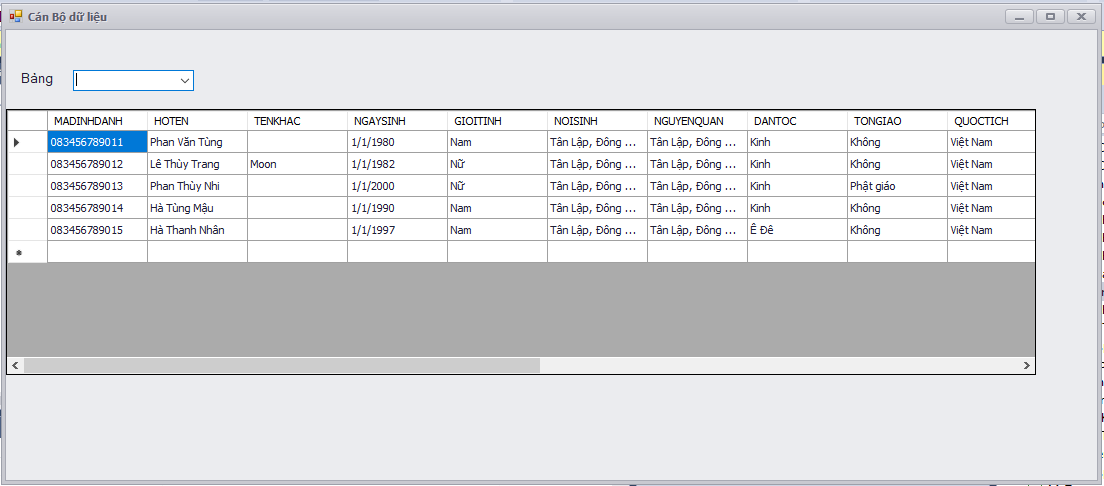
Các câu truy vấn LINQ trả về kết quả mà nó sẽ implement interrface IEnumerable - đây cũng là interface mà các control trong C# dùng để hỗ trợ gắn nối các đối tượng. Điều này có nghĩa là chúng ta có thể gắn nối kết quả của bất kỳ câu lệnh LINQ, LINQ to SQL hay LINỌ to XML vào bất kỳ control C#.NET nào. Lấy ví dụ, ta có thế khai báo một control trong một Form C# giống như sau:



Chúng ta cũng có thể gắn nối kết quả của câu LINQ to SQL đã viết trước đây trực tiếp vào DataGridView giống như sau:

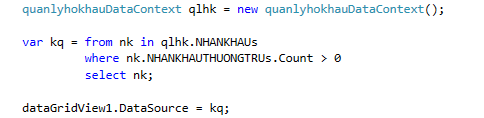


Nó sẽ sinh ra một màn hình trông như sau:

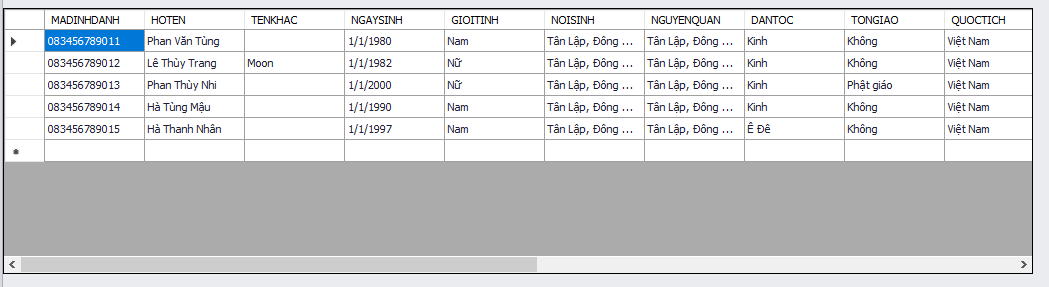


1. Data Sharping:

Hiện tại, mỗi khi xác định kết quả truy vấn, chúng ta lấy toàn bộ các cột dữ liệu cần thiết cho các đối tượng thuộc lớp NHANKHAU. Ví dụ, câu truy vấn sau lấy về các nhân khẩu thường trú:



Và toàn bộ kết quả được trả về:



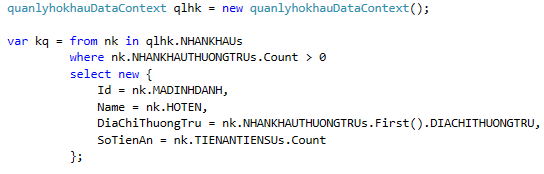
Thường thi chúng ta chỉ muốn trả về một tập con của dữ liệu về mỗi sản phẩm. Chúng ta có thế dùng tính năng data shaping mà LINQ và các C# compiler hỗ trợ để chỉ ra rằng chúng ta chỉ muốn một tập con bằng cách chỉnh sửa lại câu truy vấn như sau:



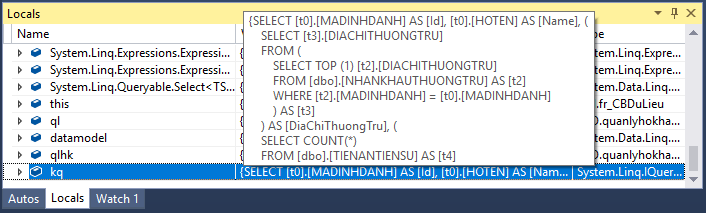
Điều này sẽ trả về chỉ một tập con dữ liệu được trả về từ CSDL:



Một điều thực sự thú vị về LINQ to SQL là ta có thể tận dụng tất cả ưu điểm của các quan hệ trong mô hình dữ liệu khi muốn gọt giũa lại dữ liệu. Nó cho phép chúng ta biếu diễn đầy đủ và hiệu quả các câu truy vấn. Lấy ví dụ, câu truy vấn dưới đây lấy về Id và Name từ thực thể NHANKHAU, địa chỉ thường trú của nhân khẩu đó, và rồi lấy tống số Tiền án của nhân khẩu:



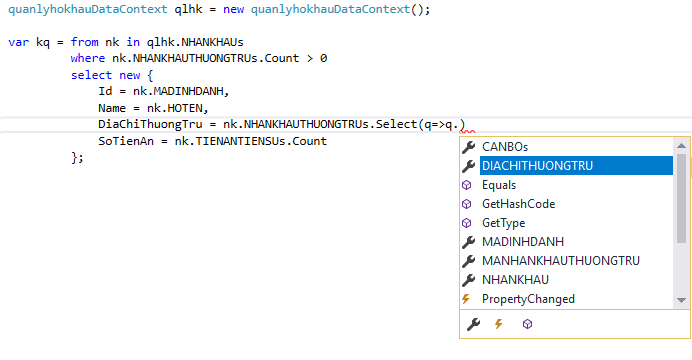
LINQ to SQL đủ thông minh đế có thể chuyến biểu thức LINQ ở trên thành câu SQL dưới đây khi nó được thực thi:



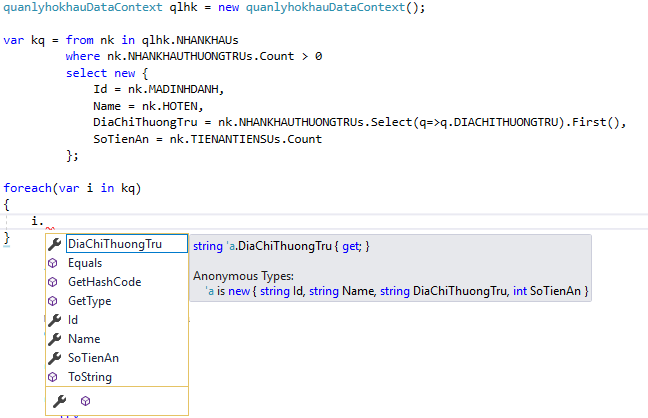
Câu SQL ở trên cho phép Lấy thông tin DiaChiThuongTru và tính số TIENANTIENSU từ ngay trên SQL server, và trả về chỉ những dữ liệu như dưới đây (làm cho việc thực thi được nhanh chóng):



Chúng ta cũng có thể được hồ trợ đầy đủ bởi tính năng intellisense bên trong VS khi viết các câu truy vấn LINQ:



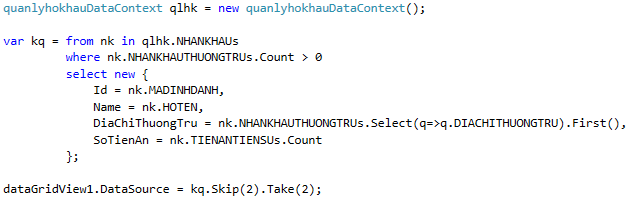
Trong ví dụ trên, ta đang sử dụng một biểu thức Lambda (Lambda Expression) và dùng object initialization đề gọt giũa và định nghĩa cấu trúc trả về. Một điều thực sự tuyệt vời là VS cung cấp intellisense đầy đủ, kiểm tra lúc dịch và cả refactoring khi làm việc cả với các tập kết quả thuộc kiểu vô danh (Anonymous Type):



1. Phân trang kết quả truy vấn:

Một trong những yêu cầu cơ bản khi xây dựng một bảng ghi đó là bảng ghi phải có khả năng phân trang một cách hiệu quả. LINQ cung cấp sẳn hai hàm mở rộng cho phép ta có thể làm điều đó - hàm Skip() và Take().

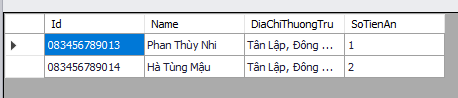
Chúng ta có thể dùng Skip() và Take() để chỉ ra trang và số phần tử muốn lấy. Xét ví dụ, khi chúng ta muốn lấy 2 đối tượng nhân khẩu bắt đầu từ một danh sách nhân khẩu cho trước mà chúng ta chỉ ra trong tham số truyền vào:



Chú ý ở trên chúng ta đã không dùng Skip() va Take() trong câu khai báo truy vấn các sản phẩm - mà chỉ sử dụng khi gắn kết dữ liệu vào GridView. Vậy, “Có phải làm như vậy thì câu lệnh đầu tiên sẽ lấy toàn bộ CSDL về, rồi sau đó mới thực hiện việc phân trang?”. Câu trả lời là “Không”. Lý do là vì LINQ chỉ thực thi các câu lệnh truy vấn khi chúng ta lấy kết quả từ nó mà thôi (Lazy Loading).

Một trong những ưu điếm của mô hình này là nó cho phép ta có thể viết các câu lệnh phức tạp bằng nhiều bước, thay vì phải viết trong một câu lệnh đơn (giúp dễ đọc hơn). Nó cũng cho phép ta tạo ra các câu truy vấn từ các câu khác, giúp xây dựng các câu truy vấn rất phức tạp cũng như có thể dùng lại được các câu truy vấn khác.

Nó sẽ cho chúng ta một trang hiển thị các nhân khẩu có thông tin thường trú và số tiền án tiền sự bắt đầu từ nhân khẩu thứ 3(bỏ qua từ vị trí 2 trở về trước) và hiển thị 2 nhân khẩu:



### Cập nhật Cơ sở dữ liệu:

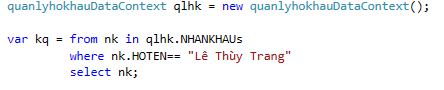
1. CSDL qlhk được mô hình hóa dùng LINQ to SQL:



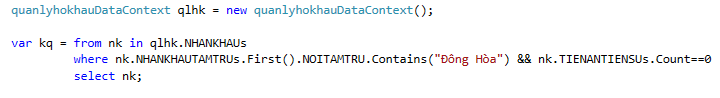
Khi chúng ta định nghĩa mô hình dữ liệu dùng O/R Designer như trên, chúng ta đã định nghĩa ra 13 lớp mô hình: CANBO, HOCSINHSINHVIEN, NHANKHAU, NHANKHAUTAMTRU, NHANKHAUTAMVANG, NHANKHAUTHUONGTRU, QUANHUYEN, SOHOKHAU, SOTAMTRU, TIENANTIENSU, TIEUSU, TINHTHANHPHO và XAPHUONGTHITRAN. Các thuộc tính của mỗi lớp ánh xạ vào các cột tương ứng trong bảng dữ liệu. Mỗi đối tượng thuộc lớp thực thế sẽ biếu diễn một dòng trong bảng CSDL.

Khi định nghĩa mô hình dữ liệu, O/R Designer cũng tạo ra một lớp DataContext cung cấp các cách thức để truy vấn và cập nhật lại dữ liệu. Trong mô hình mẫu chúng ta đã định nghĩa ở trên, lớp này được đặt tên là “quanlyhokhauDataContext”. Lớp quanlyhokhauDataContext có các thuộc tính biếu diễn các bảng chúng ta đã định nghĩa trong CSDL (CANBO, HOCSINHSINHVIEN, NHANKHAU, NHANKHAUTAMTRU, NHANKHAUTAMVANG, …).

Như đã nói trong phần V2.2, chúng ta có thể dễ dàng dùng các biểu thức LINQ để truy vấn và lấy dữ liệu từ CSDL bằng cách dùng lớp quanlyhokhauDataContext. LINQ to SQL sau đó sẽ tự động diễn dịch các biểu thức đó thành các câu lệnh SQL thích hợp để thực thi. Ví dụ, chúng ta có thể viết biểu thức LINQ như dưới đây để lấy về một đối tượng NHANKHAU bằng cách tìm dựa trên tên họ tên:



Chúng ta cũng có thể viết thêm một câu truy vấn LINQ dưới đây để lấy về tất cả nhân khẩu từ CSDL hiện tại đang tạm trú tại Đông Hòa và không có tiền án tiền sự

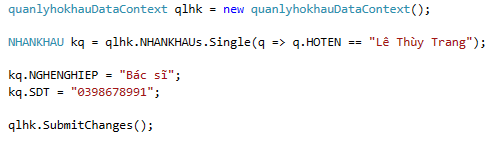


Dùng Contains kết hợp với mỗi NOITAMTRU như một phần của câu truy vấn để lấy về chỉ các nhân khẩu có địa chỉ tạm trú ở Đông Hòa.

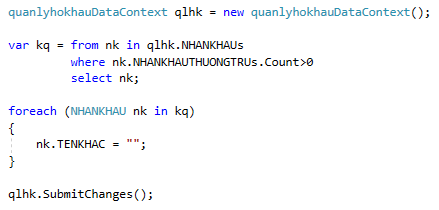
1. Change Tracking và DataContext.SubmitChanges():

Khi chúng ta thực hiện các câu truy vấn và lấy về các đối tượng như đối tượng product ở trên, LINQ to SQL sẽ mặc nhiên lưu vết các thao tác thay đổi hay cập nhật mà chúng ta thực hiện trên các đối tượng đó (gọi là change tracking). Chúng ta có thể thực hiện bao nhiêu câu truy vấn và thay đổi những giá trị mà mình muốn bằng cách dùng LINQ to SQL DataContext, và tất cả các thay đổi đó sẽ được lưu vết lại.

Ghi chú: Việc lưu vết LINQ to SQL xảy ra bên phía chương trình, và không liên quan gì đến CSDL. Có nghĩa là ta không hề dùng tài nguyên trên CSDL và không cần cài đặt thêm hay thay đối bất kỳ thứ gì trên CSDL đế cho phép làm điều này. Sau khi đã cập nhật các đối tượng chúng ta lấy từ LINQ to SQL, chúng ta có thể gọi phương thức “SubmitChanges()” trên lớp DataContext để cập nhật lại các thay đối lên CSDL. Việc gọi phương thức này sẽ làm cho LINQ to SQL để tính toán động và thực thi các câu lệnh SQL phù hợp để cập nhật CSDL. Ví dụ, câu lệnh dưới đây cập nhật lại giá tiền và số lượng đơn vị còn lại của sán phấm “Chai”:



Khi ta gọi qlhk.SubmitChanges() như ở trên, LINQ to SQL sẽ xây dựng và thực thi một câu lệnh SQL “UPDATE” mà nó sẽ cập nhật lại hai thuộc tính của NHANKHAU mà chúng ta đã sửa lại như trên. Đoạn code dưới đây sẽ duyệt qua danh sách các nhân khẩu thường trú và sau đó chuyển tên khác của nhân khẩu thành rỗng:



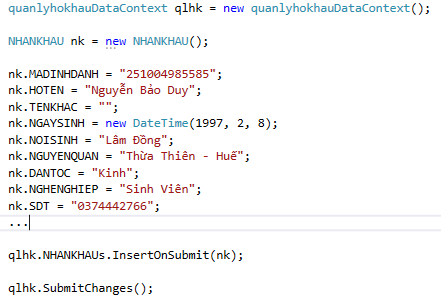
Khi ta gọi qlhk.SubmitChanges(), LINQ to SQL sẽ tính toán và thực thi một tập thích hợp các phát biểu UPDATE đế cập nhật các nhân khẩu có thuộc tính TENKHAC đã bị thay đổi. Hãy nhớ là nếu giá trị của các thuộc tính của đối tượng NHANKHAU không bị thay đối bởi câu lệnh trên, có nghĩa là bản thân đối tượng không bị thay đổi, thì LINQ to SQL cũng sẽ không thực thi bất kỳ câu lệnh UPDATE nào trên đối tượng đó. Ví dụ, nếu tất cả NHANKHAU có TENKHAC vốn đã là rỗng, thì việc gọi SubmitChanges() sẽ chắng làm thực thi bất kỳ câu SQL nào. Vì vây, chỉ các NHANKHAU trong ví dụ trên có Tên khác mới được cập nhật khi gọi SubmitChanges().

1. Các ví dụ Insert và Delete:

Ngoài việc cập nhật các dòng đã có trong CSDL, LINQ to SQL còn cho phép chúng ta thêm và xóa dữ liệu. Ta có thể làm được điều này bằng việc thêm/bớt các đối tượng dữ liệu từ các tập hợp bảng trong lớp DataContext, và sau đó gọi SubmitChanges(). LINQ to SQL sẽ lưu vết lại các thao tác này, và tự động thực thi câu lệnh SQL INSERT hay DELETE phù hợp khi phương thức SubmitChanges() được gọi.

1. Thêm một NHANKHAU:

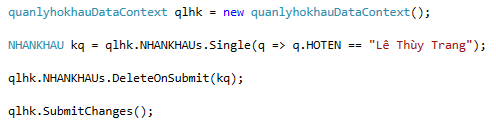
Ta có thể thêm một nhân khẩu mới vào CSDL bằng việc tạo ra một đối tượng thuộc lớp “NHANKHAU”, gán các giá trị thuộc tính, và sau đó thêm nó vào tập hợp “NHANKHAU” của DataContext:



Khi gọi “SubmitChanges” như trên, một dòng mới sẽ được thêm vào bảng NHANKHAU. Lưu ý: Với những thuộc tính chưa được điền của đối tượng NHANKHAU LINQ sẽ tự động bỏ qua các đối tượng này khi tạo câu INSERT, tuy nhiên nếu đối tượng thuộc kiểu NOT NULL, LINQ to SQL sẽ thả ra một mã lỗi thay vì tạo câu INSERT.

1. Xóa một sản phẩm:

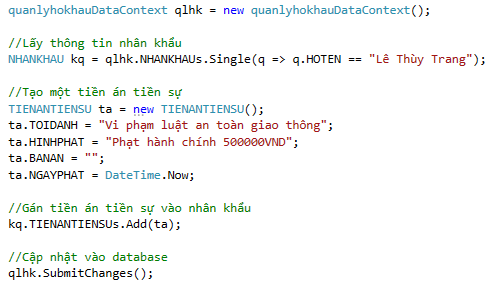
Cũng tương tự như việc thêm một sản phẩm mới bằng cách tạo đổi tượng Product vào tập hợp Products của DataContext, chúng ta cũng có thể làm một cách ngược lại khi muốn xóa một nhân khẩu từ CSDL bằng cách xóa nó khỏi tập hợp này:



Chú ý ta lấy thông tin nhân khẩu cần xóa bằng câu truy vấn LINQ đến nhân khẩu này, rồi sau đó truyền nó cho phương thức DeleteOnSubmit ( hoặc DeleteAllOnSubmit đối với một danh sách nhân khẩu) của tập hợp NHANKHAU trong DataContext. Khi gọi SubmitChanges(), tất cả các nhân khẩu đó sẽ bị xóa khỏi CSDL.

1. Cập nhật thông qua các quan hệ:

Điều làm cho các trình ORM như LINQ to SQL cực kỳ mềm dẻo là nó cho phép chúng ta dễ dàng mô hình hóa mối quan hệ giữa các bảng trong mô hình dữ liệu. Ví dụ, ta có thế mô hình hóa mỗi TIENANTIENSU trong một NHANKHAU, mỗi SOHOKHAU để chứa các NHANKHAUTHUONGTRU, kết hợp NHANKHAU với NHANKHAUTHUONGTRU, và làm cho mỗi SOHOKHAU kết hợp với một tập các TIENANTIENSU. Cách xây dựng và mô hình hóa các mối quan hệ đã được thực hiện tại phần V.2.1. LINQ to SQL cho phép ta tận dụng được ưu điểm của các mối quan hệ trong việc truy vấn và cập nhật dừ liệu. Ví dụ, ta có thể viết đoạn lệnh đề tạo một TIENANTIENSU kết hợp với NHANKHAU và thêm chúng vào CSDL như sau:



Một đối tượng TIENANTIENSU được thêm vào tập hợp TIENANTIENSUs của một NHANKHAU. Nó sẽ chỉ ra ràng có một mối quan hệ giữa hai đối tượng, và làm cho LINQ to SQL tự động duy trì mối quan hệ foreign-key/primary-key giữa cả hai khi ta gọi SubmitChanges.

Một ví dụ khác cho thấy LINỌ to SQL có thể giúp quản lý quan hệ giữa các bảng như thế nào và giúp cho việc lập trình sáng sủa hơn, xét ví dụ khi tạo một thông tin thường trú cho một NHANKHAU đã có. Sau đó ta thêm nhân khẩu thường trú này vào một sổ hộ khẩu, rồi ta sẽ tạo tiếp 2 mục tiểu sử cho nhân khẩu này. Sau đó, tạo thông tin cho nhân khẩu thường trú này làm một cán bộ và cập nhật các thay đối vào CSDL.

quanlyhokhauDataContext qlhk = new quanlyhokhauDataContext();

//Lấy thông tin nhân khẩu

NHANKHAU kq = qlhk.NHANKHAUs.Single(q => q.MADINHDANH == "083456789019");

//Tạo một NHANKHAUTHUONGTRU

NHANKHAUTHUONGTRU tt = new NHANKHAUTHUONGTRU();

tt.DIACHITHUONGTRU = "Tân Lập, Đông Hòa, Dĩ An, Bình Dương";

tt.QUANHEVOICHUHO = "Chủ hộ";

//Gán NHANKHAUTHUONGTRU vào NHANKHAU

kq.NHANKHAUTHUONGTRUs.Add(tt);

//lấy thông tin một sổ hộ khẩu

SOHOKHAU so = qlhk.SOHOKHAUs.Single(q => q.SOSOHOKHAU == "SH0000001");

//Thêm NHANKHAUTHUONGTRU vào SOHOKHAU

so.NHANKHAUTHUONGTRUs.Add(tt);

//tạo 2 TIEUSU

TIEUSU ts1 = new TIEUSU();

ts1.THOIGIANBATDAU = new DateTime(1999, 1, 1);

ts1.THOIGIANKETTHUC = new DateTime(2012, 8, 1);

ts1.CHOO = "Tân Lập, Đông Hòa, Dĩ An, Bình Dương";

ts1.NGHENGHIEP = "Học sinh";

ts1.NOILAMVIEC = "Bình Dương";

TIEUSU ts2 = new TIEUSU();

ts2.THOIGIANBATDAU = new DateTime(2012, 9, 5);

ts2.THOIGIANKETTHUC = new DateTime(2017, 8, 1);

ts2.CHOO = "Tân Lập, Đông Hòa, Dĩ An, Bình Dương";

ts2.NGHENGHIEP = "Sinh viên";

ts2.NOILAMVIEC = "Bình Dương";

//Thêm TIEUSU vào NHANKHAU

kq.TIEUSUs.Add(ts1); kq.TIEUSUs.Add(ts2);

//Tạo một CANBO

CANBO cb = new CANBO();

cb.LOAICANBO = "0";

cb.TENTAIKHOAN = "cbbongda1";

cb.MATKHAU = "123";

//Gán CANBO vào NHANKHAUTHUONGTRU

tt.CANBOs.Add(cb);

//Cập nhật vào database

qlhk.SubmitChanges();

Mô hình lập trình trên cho phép thực hiện tất cả các công việc này một cách cực kỳ rõ ràng, dễ hiểu theo phong cách hướng đối tượng.

1. 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 sẽ không bao giờ ở trong một trạng thái không toàn vẹn nếu ta thực thi nhiều câu lệnh - hoặc tất cá các thao tác ta làm sẽ được lưu lại, hoặc không có bất kỳ thay đổi nào.

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

1. Kiểm tra dữ liệu và Business Logic:

Một trong những điều quan trọng mà các nhà phát triển cần nghĩ đến khi làm việc với đữ liệu là làm sao để kết hợp được các phép xác thực dữ liệu và các quy tắc chương trình (business logic). LINQ to SQL cũng hồ trợ nhiều cách để chúng có thế dễ dàng tích hợp chúng vào với các mô hình dữ liệu của họ.

LINQ to SQL cho phép ta thêm khả năng xác thực dữ liệu mà không phụ thuộc vào cách tạo ra mô hình dữ liệu cũng như nguồn dữ liệu. Điều này cho phép chúng ta có thế lặp lại các phép kiểm tra ở nhiều vị trí khác nhau, làm cho code rõ ràng và dễ bảo trì hơn rất nhiều.

1. Hỗ trợ kiểm tra các giá trị thuộc tính dựa trên schema của CSDL:

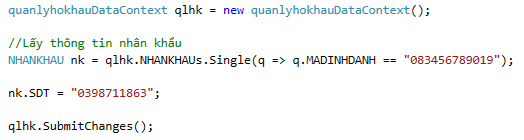
Khi định nghĩa các lớp mô hình dữ liệu dùng O/R designer trong VS, chúng sẽ mặc nhiên được gán các quy tắc xác thực dựa trên cấu trúc định nghĩa trong CSDL.

Kiểu dữ liệu của thuộc tính trong các lớp mô hình dữ liệu (thực thể) sẽ khớp với các kiểu dữ liệu tương ứng trong CSDL. Điều này có nghĩa là chúng ta sẽ gặp lỗi biên dịch nếu cố gắng gán một giá trị kiểu boolean vào cho một thuộc tính decimal, hoặc nếu thử ép kiểu dữ liệu một cách không hợp lệ. Nếu một cột trong CSDL được đánh dấu cho phép mang giá trị NULL, khi đó thuộc tính tương ứng trong mô hình dữ liệu được tạo bởi O/R designer cũng cho phép NULL. Các cột không cho phép NULL sẽ tự động thả ra các exception nếu ta cố gắng lưu một đối tượng có thuộc tính đó mang giá trị NULL. LINQ to SQL sẽ đảm bảo các cột định danh/duy nhất không bị trùng lắp trong CSDL.

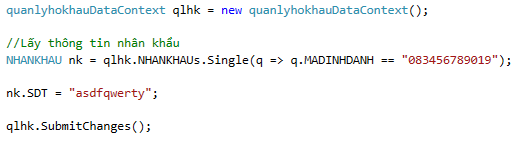
Chúng ta có thể dùng O/R designer đế ghi đè lên các quy tắc xác thực dựa trên schema nếu muốn, nhưng các quy tắc này sẽ được tạo ra tự động và chúng ta không cần làm bất kỳ điều gì để cho phép chúng. LINQ to SQL cũng tự động xử lý các chuỗi escape, do vậy ta không cần lo lắng về lỗi SQL injection.

1. Hỗ trợ tùy biến việc kiểm tra giá trị các thuộc tính:

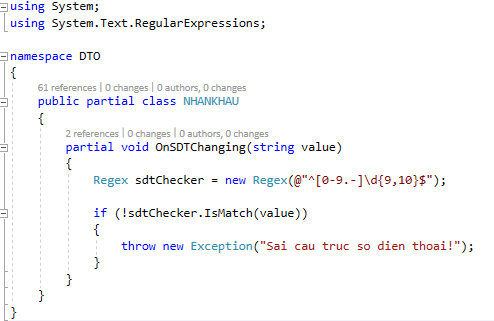
Việc kiểm tra dữ liệu dựa trên cấu trúc định nghĩa trong CSDL rất hữu ích, nhưng chỉ được coi như ở mức cơ bản, trong thực tế có thể chúng ta sẽ gặp phải những yêu cầu kiểm tra phức tạp hơn nhiều. Hãy xem một ví dụ trong CSDL qlhk, khi định nghĩa thuộc tính SDT thuộc lớp NHANKHAU có kiểu dữ liệu là nvarchar. Các nhà phát triến dùng LINQ to SQL có thế viết code giống như dưới đây để cập nhật nó với một số điện thoại hợp lệ:



Vấn đề là đoạn code trên được coi là họp lệ đứng từ góc độ kiểu dữ liệu SQL, vì chuồi trên vẫn là một chuỗi nvarchar mặc dù có thể nó không phải là một số phone hợp lệ:



Để tránh việc thêm các SDT kiểu như trên vào CSDL, chúng ta có thể thêm một quy tắc kiểm tra tính hợp lệ vào lớp NHANKHAU. Thêm một quy tắc để kiểm tra trong LINQ to SQL thực sự rất đơn giản, tất cả những gì chúng ta cần làm là thêm một partial class vào và định nghĩa phương thức như sau:



Đoạn code trên tận dụng ưu điểm của 2 đặc tính trong LINQ to SQL:

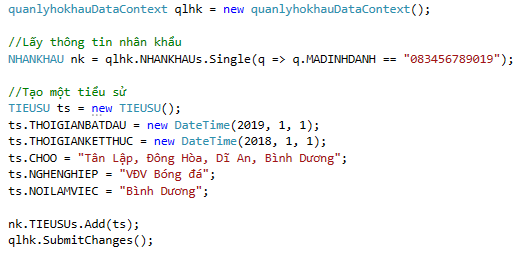
1) Tất cả các lớp được tạo ra đều là partial - có nghĩa là nhà phát triển có thể dễ dàng thêm vào các phương thức, thuộc tính và thậm chí cả các sự kiện (và đặt chúng trong một file riêng biệt). Điều này làm cho việc thêm các quy tắc xác thực và các hàm phụ trợ vào mô hình dữ liệu và lớp DataContext rất dễ dàng. Chúng ta không cần cấu hình hay viết thêm các code nào khác để làm được điều này.

2) LINQ to SQL đã tạo sẵn một loạt các điểm mở rộng trong mô hình dữ liệu và lớp DataContext mà ta có thế dùng để thêm vào các phép kiểm tra dữ liệu trước và sau khi thực hiện các công việc. Nhiều trong số đó ứng dụng một đặc tính ngôn ngữ mới được gọi là “partial method” có trong VB và C# trên VS. *(Partial Method là một tính năng giúp chia một lớp hay phương thức thành nhiều phần hay nhiều file khác nhau, các phần hay file này sẽ chứa một phần tính năng của nó và sẽ được ghép lại với nhay khi ta compile chương trình)*

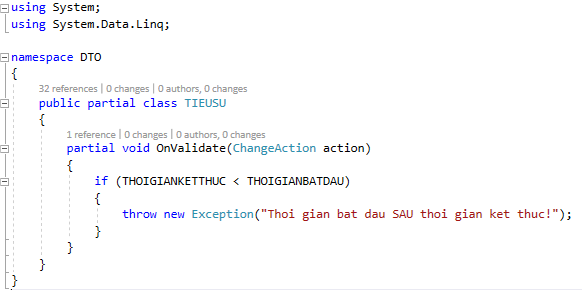
Trong ví dụ về việc kiểm tra tính hợp lệ dữ liệu ở trên, ta dùng phương thức OnSDTChanging, đây là một phương thức sẽ được thực thi bất kỳ lúc nào người dùng gán lại giá trị cho thuộc tính SDT trên một đối tượng NHANKHAU. Chúng ta cũng có thể dùng phương thức này đế xác thực giá trị đầu vào theo bất kỳ cách gì mình muốn (trong ví dụ này là một biểu thức chính quy). Nếu giá trị đã hợp lệ, ta chỉ đơn giản return và không làm gì cả, khi đó LINQ to SQL sẽ cho là các giá trị này là giá trị hợp lệ, ngược lại ta có thể thả ra một Exception bên trong phương thức kiểm tra, và phép gán khi đó sẽ không được thực hiện.

1. Hỗ trợ tùy biến kiểm tra tính hợp lệ của thực thể:

Việc kiểm tra trên từng thuộc tính như trong các ví dụ trên rất hữu dụng khi ta muốn kiểm tra giá trị của từng thuộc tính riêng lẻ. Nhưng đôi khi, cần phải kiếm tra dựa trên nhiều giá trị của các thuộc tính khác nhau. Ví dụ ta đặt giá trị cho 2 thuộc tính “THOIGIANBATDAU” và “THOIGIANKETTHUC” trong TIEUSU:



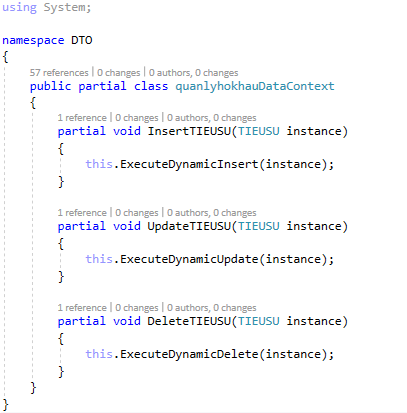
Đoạn lệnh trên là hợp lệ nếu chỉ đơn thuần xét từ góc độ ngôn ngữ - nhưng sẽ là không có ý nghĩa khi ta lại muốn đặt thời gian bắt đầu sau thời gian kết thúc. Từ phiên bản LINQ to SQL beta 2, chúng ta có thể thêm vào các quy tắc kiếm tra cho từng thực thể đế tránh các lỗi như trên bằng cách thêm lớp partial cho lớp “TIEUSU” và hiện thực hóa hàm OnValidate(), hàm này sẽ được gọi trước khi dữ liệu được đưa vào CSDL. Bên trong phương thức này, chúng ta có thể truy cập và kiểm tra tất cả các thuộc tính của lớp trong mô hình dữ liệu.



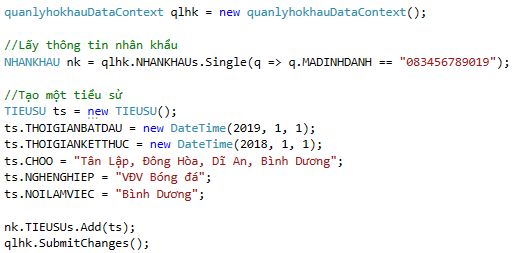
Bên trong phương thức này, ta có thể kiếm tra giá trị bất kỳ thuộc tính nào, và thậm chí có thể truy cập (chỉ đọc) vào các đối tượng liên quan, và có thể thả ra một exception nếu có tồn tại các giá trị không hợp lệ. Bất kỳ một exception nào được thả ra từ phương thức OnValidate() sẽ làm cho việc cập nhật bị hủy bỏ, và hủy bỏ các thay đổi trong transaction.

1. Tùy biến các phương thức kiểm tra việc thêm/xóa/sửa dữ liệu:

Có nhiều lúc ta muốn thêm các phép kiểm tra khi thêm/xóa/sửa dữ liệu. LINQ to SQL Beta2 cho phép làm điều này bằng cách cho phép thêm vào một lớp partial để mở rộng lớp DataContext và sau đó hiện thực hóa các phương thức để tùy biến các thao tác thêm/xóa/sửa cho các thực thể. Các phương thức này sẽ được thực thi tự động khi ta gọi SubmitChanges() trên lớp DataContext. Chúng ta có thể thêm các phép kiểm tra thích hợp vào bên trong các phương thức đó - và nếu dữ liệu hợp lệ, LINQ to SQL sẽ tiếp tục lưu lại các thay đổi vào CSDL (bằng cách gọi phương thức “ExecuteDynamicXYZ” cua DataContext).



Một trong những điều thú vị là các phương thức phù hợp sẽ được gọi tự động, không phụ thuộc vào ngữ cảnh mà đối tượng được tạo/xóa/sửa. Hãy xem ví dụ sau, ở đây ta muốn tạo một TIEUSU mới và kết hợp nó với một NHANKHAU đã có:

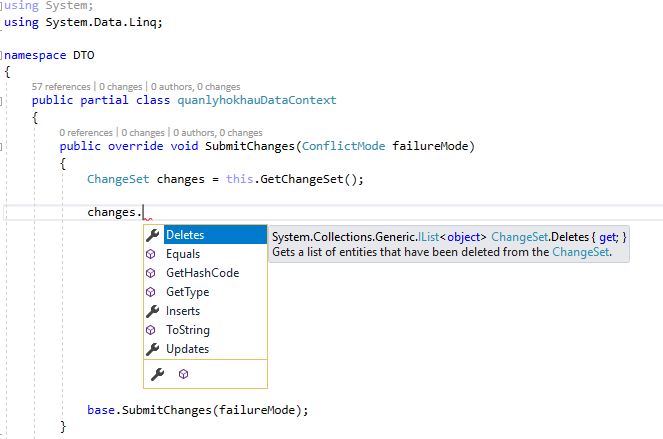


Khi ta gọi qlhk.SubmitChanges(), LINQ to SQL sẽ xác định là nó cần lưu lại một đối tượng TIEUSU, và phương thức InsertTIEUSU sẽ tự động được gọi.

1. Nâng cao: Xem danh sách các thay đổi trong transaction:

Đôi khi chúng muốn thêm các quy tắc kiểm tra mà không thế chỉ dựa trên từng thao tác thêm/xóa/sửa riêng lẻ, thay vào đó ta phải có thể duyệt qua toàn bộ các thao tác đã thực hiện trong transaction.

Bắt đầu từ bản Beta2 của .NET 3.5, LINQ to SQL cho phép ta truy cập vào danh sách này bằng cách gọi phương thức DataContext.GetChangeList(). Nó sẽ trả về một đối tượng ChangeList chứa các tập hợp cho các thao tác thêm/xóa/sửa đã được thực hiện. Một cách tiếp cận là ta có thể tạo một lớp thừa kế từ lớp DataContext và override phương thức SubmitChanges(). Khi đó ta có thể lấy ChangeList (Deletes, Inserts, Updates) cho thao tác cập nhật và thực hiện các phép kiếm tra cần thiết trước khi thực thi:



1. Xử lý các thay đổi đồng thời với Optimistic Concurrency:

Một trong những vấn đề mà các nhà phát triển phải nghĩ đến trong môi trường đa người dùng (Multi-user) là làm thể nào có thế xử lý các thao tác cập nhật trên các cùng một tập dữ liệu. Ví dụ, cho là có hai người dùng đang cùng lấy về một đối tượng CANBO bên trong một ứng dụng, và một người đặt lại giá trị cho LOAICANBO là 0, trong khi người kia đặt lại là 1. Nếu cả hai người dùng đều lưu lại các thay đổi đó vào CSDL, nhà phát triến cần cân nhắc việc xử lý tranh chấp dữ liệu.

Một cách tiếp cận đơn giản là “let the last writer win” (người cuối cùng là người chiến thắng) - có nghĩa là những thay đối bởi người đầu tiên sẽ bị thay đối mà không biết. Và đây thường được coi là một phương pháp kém cỏi (và không đúng) - vì nó sẽ khiến người dùng cảm thấy khó sử dụng. Một cách tiếp cận khác mà LINQ to SQL hỗ trợ là dùng mô hình optimistic concurrency - khi đó LINQ to SQL sẽ tự động xác định xem giá trị gốc trong CSDL đã bị thay đổi bởi người dùng khác hay chưa. LINQ to SQL sau đó sẽ cung cấp một danh sách các giá trị bị xung đột để nhà phát triển có thế chọn giải pháp xử lý hoặc có thể yêu cầu người dùng chọn một thao tác nào họ muốn.

1. Dùng SPROCs hoặc tùy biến logic các câu SQL:

Một trong những câu hỏi mà các nhà phát triến (và đặc biệt là các DBA - các nhà quản trị CSDL), những người đã từng viết các thủ tục (SPROC) với các câu SQL tùy biến thường hỏi khi nhìn thấy LINQ to SQL lần đầu tiên là: “làm sao tôi có thể kiếm soát hoàn toàn các câu lệnh SQL được thực thi bên dưới?” Tin tốt là LINQ to SQL là một mô hình cực kỳ mềm dẻo, nó cho phép các nhà phát triển có thế thay thế các câu lệnh của LINQ to SQL bằng các thủ tục insert, update, delete mà họ tự định nghĩa.

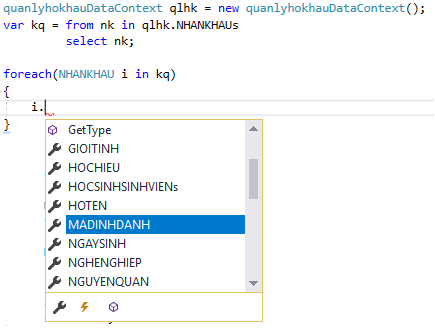
Điều thực sự thú vị là ta có thể bắt đầu bằng cách định nghĩa mô hình dữ liệu của riêng chúng ra và để LINQ to SQL tự thực hiện các thao tác thêm/sửa/xóa. Rồi sau đó ta có thể tùy biến lại mô hình dữ liệu để thực hiện các thao tác cập nhật với các thủ tục hoặc các câu SQL của mình mà không phải thay đổi bất kỳ đoạn lệnh nào dùng mô hình dữ liệu đó, và cũng chẳng phải thay đối bất kỳ quy tắc kiếm tra đã tạo trước đó. Điều này cung cấp khả năng tùy biến rất lớn khi xây dựng ứng dụng.

### Lấy dữ liệu dùng Stored Procedure:

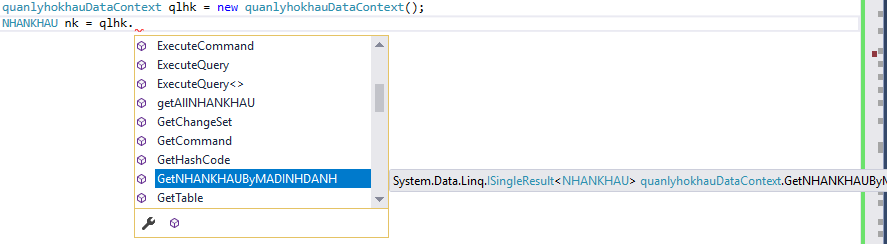
1. Có nên dùng SPROC hay không:

Câu hỏi liệu nên dùng các câu SQL động được sinh ra bởi trình ORM hay dùng Stored Procedure khi xây dựng lớp dữ liệu là một chủ đề không bao giờ kết thúc tranh cãi giữa các nhà phát triển, kĩ sư phần mềm và các DBA. Và câu trả lời về vấn đề này thường là: “Tùy thuộc vào điểm mạnh của team lập trình!”

LINQ to SQL đi cùng với .NET rất mềm dẻo, và có thế 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 lại dùng các câu SQL động hay thông qua các SPROCs. Trong phần V.2.2 chúng ta đã nói về cách viết các biểu thức truy vấn LINQ to một mô hình dữ liệu LINQ to SQL dùng đoạn mã như sau:

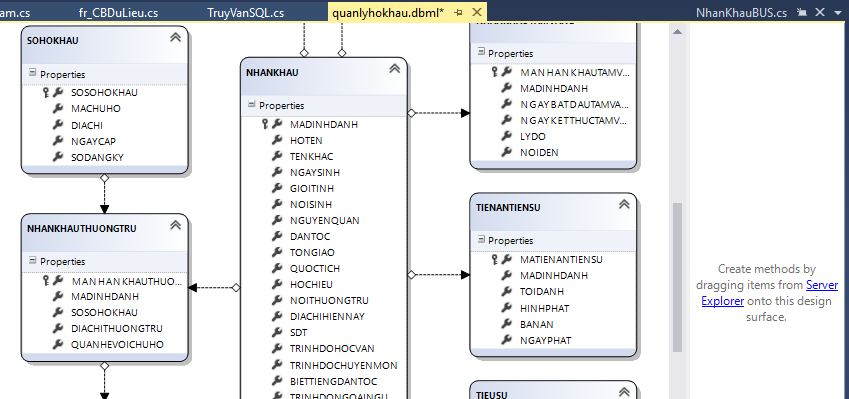


Khi ta viết các biếu thức LINQ kiểu như vậy, LINQ to SQL sẽ thực thi các câu lệnh SQL động để ta có thế lấy về các đối tượng khớp với câu truy vấn của mình. Và cũng như đã nói ở phần V.2.2, chúng ta cũng có thế dùng các SPROCs trong CSDL trong lớp DataContext, nó cung cấp một cách khác để lấy về các đối tượng Products bằng cách gọi thủ tục tương ứng:



Khả năng này cho phép ta dùng các câu truy vấn 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.

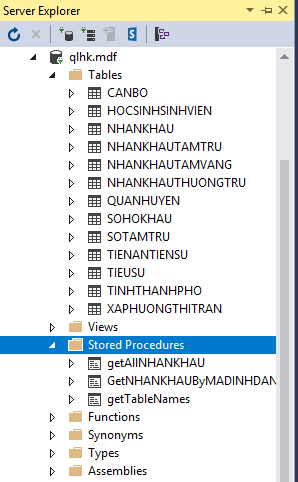
1. Các bước ánh xạ gọi SPROC dùng LINQ to SQL:



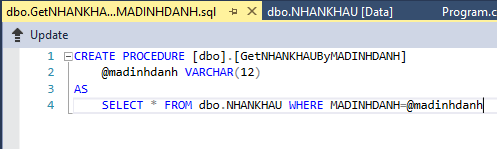
Như đã giới thiệu ở mục IV.6.2, Mô hình dữ liệu LINQ to SQL bao gồm 2 cửa sổ con, cửa sổ bên trái cho phép chúng ta định nghĩa mô hình dữ liệu sẽ ánh xạ vào CSDL, cửa sổ bên phải cho phép ánh xạ các thủ tục và hàm vào đối tượng DataContext, điều này cho phép chúng ta có thể thay thế các câu SQL động trong việc lấy dữ liệu về.

1. Ánh xạ một SPROC vào một DataContext của LINQ:

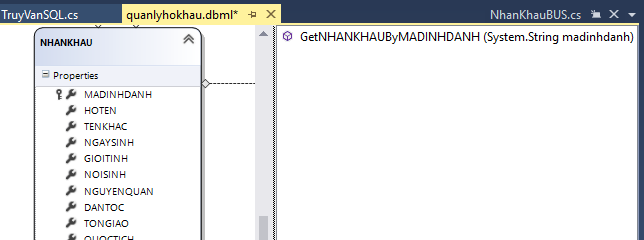
Để ánh xạ một SPROC vào lớp DataContext, ta cần sử dụng cửa sổ Server Explorer trong VS và chọn Stored Procedures hoặc Functions:



Chúng ta có thể nháy đúp vào bất kỳ thủ tục SPROC nào ở trên để mở và chỉnh sửa chúng, ví dụ như “GetNHANKHAUByMADINHDANH” trong QLHK như dưới đây:



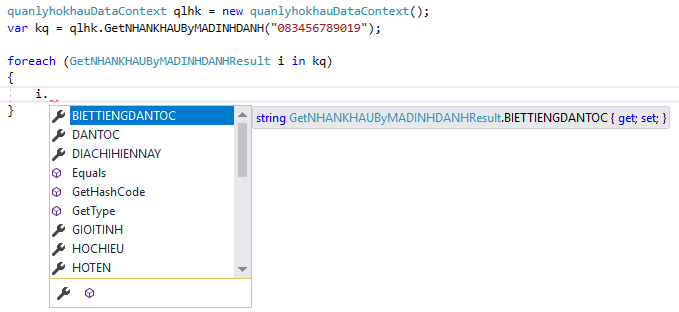
Để ánh xạ SPROC ở trên vào DataContext, chúng ta có thể kéo/thả chúng từ cửa sổ Server Explorer lên cửa sổ O/R Designer. Việc này sẽ giúp sinh ra một thủ tục trong lớp DataContext của LINQ to SQL như sau:



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 về một dãy các đối tượng có kiểu “GetNHANKHAUByMADINHDANHResult”. Chúng ta có thể đổi tên của phương thức nếu muốn bằng cách chọn nó rồi dùng Property Grid để đặt lại tên khác.

1. Gọi một SPROC mới được tạo:

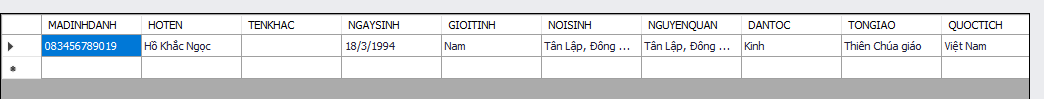
Khi đã hoàn thành các bước trên để ánh xạ một SPROC vào lớp DataContext của chúng ta, ta có thể dễ dàng lấy dữ liệu về. Tất cả những gì chúng ta cần làm là gọi phương thức chúng ta đã ánh xạ trong DataContext để lấy về một chuỗi các đối tượng SPROC:



Thêm nữa, thay vì lặp qua tập kết quả như ở trên, ta cũng có thế gắn nối nó vào cho một control đế hiến thị ra màn hình, ví dụ như tôi có thể dùng DataGridView:



Khi đó (danh sách) thông tin nhân khẩu sẽ được hiển thị như sau:

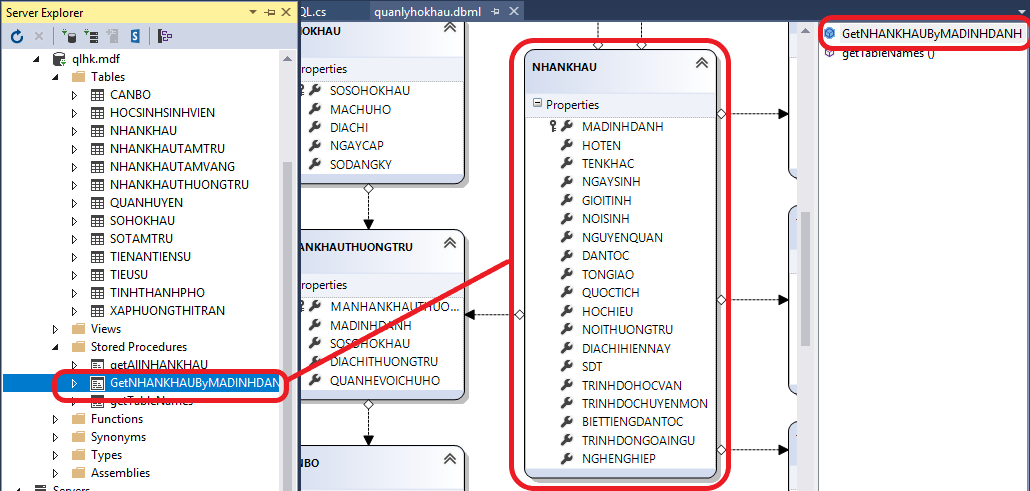


1. Ánh xạ kiểu trả về của phương thức SPROC vào một lớp trong mô hình dữ liệu:

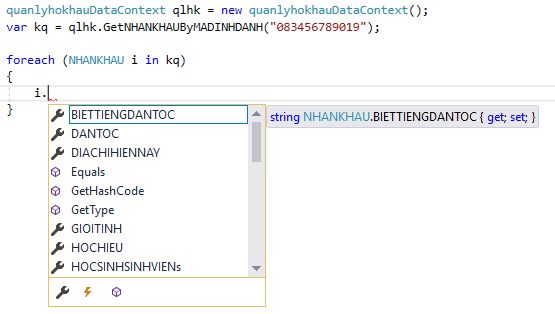
Trong thủ tục GetNHANKHAUByMADINHDANH ở trên, thủ tục trả về một danh sách dữ liệu bao gồm 10 cột. LINQ to SQL designer sẽ tự động tạo ra một lớp có tên GetNHANKHAUByMADINHDANHResult để biểu diễn kết quả này.

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

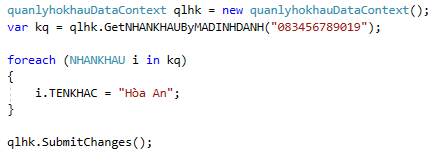
Xét lại ví dụ cũng như trước, ta tiếp tục dùng GetNHANKHAUByMADINHDANH va tạo bên trong lớp DataContext mà sẽ gọi thủ tục này bằng cách kéo nó vào cửa sổ O/R designer. Tuy nhiên thay vì thả nó vào một vị trí bất kì, Chúng ta sẽ thả nó lên trên lớp NHANKHAU mà đã tạo ra trên cửa sổ này:



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



Một ưu điếm của việc sử dụng lớp NHANKHAU như kiếu trả về là LINQ to SQL sẽ tự động quản lý các thay đổi được tạo ra trên đối tượng được trả về này, giống như làm với các đối tượng được trả về thông qua các câu truy vấn LINQ. Khi gọi “SubmitChanges()” trên DataContext, những thay đổi này cũng sẽ được cập nhật trở lại CSDL. Ví dụ, ta có thể viết đoạn code giống trên (dùng một SPROC) và thay đổi giá trị TENKHAC rồi cập nhật vào CSDL như sau:



Khi gọi SubmitChanges, nó sẽ cập nhật lại TENKHAC của NHANKHAU.

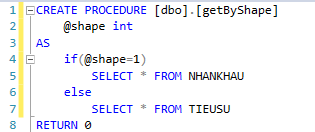
1. Xử lý các tham số thủ tục dạng OUTPUT:

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 “GetNHANKHAUByMADINHDANH” sau sẽ nhận vào một MADINHDANH như tham số đầu vào, và trả về NHANKHAU như một tham số dạng OUTPUT

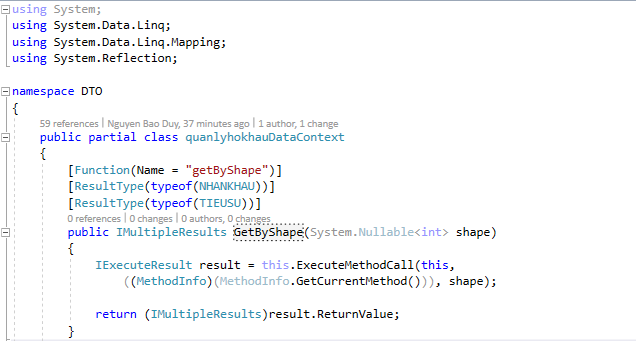
Chú ý Nếu thủ tục được kéo vào một bảng mà bảng đó không đủ biến thì thủ tục đó sẽ trả về biến thuộc kiểu bảng đó và một biến riêng dạng output.

1. Xử lý các thủ tục trả viề nhiều kiểu kết quả khác nhau:

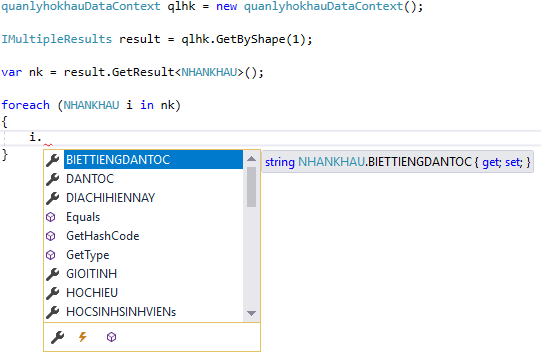
Khi một thủ tục trả về nhiều kiểu kết quả khác nhau, kiếu trả về của phương thức trên lớp DataContext không thể được ép về một kiểu cụ thể nào đó. Ví dụ, thủ tục dưới đây có thể trả về một tập các nhân khẩu hay tiểu sử tùy thuộc vào tham số đầu vào:



LINQ to SQL hỗ trợ việc tạo các phương thức trợ giúp cho phép trả về NHANKHAU hay TIEUSU bằng cách thêm một lớp partial quanlyhokhauDataContext vào project và định nghĩa một phương thức trong lớp này (trong ví dụ này chúng ta gọi là getByShape) để gọi thủ tục và trả về một đối tượng có kiểu IMultipleResult như trong ví dụ sau:

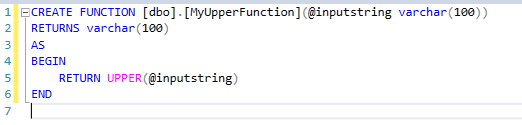


Một khi đã thêm phương thức này vào project, ta có thể gọi chuyển kiểu về kiểu thích hợp là NHANKHAU hoặc TIEUSU:



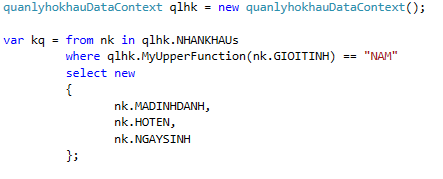
1. Hỗ trợ các hàm do người dùng tự định nghĩa (UDF):

Thêm vào việc hỗ trợ 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 thêm vào lớp DataContext như một phương thức, ta có thể dùng các hàm UDF này trong các câu lệnh LINQ. Ví dụ, hãy xem các hàm UDF đơn giản có tên MyUpperFunction sau đây:

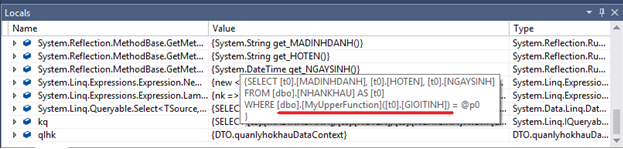


Chúng ta có thế kéo và thả nó từ cửa sổ Server Explorer lên cửa sồ LINQ to SQL Designer để thêm nó vào lớp DataContext như một phương thức.

Chúng ta sau đó có thể dùng hàm UDF này ngay bên trong các biếu thức LINQ khi viết các câu truy vấn (giống như chúng ta đang dùng trong biếu thức Where như dưới đây):



Khi mở bằng LINQ to SQL Debug Visualizer bạn có thể thấy các LINQ to SQL chuyển đổi câu truy vấn ở trên thành câu lệnh SQL đế thực thi hàm UDF khi chạy:



### Cập nhật dữ liệu dùng Stored Procedure:

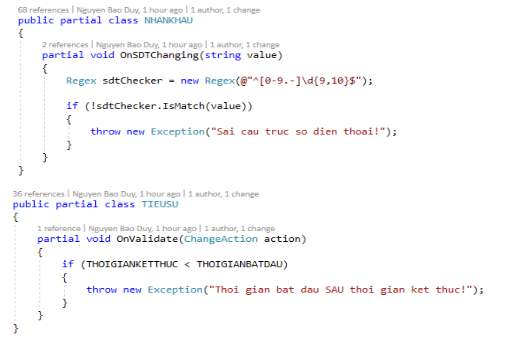
1. Tạo lớp truy vấn dữ liệu (chưa dùng đến các thủ tục):



*Thêm các quy tắc kiểm tra dữ liệu vào các mô hình dữ liệu:*

Sau khi định nghĩa các Ịớp trong mô hình dữ liệu và các quan hệ giữa chúng, chúng ta sẽ tiếp tục thêm vào các quy tắc kiếm tra tính hợp lệ của dữ liệu. Chúng ta có thể làm điều này bằng cách thêm các lóp partial vào trong project và thêm các quy tắc kiểm tra vào các lớp mô hình dữ liệu (Mục V.2.3).

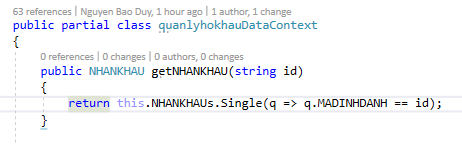
Ví dụ, thêm một quy tắc để đảm bảo rằng số điện thoại của nhân khẩu được nhập đúng định dạng, và chúng ta không cho phép thêm một tiểu sử nếu trường THOIGIANKETTHUC lớn hơn THOIGIANBATDAU. Một khi đã được định nghĩa như dưới đây, các phương thức kiểm tra sẽ tự động được thực thi bất kỳ lúc nào chúng ta cập nhật lại các đối tượng trong hệ thống.



*Thêm phương thức GetNHANKHAU() vào lớp DataContext*

Sau khi đã tạo các lớp mô hình dữ liệu và đã áp dụng các phương thức kiểm tra trên chúng, chúng ta có thể truy vấn và tương tác với dữ liệu. Chúng ta có thể làm được điều này bằng cách viết các câu lệnh LINQ với các lớp mô hình dữ liệu để truy vấn và cập nhật CSDL (Mục V.2.2). Thêm nữa ta cũng có thế ánh xạ các SPROC vào lớp DataContext và dùng chúng để đưa dữ liệu vào CSDL (Mục V.2.5).

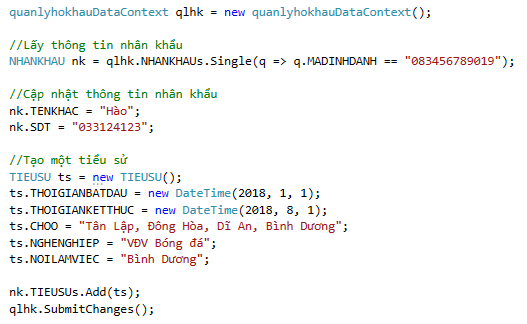
Khi xây đựng các lớp dữ liệu LINQ to SQL, chúng ta sẽ thường có nhu cầu đưa các câu lệnh LINQ thường dùng vào các phương thức hỗ trợ trong lớp DataContext. Và chúng ta có thể thực hiện điều này bằng cách thêm một class Partial vào project. Ví dụ, thêm một phương thức có tên "GetNHANKHAU()" cho phép chúng ta tìm kiếm và lấy về các đối tượng NHANKHAU từ CSDL dựa trên giá trị của MADINHDANH:



1. Dùng lớp truy cập dữ liệu (chưa sử dụng SPROC):

Hiện tại chúng ta đã có một lớp truy cập dữ liệu (data access layer) để biểu diễn mô hình dữ liệu, tích hợp các quy tắc và cho phép chúng ta có thế thực hiện truy vấn, cập nhật, thêm và xóa dữ liệu.

Xét một trường hợp đơn giản là khi chúng ta lấy về một đối tượng nhân khẩu đã có, cập nhật lại giá trị của trường TENKHAC và SDT, sau đó tạo mới một đối tượng TIEUSU và kết hợp chúng với nhau. Chúng ta có thế viết đoạn lệnh dưới đây để làm tất cả điều này trong một transaction. LINQ to SQL sẽ đảm bảo các thủ tục kiếm tra sẽ được thực thi và cho phép trước khi dữ liệu có thể được cập nhật một cách thực sự:



LINQ to SQL theo dõi các thay đối mà chúng ta đã tạo trên các đối tượng được lấy về từ DataContext, và cũng theo dõi cả các đối tượng mà chúng ta thêm vào. Khi gọi SubmitChanges(), LINQ to SQL sẽ kiểm tra xem dữ liệu có hợp lệ hay không, và có đúng với các quy tắc logic hay không, nếu đúng thì các câu SQL động sẽ được sinh ra để cập nhật bản ghi NHANKHAU ở trên, và thêm một bản ghi mới vào bảng TIEUSU.

Mô hình lập trình của LINQ to SQL đế làm việc với các đối tượng mô hình dữ liệu bằng SPROC cũng hoàn toàn tương tự với việc sử dụng các câu SQL động. Cách chúng ta thêm các quy tắc kiểm tra cũng hoàn toàn tương tự (do vậy các quy tắc mà ta đã thêm vào trước đây sẽ vẫn có hiệu quả khi chúng ta chuyến sang dùng SPROC). Đoạn lệnh ở trên để lấy về một nhân khẩu, rồi cập nhật và thêm một tiểu sử sẽ hoàn toàn giống nhau, không phụ thuộc vào việc chúng ta dùng các câu SQL động hay các SPROC đế thực hiện việc truy cập vào CSDL.

Mô hình lập trình này rất mạnh mẽ theo cả hai nghĩa: nó không bắt ta phải học hai cách dùng khác nhau, và ta cũng không cần phải quyết định ngay từ đầu là dùng SPROC hay không. Ban đầu, chúng ta có thể dùng các câu SQL động được cung cấp bởi LINQ to SQL cho tất cả các câu truy vấn, chèn, cập nhật và xóa dữ liệu. Bạn sau đó có thể thêm vào các quy tắc để kiểm tra tính hợp lệ của dữ liệu, và rồi sau nữa lại có thể thay đổi để dùng các SPROC - hoặc không tùy ý. Các đoạn lệnh và các đoạn test chúng ta đã viết trước đây sẽ vẫn được sử dụng tiếp, không phụ thuộc vào việc dùng SQL hay SPROC.

Phần tiếp theo của bài này sẽ biểu diễn cách cập nhật mô hình dữ liệu mà chúng ta đã tạo ra để dùng SPROC trong việc thêm/sửa/xóa dữ liệu, chúng ta vẫn tiếp tục dùng các quy tắc xác thực, và vẫn tiếp tục làm việc với cùng các đoạn lệnh đã viết ở trên.

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

Chúng ta có thế sửa lại lớp truy cập dữ liệu đã được xây dựng trước đây đế xử lý các thao tác cập nhật, thay vì dùng các câu SQL động, theo một trong 2 cách sau:

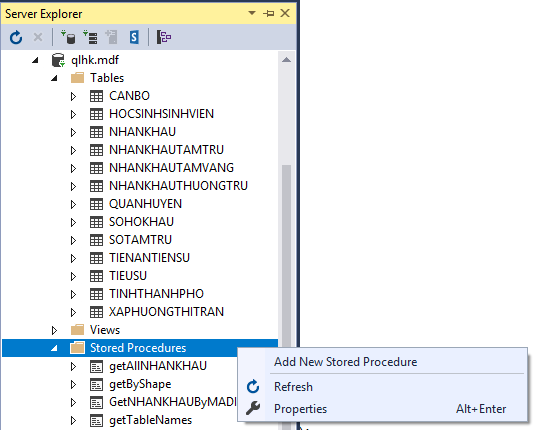
1. Dùng O/R designer để cấu hình các SPROC để thực thi khi gặp thao tác thêm/xóa/sửa dữ liệu trên các lớp mô hình dữ liệu.
2. Thêm một lớp partial quanlyhokhauDataContext vào project, rồi viết các phương thức partial tương ứng với các thao tác Insert/Update/Delete (ví dụ: InsertOrder, UpdateOrder, DeleteOrder) mà nó sẽ được gọi khi chúng ta thực hiện Insert/Update/Delete trên các đối tượng mô hình dữ liệu. Các phương thức partial đó sẽ được truyền vào các đối tượng dữ liệu mà ta muốn cập nhật, và chúng ta có thể thực thi các thủ tục hay câu lệnh SQL mà chúng ta muốn dùng đế lưu đối tượng đó vào CSDL.

Khi dùng cách [1] (dùng LINQ to SQL designer) để cấu hình các SPROC để gọi, thì thực ra nó cũng sẽ tạo ra cách lệnh tương tự như chúng ta dùng trong cách [2]. Tuy nhiên các lập trình viên có kinh nghiệm khuyến khích dùng O/R designer để cấu hình các SPROC trong 90% trường hợp - và chỉ trong các trường hơp nào chúng ta cần tùy biến lại cách gọi ở một mức độ cao, mới nên viết các lệnh một cách trực tiếp.

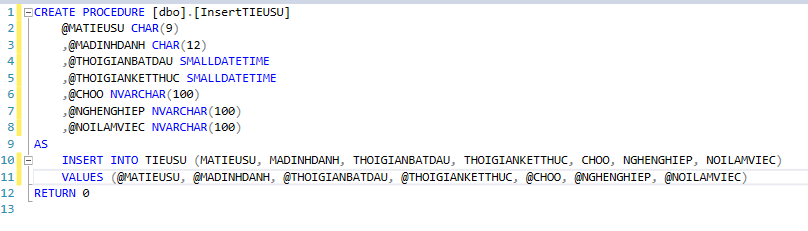
1. Thêm TIEUSU bằng cách dung SPROC:

Để bắt đầu chuyển mô hình dữ liệu sang dùng SPROC, chúng ta nên bắt đầu từ đối tượng Object.

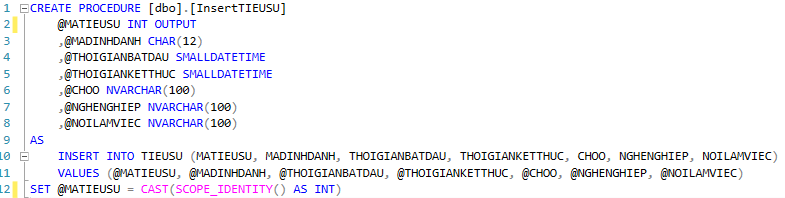
Đầu tiên, chúng ta đến cửa sổ "Server Explorer" mở nhánh Stored Procedures trong CSDL của chúng ta, và sau đó nhấn phải chuột và chọn "Add New Stored Procedure":



Sau đó ta tạo một thủ tục có tên "InsertTIEUSU" có nhiêm vụ chèn thêm một bản ghi mới vào bảng TIEUSU:

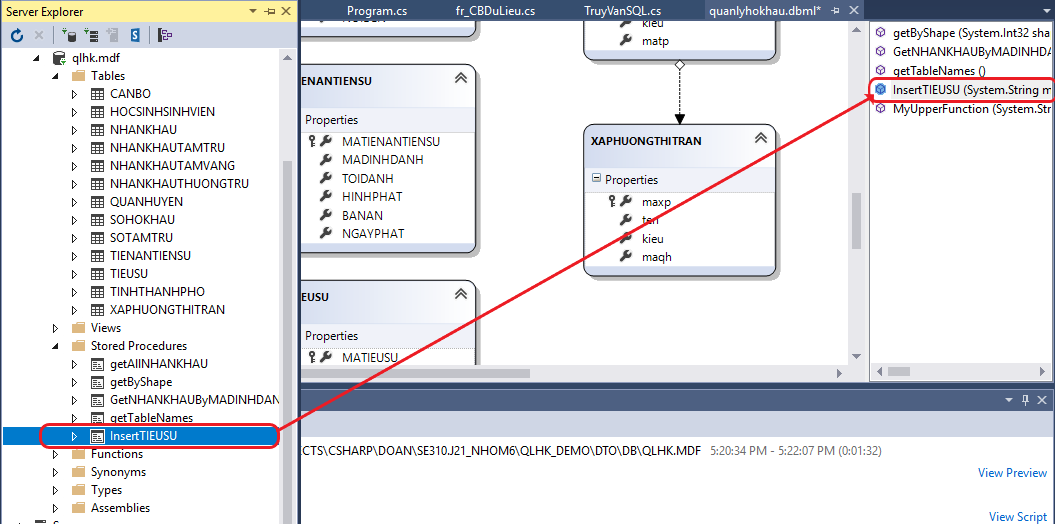


Lưu ý: Trong một số trường hợp, SPROC sẽ định nghĩa MATIEUSU như một tham số dạng OUTPUT nếu cột MATIEUSU trong CSDL là cột tự tăng mỗi khi thêm một bản ghi mới vào. Người gọi sẽ truyền giá trị NULL khi gọi nó - và thủ tục này sẽ trả về giá trị của MATIEUSU mới được tạo ra (bằng cách gọi hàm SCOPE\_IDENTITY() ở cuối thủ tục).

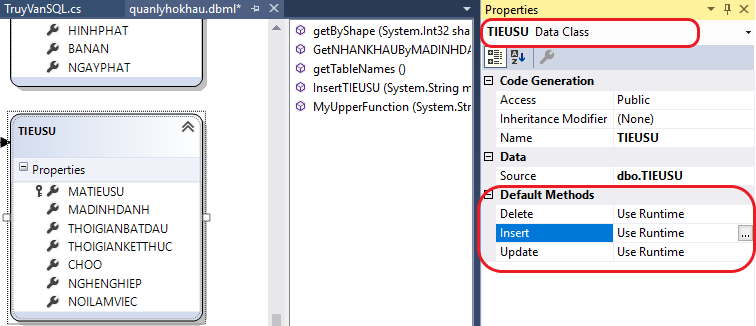


Sau khi tạo ra SPROC, chúng ta sẽ mở O/R designer của lớp truy cập dữ liệu. Chúng ta có thể kéo/thả các SPROC từ Server Explorer lên trên màn hình chính của trình thiết kế.

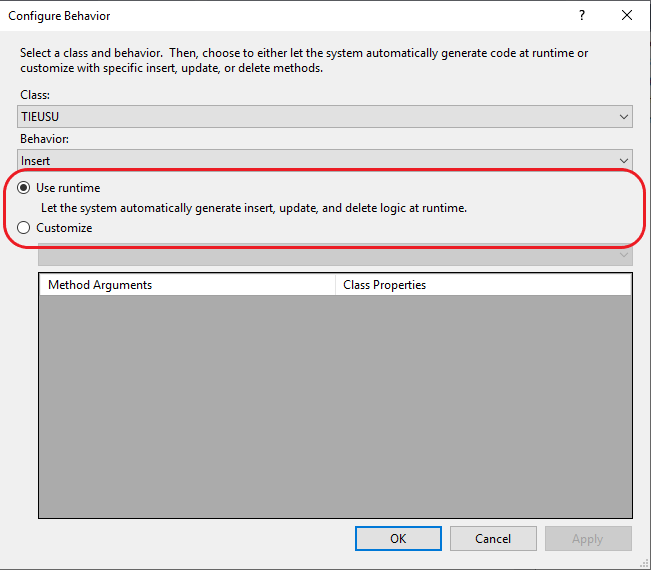
Chúng ta cũng sẽ làm điều tương tự với thủ tục InsertTIEUSU vừa được tạo:



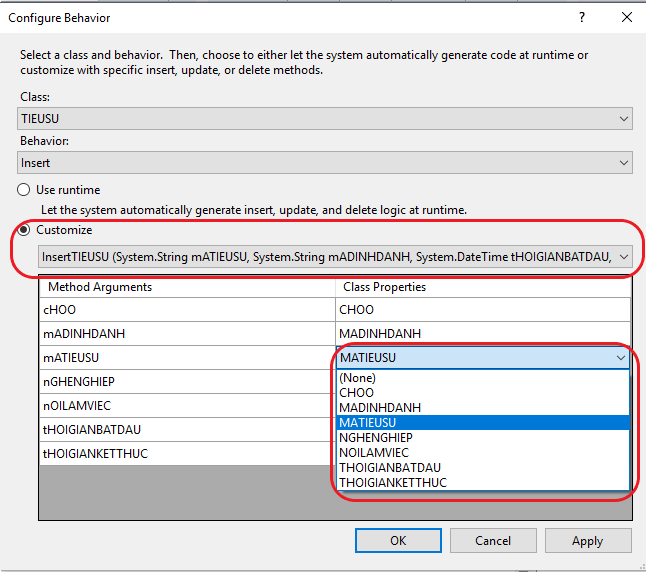
Bước cuối cùng là cấu hình lại để lớp truy cập dữ liệu dùng thủ tục SPROC khi chèn các đối tượng TIEUSU mới vào trong CSDL. Chúng ta có thể làm điều này bằng cách chọn lớp TIEUSU trong cửa sổ O/R designer, và sau đó chuyển đến bảng thuộc tính và nhấn nút 3 chấm (...) ở mục Insert để chọn thao tác thủ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



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



Trong trường hợp nếu chọn chế độ mặc định ("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 đối bằng cách nhấn chuột vào Customize và chọn thủ tục InsertTIIEUSU từ danh sách các SPROC:



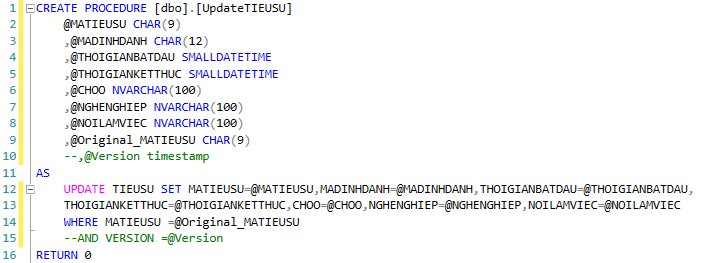
LINQ to SQL sẽ hiển thị các tham số của thủ tục mà ta đã chọn, và cho phép ánh xạ các thuộc tính của lớp TIEUSU và các tham số của InsertTIEUSU. Mặc nhiên, LINQ to SQL cũng tự động xác định các tham số tương ứng theo tên, tuy nhiên ta vẫn có thế sửa lại nếu muốn.

Nhấn vào nút Ok và kết thúc. Giờ đây bất cứ khi nào một đối tượng Order được thêm vào DataContext và phương thức SubmitChanges() được gọi, thủ tục InsertTIEUSU sẽ được thực thi thay cho câu lệnh SQL động.

**Quan trọng:** Mặc dù hiện tại chúng ta đã dùng SPROC để cập nhật, phương thức "OnValidate" của TIEUSU mà chúng ta đã tạo(trong bước 1) kiểm tra tính hợp lệ của đối tượng TIEUSU sẽ vẫn được thực thi trước khi bất kỳ thay đổi nào được thực hiện. Do vậy chúng ta sẽ có thể để xử lý và kiểm tra các quy tắc một cách rõ ràng, và có thể sử dụng lại mà không phụ thuộc vào việc chúng ta dùng SQL động hay dùng SPROC.

1. Thực hiện cập nhật dùng SPROC:

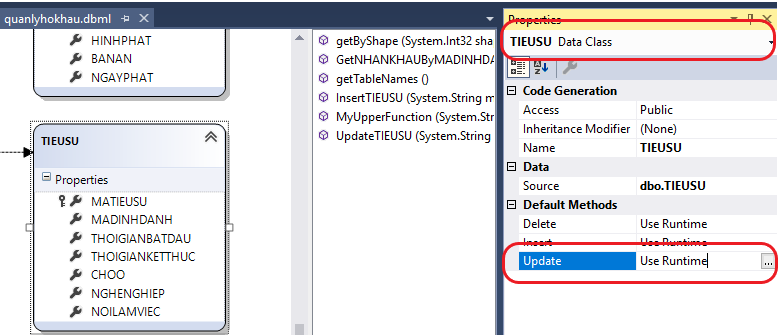
Giờ chúng ta sẽ sửa lại đối tượng TIEUSU để cho phép cập nhật bằng cách dùng SPROC, bắt đầu bằng cách tạo một SPROC tên "UpdateTIEUSU" như sau:



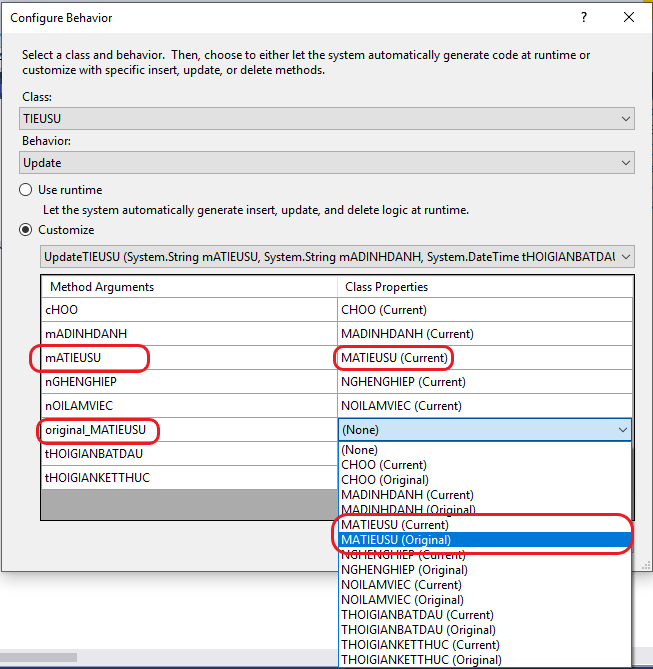
Chú ý ở trên, ngoài việc truyền giá trị cho tham số CustomerlD, ta cũng truyền một tham số khác có tên @Original\_CustomerID. Cột CustomerlD trong bảng Customers không phải là một cột tự tăng, và nó cỏ thế được chỉnh sửa trong thao tác cập nhật. Do vậy chúng ta sẽ phải truyền cả giá trị của CustomerlD cũ và CustomerlD mới để có thể cập nhật. Chúng ta sẽ xem cách ánh xạ các cột ngay sau đây.

*Chú ý: trong ví dụ trên, chúng ta có thể truyền một tham số có tên @Version (có kiều timestamp) vào cho SPROC. Cột này được thêm vào bảng TIEUSU để có thể xử lý việc tranh chấp khi các thao tác cập nhật được diễn ra đồng thời (optimistic concurrency). Nhớ rằng LINQ to SQL hỗ trợ đầy đủ optimistic concurrency, và cho phép ta 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 đọc dữ liệu. Trong ví dụ này là timestamp vì nó giúp viết code rõ ràng hơn.*

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



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



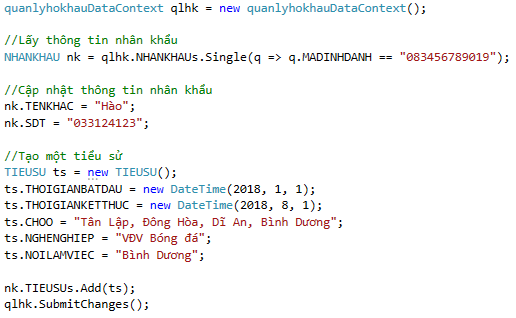
Khi ánh xạ các đối tượng TIEUSU vào các tham số của SPROC, chúng ta sẽ được hỏi lựa chọn giá trị mới (Current) hay các giá trị gốc (Original) - là các giá trị lấy ban đầu từ CSDL. Ví dụ, chúng ta sẽ cần gán giá trị thuộc tính TIEUSU.MATIEUSU "mới" vào cho tham số @MATIEUSU của SPROC, và TIEUSU.MATIEUSU "gốc" vào cho @original\_ MATIEUSU.

Khi nhấn "Ok" trên cửa sổ này, ta 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 TIEUSU và gọi SubmitChanges(), thủ tục UpdateTIEUSU sẽ được gọi thay cho câu lệnh SQL động.

**Quan trọng:** Trong trường hợp này, phương thức “OnValidate()” cũng sẽ được thực thi trước khi bất kỳ thay đổi nào được lưu lại hay "UpdateTIEUSU" được gọi. Chúng ta có thể thực hiện các quy tắc xử lý cũng như xác thực dữ liệu một cách rõ ràng, trực quan và có thể dùng chúng mà không phụ thuộc và việc chúng ta đang dùng câu lệnh SQL động hay SPROC.

1. Dùng Lớp DAO lần nữa:

Một khi đã cập nhật lớp truy cập dữ liệu (DAO) để dùng SPROC thay vì câu lệnh SQL động, chúng ta có thể chạy lại các câu lệnh tương tự các câu lệnh ta đã làm ở bước 2 để làm việc với các lớp mô hình dữ liệu:



Giờ đây việc thêm và cập nhật các đối tượng Order sẽ được thực thi thông qua thủ tục đã tạo thay vì dùng các câu SQL động. Các quy tắc kiểm tra cũng được thực thi hệt như trước đây, và các câu lệnh chúng ta đã dùng đế sử dụng các lớp mô hình dữ liệu cũng hoàn toàn tương tự.

* Ưu điểm của Việc dùng SPROC:
  + Dùng các tham số dạng output:

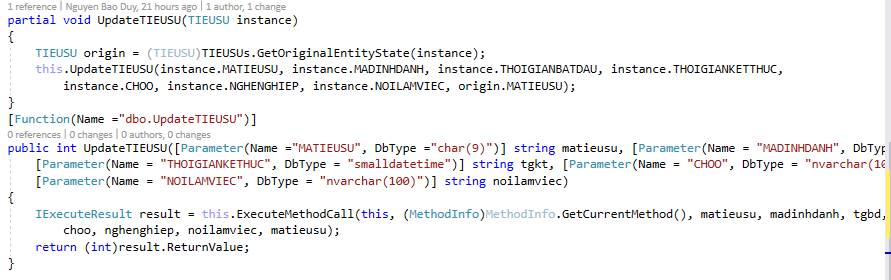
Trong Bước 3, đã có lưu ý về cách chúng ta có thế trả về giá trị OrderlD mới được tạo (đây là một cột tự tăng trong CSDL) bằng cách dùng một tham số dạng output. Chúng ta sẽ không bị giới hạn trong việc trả về chỉ các cột tự tăng - mà thật sự ta có thể trả về các giá trị cho bất kỳ tham số nào của SPROC. Chúng ta có thể dùng cách tiếp cận này cho cả trường hợp Insert và update. LINQ to SQL có thể lấy giá trị trả về và dùng nó để cập nhật giá trị của các thuộc tính của các đối tượng trong mô hình dữ liệu mà không cần thực thi thêm một câu truy vấn thứ 2 đế lấy các giá trị đã được tạo ra.

* + Hành động khi SPROC thả ra một lỗi:

Nếu một SPROC thả ra một lỗi khi thực hiện việc Insert/Update/Delete, LINỌ to SQL sẽ tự động hủy và rollback toàn bộ các thay đổi đã tạo ra trong transaction kết hợp với lời gọi SubmitChanges(). Điều này đảm bảo rằng dữ liệu của chúng ta sẽ luôn trong trạng thái đúng đắn.

* + Có thể code trực tiếp thay vì gọi SPROC thông qua O/R Designer:

Chúng ta có thể dùng LINQ to SQL designer đế ánh xạ các thao tác thêm/sửa/xóa vào các SPROC, hoặc cũng có thể thêm các phương thức partial vào lớp DataContext và viết lệnh gọi chúng. Đây là một ví dụ về cách viết các phương thức trong lớp partial của quanlyhokhauDataContext dùng UpdateTIEUSU để gọi một thủ tục:



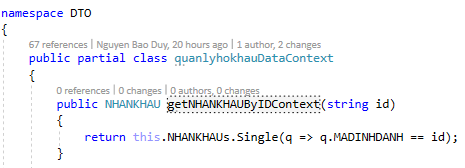
Đoạn lệnh ở trên thực ra chính là cái được tạo ra khi ta dùng O/R designer để ánh xạ SPROC và kết hợp nó với thao tác cập nhật đối tượng Customer. Ta có thể xem nó như điểm khởi đầu và sau đó tiếp tục thêm bất kỳ lệnh xử lý nào mình muốn (ví dụ: dùng giá trị trả về của SPROC để thả ra các exception tương ứng với mã lỗi nhận được, optimistic concurrency...)

### Thực thi các biểu thức SQL tùy biến:

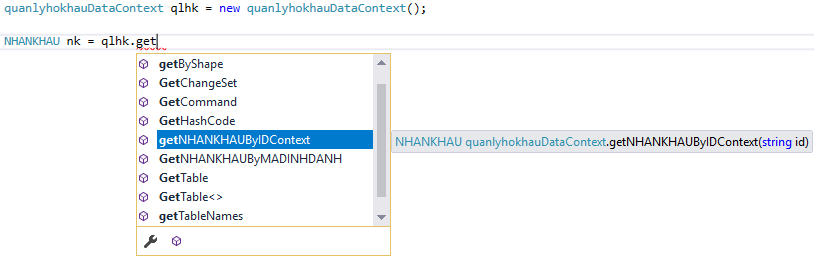
Khả năng của LINQ to SQL trong hỗ trợ làm việc với CSDL SQL đã được chứng minh là rất tốt, tuy nhiên, có cách nào để kiểm soát các câu SQL được dùng bởi LINQ to SQL mà không cần phải viết các SPROC hay không?

Ở Mục V.2.2 chúng ta đã tìm hiểu cách dùng LINQ để truy vấn mô hình dữ liệu ở trên và trả về một tập đối tượng biểu diễn các dòng/cột trong CSDL.

Ví dụ, ta có thể thêm một phương thức trợ giúp “getNHANKHAUByIDContext” vào lớp DataContext trong mô hình dữ liệu của chúng ta mà nó sẽ dùng cách truy vấn LINQ để trả về các đối tượng nhân khẩu từ CSDL:



Một khi đã định nghĩa phương thức LINQ như trên, ta có thể viết code để dùng nó lấy về các sản phẩm, và duyệt qua các kết quả trả về:



Khi biểu thức LINQ bên trong phương thức “getNHANKHAUByIDContext” được thực thi, trình quản lý LINQ to SQL sẽ tự động thực thi câu SQL động đế lấy về dữ liệu nhân khẩu và tạo ra danh sách các đối tượng NHANKHAU. Chúng ta có thế dùng trình debug đế xem cách biểu thức LINQ này thực thi.

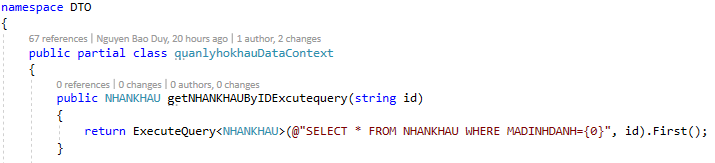
1. Dùng các câu truy vấn SQL tùy biến với LINQ to SQL:

Trong ví dụ mẫu ở trên chúng ta đã không viết bất kỳ câu lệnh SQL nào để truy vấn dữ liệu và lấy về các đối tượng có kiểu NHANKHAU. Thay vì vậy, LINQ to SQL sẽ tự động dịch biểu thức LINQ thành câu lệnh SQL và thực thi nó trong CSDL. Nhưng liệu nếu chúng ta muốn kiểm soát hoàn toàn câu lệnh SQL được thực thi với CSDL, và không muốn LINQ to SQL làm điều đó tự động được hay không? Một cách để làm điều này là dùng một SPROC giống như tại Mục V.2.6 và V.2.7. Một cách khác là dùng phương thức “ExcecuteQuery” trong lớp DataContext để thực thi một câu SQL do chúng ta cung cấp.

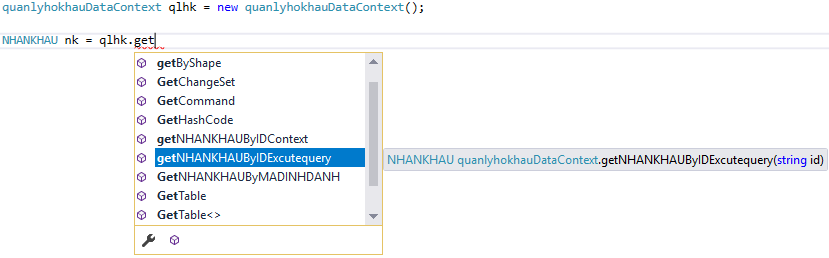
1. Dùng ExecuteQuery:

Phương thức ExecuteQuery nhận vào một câu SQL, cùng với một tập các tham số mà ta có thể dùng để tạo nên câu SQL. Bằng cách dùng nó, ta có thể thực thi bất kỳ câu lệnh SQL mình muốn với CSDL (kể cả các câu lệnh JOIN nhiều bảng).

Điều làm cho ExecuteQuery thực sự hữu dụng là nó cho phép ta chỉ ra kiểu dữ liệu mà nó trả về. Chúng ta làm điều này bằng cách truyền một đối tượng có kiểu mong muốn như một tham số của phương thức, hay dùng kiểu generic. Ví dụ, ta có thể thay đối phương thức getNHANKHAUByID bằng cách dùng phương thức ExecuteQuery thực thi một câu SQL với CSDL và trả về một tập đối tượng Product như sau:



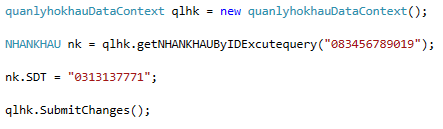
Chúng ta CÓ thể gọi getNHANKHAUByIDExcutequery() dùng cùng cách như trước:



Tuy nhiên, trong trường hợp này câu SQL tùy biến sẽ được gọi thay cho câu SQL động được tạo bởi biểu thức LINQ.

1. Tùy biến các biểu thức SQL và lưu vết (Tracking) các thao tác cập nhật:

Mặc nhiên, khi ta lấy về một mô hình dữ liệu dùng LINQ to SQL, nó sẽ lưu lại các thay đối đã thực hiện. Như gọi phương thức “SubmitChanges()” trên lớp DataContext, nó sẽ lưu lại các thay đổi vào CSDL (Mục V.2.3). Một trong nhưng tính năng nổi trội của ExecuteQuery là nó có thế kết hợp hoàn toàn vào quá trình lưu vết và cập nhật lại mô hình dữ liệu. Ví dụ, ta có thế viết đoạn lệnh dưới đây lấy về một nhân khẩu và cập nhật lại SDT của nhân khẩu đó:



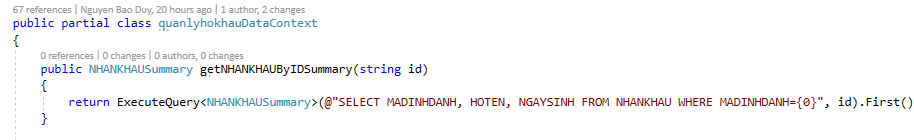
Bởi vì chúng ta đã chỉ ra rõ kiểu trả về của câu lệnh ExecuteQuery trong phương thức getNHANKHAUByIDExcutequery, do vậy LINQ to SQL sẽ biết cách dể dò ra các thay đổi trên các dối tượng NHANKHAU mà chúng ta trả về, và khi gọi “SubmitChanges()” trên đối tượng đó, chúng sẽ được lưu lại trong CSDL.

1. Tùy biến biểu thức SQL với các lớp trong phần mềm:

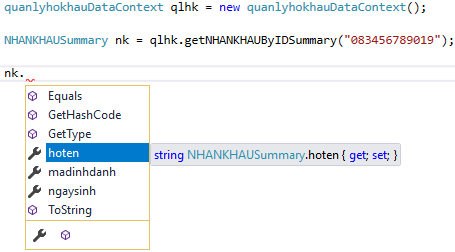
Phương thức ExecuteQuery() cho phép ta chọn bất kỳ lớp nào như kiểu trả về của câu truy vấn. Lớp này không nhất thiết phải được tạo ra bởi O/R designer, hay phải thừa kế từ bất kỳ class/interface nào. Ví dụ, ta có thể định nghĩa một lớp NHANKHAUSummary mới chứa các thuộc tính là tập con của nhân khẩu như dưới đây:



Chúng ta có thể sau đó tạo ra một phương thức tên là getNHANKHAUByIDSummary() trong lớp quanlyhokhauDataContext, nó sẽ trả về các kết quả dựa trên kiếu NHANKHAUSummary. Đế ý là câu SQL dưới đây chỉ yêu cầu các thuộc tính của Nhân khẩu nó cần - ExecuteQuery sẽ tự biết cách đưa các giá trị đó vào đối tượng NHANKHAUSummary mà nó sẽ trả về.

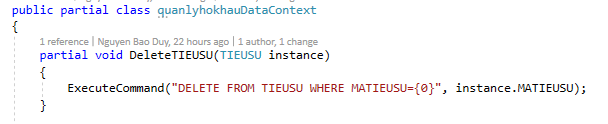


Sau đó chúng ta có thể dùng phương thức này để truy vấn và duyệt tập kết quả trả về:

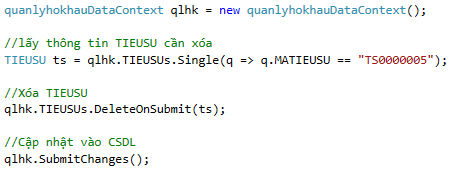


1. Tùy biến các câu SQL cho Insert/Update/Delete:

Thêm vào việc dùng các biểu thức SQL tùy biến để truy vấn, ta cũng có thế dùng chúng để thực hiện các thao tác như thêm/xóa/sửa. Chúng ta có thế làm được điều này bằng cách tạo ra các phương thức partial trong lớp DataContext tương ứng các thao tác Insert/Update/Delete cho thực thể mà chúng ta muốn thay đổi. Và chúng ta sau đó có thể dùng phương thức ExecuteCommand để thực thi các câu SQL cần thiết. Ví dụ, để thay thế hành vi Delete mặc nhiên cho lớp TIEUSU, ta có thể định nghĩa một phương thức partial DeleteTIEUSU như sau:



Và bây giờ, nếu ta viết đoạn code dưới đây để xóa một TIEUSU nào đó khỏi CSDL, LINQ to SQL sẽ gọi phương thức DeleteTIEUSU - và khi đó các câu SQL tùy biến sẽ được thực thi thay thế cho câu SQL được sinh ra tự động bởi LINQ to SQL:



## Tổng kết:

# LINQ to XML:

using **System.Linq; System.Xml; System.Xml.Linq**; **System.IO**;

XML đã được áp dụng rộng rãi như một cách để định dạng dữ liệu dạng văn bản trong nhiều ngữ cảnh. Ví dụ: ta có thể tìm thấy XML trên Web, trong các tệp cấu hình, trong các tệp Microsoft Office Word và trong cơ sở dữ liệu.

LINQ to XML là một cách tiếp cận được cập nhật, thiết kế lại để lập trình làm việc với XML. Nó cung cấp các khả năng sửa đổi tài liệu trong bộ nhớ theo Mô hình đối tượng tài liệu (DOM) và hỗ trợ các biểu thức truy vấn LINQ. Mặc dù các biểu thức truy vấn này khác về mặt cú pháp so với XPath, chúng cung cấp chức năng tương tự.

## Các khái niệm trong LINQ to XML:

### XML - eXtensible Markup Language

XML là viết tắt của từ eXtensible Markup Language, hay còn gọi là ngôn ngữ đánh dấu mở rộng do W3C đề nghị với mục đích tạo ra các ngôn ngữ đánh dấu khác. Đây là một tập hợp con đơn giản có thể mô tả nhiều loại dữ liệu khác nhau nên rất hữu ích trong việc chia sẻ dữ liệu giữa các hệ thống. Ví dụ khi ta xây dựng một ứng dụng bằng C# và một ứng dụng bằng PHP thì hai ngôn ngữ này không thể hiểu nhau, vì vậy ta sẽ sử dụng XML để trao đổi dữ liệu.

Tất cả những đặc tả dữ liệu XML đều phải tuân theo quy luật và cú pháp của nó nên hầu như các file XML đều được xử lý rất nghiêm khắc trong việc biên dịch. Tuy nhiên công nghệ này cần phải được xử lý lại bởi vì trong quá trình thao tác và truyền dữ liệu nó có tỉ lệ sai sót lên tới 5% - 7%. Con số này không cao nhưng cũng rất đáng để cân nhức khi sử dụng.

Điển hình nhất là ngôn ngữ đánh dấu siêu văn bản HTML sử dụng cú pháp của XML để tạo nên và nó có các bộ phần tử và thuộc tính không mềm dẻo nên chỉ có tác dụng trong việc trình bày dữ liệu trên trình duyệt Browser.

XML được xây dựng dựa vào cấu trúc NODE lồng nhau, mỗi node sẽ có một thẻ mở và một thẻ đóng như sau:

<nodename>content </nodename>

Trong đó:

* <nodename> là thẻ mở, tên của thẻ này do người dùng tự định nghĩa.
* </nodename> là thẻ đóng, tên của thẻ này phải trùng với tên của thẻ mở.
* content là nội dung của thẻ này

Ví dụ như lưu trữ tên database thì cấu trúc như sau:

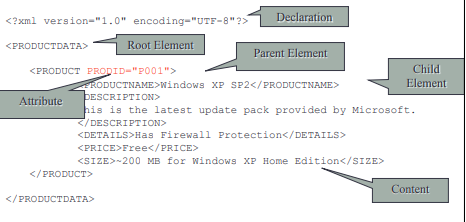
<database>QLHK</database>

Chúng ta hoàn toàn có thể bổ sung các thuộc tính vào các thẻ XML bằng cách sử dụng cú pháp sau:

<nodename ten\_thuoc\_tinh="giá trị">content</nodename>

Ví dụ, viết lại cấu trúc Database như trên nhưng tên database để trong thuộc tính name:

<database name="QLHK">content</database>



#### Khai báo header(Chỉ thị xử lý):

Trên đầu mỗi file XML bạn phải khai báo một thẻ để thông báo version XML đang sử dụng (thường là version 1.0), và còn có thể chứa các thông tin về mã hóa ký tự hoặc các phụ thuộc bên ngoài khác (sẽ tìm hiểu sau). Giá trị của encoding (kiểu mã hóa ký tự) thuộc một trong các định dạng sau: UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift\_JIS, EUC-JP.

Cú pháp của thẻ chỉ thị xử lý như sau:

<?xml version="1.0" encoding="UTF-8"?>

Như vậy với các ví dụ trên thì cấu trúc đúng sẽ phải là:

<?xml version="1.0" encoding="UTF-8"?>

<database name="QLHK">content</database>

#### Root node:

Mỗi tài liệu XML nên có một thẻ ngoài cùng và ta gọi thẻ này là root node. Thẻ này sẽ khai báo tên chính của tài liệu XML.

Ví dụ mình cần lưu trữ danh sách domain thì có thể viết như sau:

<?xml version="1.0" encoding="UTF-8"?>

<database name="QLHK">

    <table name="NHANKHAU"> </table>

    <table name="SOHOKHAU"> </table>

</database>

Không có một quy tắc đặt tên nào cả mà quy tắc do lập trình viên đặt ra, tuy nhiên chúng ta nên đặt tên sao cho ngữ nghĩa phù hợp với nội dung của file.

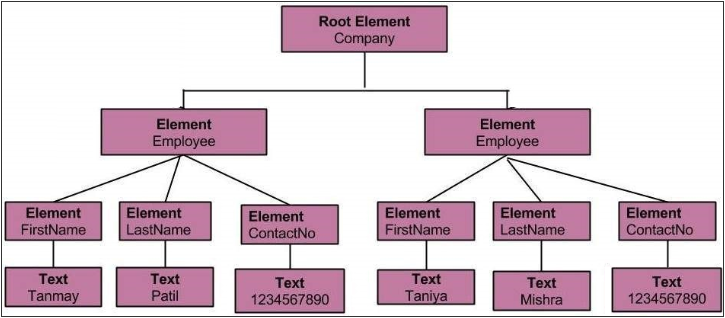
#### Cấu trúc cây trong XML:

Một tài liệu XML được tạo bởi các thẻ (XML element) và chúng có thể được tổ chức theo một cấu trúc cây thư mục, cách tổ chức này gọi là Nested Elements trong XML. Vậy tổ chức như thế nào để có thể lưu trữ được dữ liệu trong thế giới thực?

1. Cấu trúc cây trong XML:

Từ ví dụ cuối phần c, ta có thể rút ra kết luận rằng các thẻ XML có thể lồng lên nhau, thẻ ngoài ta gọi là thẻ cha và các thể bên trong ta gọi là thẻ con.

Ví dụ: Sơ đồ cấu trúc cây tổ chức lưu trữ thông tin nhân viên trong một công ty



Với sơ đồ này ta sẽ thực hiện từng bước tạo tài liệu XML như sau:

Bước 1: Thẻ ngoài cùng root là company

<?xml version="1.0" encoding="UTF-8"?>

<company>

</company>

Bước 2: Bên trong thẻ company có hai thẻ employee.

<?xml version="1.0" encoding="UTF-8"?>

<company>

<employee>

</employee>

<employee>

</employee>

</company>

Bước 3: Bên trong thẻ employee đầu tên gồm ba thẻ firstname, lastname và contactno tương đương với ba giá trị như sau:

* firstname => Tanmay
* lastname => Patil
* contactno => 123456789

<?xml version="1.0" encoding="UTF-8"?>

<company>

<employee>

<firstname>

Tanmay

</firstname>

<lastname>

Patil

</lastname>

<contactno>

123456789

</contactno>

</employee>

<employee>

</employee>

</company>

Bước 4: Tương tự cho nội dung bên trong thẻ employee thứ hai.

<?xml version="1.0" encoding="UTF-8"?>

<company>

<employee>

<firstname>

Tanmay

</firstname>

<lastname>

Patil

</lastname>

<contactno>

123456789

</contactno>

</employee>

<employee>

<firstname>

Taniya

</firstname>

<lastname>

Mishra

</lastname>

<contactno>

123456789

</contactno>

</employee>

</company>

Như vậy là ta đã đặc tả xong cấu trúc XML lưu trữ dữ liệu ứng dụng quản lý nhân viên trong công ty.

Câu hỏi đặt ra là nếu ta lưu trữ thêm một nhân viên nữa thì phải làm thế nào? Rất đơn giản ta chỉ việc tạo thêm một thẻ employee và thêm thông tin là được.

<?xml version="1.0" encoding="UTF-8"?>

<company>

<employee>

<firstname>

Tanmay

</firstname>

<lastname>

Patil

</lastname>

<contactno>

123456789

</contactno>

</employee>

<employee>

<firstname>

Taniya

</firstname>

<lastname>

Mishra

</lastname>

<contactno>

123456789

</contactno>

</employee>

<employee>

<firstname>

Cuong

</firstname>

<lastname>

Nguyen

</lastname>

<contactno>

0979306603

</contactno>

</employee>

</company>

1. Cấu trúc cây nhiều cấp trong XML:

Lấy ví dụ ở phần thứ nhất và bổ sung thêm yêu cầu sau: Môi nhân viên lưu trữ thêm danh sách người thân của nhân viên đó (chỉ cần lưu trữ tên, mối quan hệ).

Trước tiên ta cần tổ chức XML lưu trữ người thân đã. Giả sử ta sẽ lưu trữ dạng sau:

<family>

<person>

<name>Nguyễn Sơn</name>

<relationship>Cha</relationship>

</person>

<person>

<name>Lê Thị Sửu</name>

<relationship>Mẹ</relationship>

</person>

<person>

<name>Nguyễn Văn Trường</name>

<relationship>Anh trai</relationship>

</person>

</family>

Ráp vào bài toán thứ nhất ta sẽ có cấu trúc XML sau:

<?xml version="1.0" encoding="UTF-8"?>

<company>

<employee>

<firstname>

Tanmay

</firstname>

<lastname>

Patil

</lastname>

<contactno>

123456789

</contactno>

<family>

<person>

<name>Nguyễn Sơn</name>

<relationship>Cha</relationship>

</person>

<person>

<name>Lê Thị Sửu</name>

<relationship>Mẹ</relationship>

</person>

<person>

<name>Nguyễn Văn Trường</name>

<relationship>Anh trai</relationship>

</person>

</family>

</employee>

</company>

Như vậy mỗi lần thêm nhân viên thì chỉ việc bổ sung thẻ employee và muốn thêm người thân thì chỉ việc bổ sung thẻ person.

#### Element trong XML:

Như ta biết một tài liệu XML sẽ chứa nhiều thẻ XML, các thẻ XML bao gồm tất cả mọi thứ từ thẻ khai báo trên header cho đến thẻ Root, các thẻ con, ... cho đến thẻ cuối cùng trong tài liệu.

Một thẻ XML dùng để chứa dữ liệu dạng đơn giản (number, string) cho đến phức tạp (thẻ XML chứa thẻ XML).

Ví dụ ta cần viết một file XML để lưu trữ danh sách sinh viên gồm các thông tin:

* Tên sinh viên (tensv)
* Năm sinh (namsinh)
* Giới tính (gioitinh)

Lúc này ta sẽ có ba thẻ chứa dữ liệu đơn giản đó là tensv, namsinh và gioitinh. Có một thẻ chứa dữ liệu phức tạp đó là thẻ sinhvien. Nhưng chưa dừng lại ở đó, như ở phần c chúng ta cần xác định thêm một thẻ Root nữa. Phân tích kỹ thì ta thấy mỗi sinh viên sẽ được mô tả bởi thẻ sinhvien và tài liệu sẽ chứa nhiều sinh viên, vì vậy mình sẽ đặt tên cho thẻ root là ds-sinhvien.

Như vậy dưới đây là cấu cây XML của chúng ta.

<?xml version="1.0" encoding="UTF-8"?>

<ds-sinhvien>

<sinhvien>

<tensv>[Data]</tensv>

<namsinh>[Data]</namsinh>

<gioitinh>[Data]</gioitinh>

</sinhvien>

</ds-sinhvien>

1. Thẻ XML rỗng và rút gọn thẻ đóng:

Một thẻ XML có thể chứa dữ liệu hoặc không chứa dữ liệu cũng được. Nếu không có dữ liệu thì ta để trống.

Ví dụ:

<?xml version="1.0" encoding="UTF-8"?>

<ds-sinhvien>

<sinhvien></sinhvien>

</ds-sinhvien>

Trường hợp này ta có thể rút gọn thẻ đóng bằng cách sau:

<?xml version="1.0" encoding="UTF-8"?>

<ds-sinhvien>

<sinhvien/>

</ds-sinhvien>

Nghĩa là ta sẽ đóng ngay thẻ mở luôn.

Một ví dụ điển hình của trường hợp thẻ XML rỗng như sau: Giả sử cần bổ sung thêm một thuộc tính mô tả (mota) dùng để mô tả sinh viên ở ví dụ trên. Lúc này cấu trúc XML sẽ như sau:

<?xml version="1.0" encoding="UTF-8"?>

<ds-sinhvien>

<sinhvien>

<tensv>[Data]</tensv>

<namsinh>[Data]</namsinh>

<gioitinh>[Data]</gioitinh>

<mota>[Data]</mota>

</sinhvien>

</ds-sinhvien>

Bây giờ mình sẽ lưu hai sinh viên thì cấu trúc XML sẽ như sau:

<?xml version="1.0" encoding="UTF-8"?>

<ds-sinhvien>

<sinhvien>

<tensv>Nguyễn Văn Cường</tensv>

<namsinh>1990</namsinh>

<gioitinh>Nam</gioitinh>

<mota>Sinh viên đẹp zai thích viết tutorials.</mota>

</sinhvien>

<sinhvien>

<tensv>Vũ Thị Thu Tình</tensv>

<namsinh>1992</namsinh>

<gioitinh>Nữ</gioitinh>

<mota></mota>

</sinhvien>

</ds-sinhvien>

Như vậy ở sinh viên thứ hai phần mô tả bị trống do không có mô tả cho sinh viên đó. Lúc này ta có thể ghi gọn cho thẻ mota như sau:

<sinhvien>

<tensv>Vũ Thị Thu Tình</tensv>

<namsinh>1992</namsinh>

<gioitinh>Nữ</gioitinh>

<mota/>

</sinhvien>

1. Một số quy tắc liên quan đến XML Element:

Đặt tên thẻ:

* Tên thẻ phải bắt đầu là một chữ cái hoặc ký tự gạch dưới.
* Tên thẻ không được bắt đầu bằng XML (hoặc Xml, xMl, xmL) vì nó giống với thẻ khai báo header.
* Tên thẻ không được chứa khoảng trắng.
* Tên thẻ có thể chứa chữ cái, chữ số, dấu gạch dưới, dấu gạch ngang, dấu chấm và dấu hai chấm.

Cấu trúc thẻ:

* Mỗi thẻ mở đều phải có thẻ đóng và chúng phải tuân theo đúng thứ tự.
* Bạn có thể dùng thẻ đóng rút gọn nếu như thẻ đó rỗng

Naming styles:

Không có một chuẩn đặt tên thẻ nào trong XML cả nhưng người ta đề xuất ra một số cách đặt tên như sau:

| **Style** | **Example** | **Description** |
| --- | --- | --- |
| Lower case | <firstname> | Tất cả đều chữ in thường |
| Upper case | <FIRSTNAME> | Tất cả đều chữ in hoa |
| Underscore | <first\_name> | Các từ cách nhau bởi dấu gạch dưới |
| Pascal case | <FirstName> | Viết hoa ký tự đầu tiên trong các từ |
| Camel case | <firstName> | Viết hoa ký tự đầu tiên trong từ thứ hai trở đi. |

Tên thẻ nên là một danh từ, nên đặt tên bằng tiếng Anh vì nó đơn giản và là ngôn ngữ chuẩn trên thế giới.

#### Các thuộc tính (Attributes) trong XML:

Trong các thẻ HTML chúng ta sẽ có các thuộc tính như href, id, class, src, ... Các thuộc tính này chỉ có tác dụng đặc biệt trong tài liệu HTML chứ đối với XML nó không có tác dụng gì, chỉ là các thuộc tính bình thường.

1. XML Attributes là gì:

XML Attributes là những dữ liệu được khai báo báo bên trong thẻ mở theo cú pháp:

<element\_name property\_name="property\_value">

</element\_name>

Có thể sử dụng cặp thẻ nháy kép "" hoặc nháy dơn đều được ' '. Tuy nhiên lời khuyên là ta nên sử dụng cặp nháy kép.

Mỗi thẻ XML (XML element) có thể không có thuộc tính nào hoặc có nhiều thuộc tính tùy vào cách định nghĩa của lập trình viên.

Ví dụ: Trường hợp có nhiều thuộc tính

<?xml version="1.0" encoding="UTF-8"?>

<coders>

<person skill="PHP Javascript jQuery" master\_at="PHP">

<name>Nguyễn Văn Cường</name>

<website>Dttv.com</website>

</person>

</coders>

Chúng ta có thể sử dụng thuộc tính để thay thế cho các thẻ XML như ví dụ dưới đây:

Cách 1: Sử dụng thuộc tính

<?xml version="1.0" encoding="UTF-8"?>

<coders>

<person skill="PHP Javascript jQuery" master\_at="PHP">

<name>Nguyễn Văn Cường</name>

<website>Dttv.com</website>

</person>

</coders>

Cách 2: Sử dụng thẻ XML

<?xml version="1.0" encoding="UTF-8"?>

<coders>

<person>

<master\_at>PHP</master\_at>

<skill>PHP Javascript jQuery</skill>

<name>Nguyễn Văn Cường</name>

<website>Dttv.com</website>

</person>

</coders>

1. XML Attributes và XML Properties:

Như ở ví dụ so sánh cách sử dụng XML Attributes thay thế cho XML Elements ở trên thì chúng ta đã thấy được sự chuyển đổi đơn giản giữa hai khái niệm này, tuy nhiên vẫn còn một số lưu ý nữa mà chúng ta nên đọc để hiểu thêm.

*XML Multi value:*

Thứ nhất: Mỗi một thuộc tính XML chỉ có thể chứa một giá trị duy nhất cho dù ta nhập gì bên trong nó đi nữa thì XML vẫn tính là một giá trị, bởi vì XPath sẽ không tính nó là nhiều giá trị.

Thứ hai: Các thuộc tính XML chỉ chứa giá trị dạng text, number chứ không thể chứa một thẻ XML khác.

Từ hai tính chất trên ta thấy khi sử dụng thuộc tính XML thì rất khó mở rộng tài liệu, ví dụ sau này ta cần bổ sung thông tin là con của thuộc tính nào đó thì không thể làm được, nhưng với element thì hoàn toàn làm được.

Ví dụ: Trường hợp attributes khó nâng cấp và mở rộng

<?xml version="1.0" encoding="UTF-8"?>

<coders>

<person master\_at="PHP" skill="PHP Javascript jQuery"

name="Nguyễn Văn Cường" website="Dttv.com">

</person>

</coders>

Giả sử giờ mình muốn lấy họ của lâp trình viên thì phải xử lý tách chuỗi, Nhưng với cách thiết kế dưới đây thì không cần:

<?xml version="1.0" encoding="UTF-8"?>

<coders>

<person>

<master\_at>PHP</master\_at>

<skill>PHP Javascript jQuery</skill>

<name>

<firstname>Cường</firstname>

<middlename>Văn</middlename>

<lastname>Nguyễn</lastname>

</name>

<website>Dttv.com</website>

</person>

</coders>

Vừa dễ dàng mở rộng mà nhìn lại trực quan hơn

Metadata trong XML:

Phần này sẽ chứng minh công dụng của Attribute trong XML. Chúng ta thường sử dụng Attributes để lưu trữ những thông tin chính như là khóa chính của thẻ XML đó.

Ví dụ:

<?xml version="1.0" encoding="UTF-8"?>

<coders>

<person id="1">

<master\_at>PHP</master\_at>

<skill>PHP Javascript jQuery</skill>

<name>

<firstname>Cường</firstname>

<middlename>Văn</middlename>

<lastname>Nguyễn</lastname>

</name>

<website>Dttv.com</website>

</person>

<person id="2">

<master\_at>Javascript</master\_at>

<skill>PHP Javascript AngularJS</skill>

<name>

<firstname>Tình</firstname>

<middlename>Thị Thu</middlename>

<lastname>Vũ</lastname>

</name>

<website>Dttv.com</website>

</person>

</coders>

Ngoài cách sử dụng thuộc tính ra thì ta cũng có thể tạo thêm một thẻ id để lưu trữ, tuy nhiên lời khuyên cho bạn nên sử dụng thuộc tính cho trường hợp này.

#### XML và các hệ quản trị CSDL:

XML cũng là một dạng lưu trữ dữ liệu nhưng có điểm khác là nó lưu trữ trên file nên đối với những ứng dụng lớn vài triệu records thì rất khó để đọc và truy xuất. Nhưng khi ta sử dụng các hệ quản trị CSDL thì dễ dàng hơn bởi nó cung cấp cho ta ngôn ngữ T-SQL giúp việc xử lý truy vấn rất hiệu quả.

Vậy làm cách nào để chuyển một cấu trúc lưu trữ trong XML sang một hệ quảng trị CSDL như SQL?

1. Phân tích một file XML:

Trước tiên ta có một ví dụ như sau: Cần lưu trữ danh sách sinh viên của một trường đại học, mỗi sinh viên cần lưu các thông tin:

* Tên sinh viên (TenSV)
* Mã SV (MaSV)
* Năm sinh (NamSinh)
* Giới tính (GioiTinh)
* Mã lớp (MaLop)

Phân tích bài toán để lưu trữ trong XML như sau:

Mỗi sinh viên sẽ được bao bọc bởi thẻ SinhVien.

Bên trong thẻ SinhVien là các thẻ TenSV, MaSV, NamSinh, GioiTinh, MaLop

Cần một thẻ root bên ngoài để khai báo project và ta đặt nó là thẻ DS\_SinhVien

Và đây là file XML của chúng ta:

<?xml version="1.0" encoding="UTF-8"?>

<DS\_SinhVien>

<SinhVien>

<MaSV></MaSV>

<TenSV></TenSV>

<NamSinh></NamSinh>

<GioiTinh></GioiTinh>

<MaLop></MaLop>

</SinhVien>

</DS\_SinhVien>

Nếu có nhiều sinh viên thì ta chỉ cần copy thẻ SinhVien là được:

<?xml version="1.0" encoding="UTF-8"?>

<DS\_SinhVien>

<SinhVien>

<MaSV></MaSV>

<TenSV></TenSV>

<NamSinh></NamSinh>

<GioiTinh></GioiTinh>

<MaLop></MaLop>

</SinhVien>

<SinhVien>

<MaSV></MaSV>

<TenSV></TenSV>

<NamSinh></NamSinh>

<GioiTinh></GioiTinh>

<MaLop></MaLop>

</SinhVien>

</DS\_SinhVien>

1. Phân tích sang SQL:

Với MySQL thì ta sẽ lưu mỗi sinh viên là một record, có nhiều sinh viên thì cứ bổ sung một record là được.

Lúc này nhìn vào file XML ở phần 1 ta có thể rút ra kết luận như sau: Dữ liệu trong thẻ SinhVien tương ứng với một record trong SQL, khóa chính là thẻ MaSV và khóa ngoại là MaLop.

Cấu trúc table như sau:

* Tên table: DS\_SinhVien
* Các fields: MaSV, TenSV, NamSinh, GioiTinh, MaLop

Tất cả được mô phỏng trong hình dưới đây:



Lưu ý ta có thể đặt tên database và tên fields khác với file XML, bởi vì nếu đặt tên giống thì khi viết API dễ dàng bị lộ tên table, không đảm bảo tính bảo mật dữ liệu.

#### XML Namespace:

Để tạo một namespace thì ta sẽ khai báo một thuộc tính xmlns:uri.

Trong đó xmlns là viết tắt của từ XML namespace.

uri là đường dẫn URL của namespace, có thể là một địa chỉ nào đó trên internet hoặc một địa chỉ nào đó đều được nhưng phải đảm bảo rằng nó là duy nhât trong file tài liệu XML của chúng ta.

Ví dụ:

<?xml version="1.0" encoding="UTF-8"?>

<ShopOrder>

<Order>

<cus:Customer xmlns:cus="https://dttv.com/customer">

<cus:Title>Nguyễn Văn Cường</cus:Title>

<cus:Address>Buôn Ma Thuột - Đăklăk</cus:Address>

</cus:Customer>

</Order>

</ShopOrder>

Như vậy các thẻ XML chúng ta cũng phải bổ sung tên của namespace theo cú pháp namespace\_name:element\_name.

Liên tưởng tới truy vấn T-SQL:

Trong ngôn ngữ T-SQL thì để giải quyết vấn đề này ta sẽ sử dụng lệnh as (alias) và dấu chấm để tránh sự nhầm lẫn này.

Ví dụ: Có hai bảng tin tức và chuyên mục có trùng tên field id. Lúc này để lấy danh sách tin tức và chuyên mục của tin tức đó kèm id của cả hai thì ta phải viết câu SQL dạng:

SELECT p.id, p.title, c.id, c.title

FROM products as p JOIN categories as c on p.cate\_id = c.id

Namespace mặc định:

Nếu tài liệu của chúng ta chỉ sử dụng một namespace thì ta có thể khai báo namespace mặc định cho nó bằng cách chỉ sử dụng từ khóa xmlns và bỏ đi tên của namespace.

Ví dụ:

<?xml version="1.0" encoding="UTF-8"?>

<ShopOrder xmlns="https://dttv.com/customer">

<Order>

<Customer>

<Title>Nguyễn Văn Cường</Title>

<Address>Buôn Ma Thuột - Đăklăk</Address>

</Customer>

<Customer>

<Title>Nguyễn Văn Kính</Title>

<Address>Buôn Ma Thuột - Đăklăk</Address>

</Customer>

</Order>

</ShopOrder>

Nhưng giả sử chúng ta có hai namespace mặc định trong một tài liệu XML như trường hợp dưới đây:

<?xml version="1.0" encoding="UTF-8"?>

<ShopOrder>

<Order>

<Customer xmlns="https://dttv.com/customer">

<Title>Nguyễn Văn Cường</Title>

<Address>Buôn Ma Thuột - Đăklăk</Address>

</Customer>

<Product xmlns="https://dttv.com/product">

<Title>Dép thái cao cấp</Title>

<Qua>20</Qua>

<Price>200.000 vnđ</Price>

</Product>

</Order>

</ShopOrder>

Trường hợp này nếu ta có nhiều đơn hàng thì phải tạo nhiều namespace mặc định. Đều này khá phiền toái nếu ta có 1000 đơn hàng phải không nào? Để giải quyết vấn đề này thì ta khai báo namespace ngay tại thẻ Root.

<?xml version="1.0" encoding="UTF-8"?>

<ShopOrder

xmlns:cus="https://dttv.com/customer"

xmlns:pro="https://dttv.com/product">

<Order>

<cus:Customer>

<cus:Title>Nguyễn Văn Cường</cus:Title>

<cus:Address>Buôn Ma Thuột - Đăklăk</cus:Address>

</cus:Customer>

<pro:Product>

<pro:Title>Dép thái cao cấp</pro:Title>

<pro:Qua>20</pro:Qua>

<pro:Price>200.000 vnđ</pro:Price>

</pro:Product>

</Order>

</ShopOrder>

Trên là những vấn đề ta cần biết về namespace trong XML.

#### CDATA trong XML:

CDATA được sử dụng để giữ nguyên văn bản trong các thẻ XML, đây là một chức năng khá quan trọng giúp file XML không bị lỗi về cú pháp.

1. CDATA trong XML:

Một tài liệu XML được xây dựng bởi các thẻ XML do lập trình viên tự định nghĩa ra, và đương nhiên lập trình viên phải tuân thủ các quy tắc mà XML đề ra (phần E) Tuy nhiên có một số trường hợp dữ liệu chứa trong các thẻ XML lại có những ký tự đặc biệt dẫn đến tài liệu XML bị đảo lộn.

Ví dụ:

<?xml version="1.0" encoding="UTF-8"?>

<Tutorials>

<Post>

<Title>

Học CDATA trong XML miễn phí

</Title>

<Code>

<?xml version="1.0" encoding="UTF-8"?>

<Persons>

<Person>

Nguyễn Văn Cường

</Person>

</Persons>

</Code>

</Post>

</Tutorials>

Có thể thấy nội dung bên trong thẻ Code là một đoạn mã XML khác, có nghĩa đây là trường hợp giá trị của thẻ XML lại là một đoạn mã XML khác. Nếu chạy lên sẽ bị lỗi như hình dưới đây:



Để khắc phục trường hợp này ta sẽ sử dụng CDATA như sau:

<?xml version="1.0" encoding="UTF-8"?>

<Tutorials>

<Post>

<Title>

Học CDATA trong XML miễn phí

</Title>

<Code>

<![CDATA[

<?xml version="1.0" encoding="UTF-8"?>

<Persons>

<Person>

Nguyễn Văn Cường

</Person>

</Persons>

]]>

</Code>

</Post>

</Tutorials>

Như vậy cú pháp của CDATA là <![CDATA[nội dung]]>.

1. Sử dụng Special Character trong XML:

Special character (đôi lúc ta gọi là thực thể) là những ký tự đặc biệt được chuyển đổi thành mỗi dãy các chữ cái và dấu &;.

Như ở trong phần một ta sử dụng CDATA để khắc phục lỗi nội dung XML chứa kí tự đặc biệt. Nhưng cũng có một cách khác là bạn chuyển những ký tự đặc biệt đó sang Special characters như bảng dưới đây.

| **Thực thể** | **Mô tả** |
| --- | --- |
| &apos; | Dấu nháy đơn ' |
| &amp; | Dấu & |
| &gt; | Dấu > |
| &lt; | Dấu < |
| &quot; | Dáu nháy kép " |

Ví dụ: Viết lại ví dụ trên bằng cách sử dụng Special character.

<?xml version="1.0" encoding="UTF-8"?>

<Tutorials>

<Post>

<Title>

Học CDATA trong XML miễn phí

</Title>

<Code>

&lt;?xml version="1.0" encoding="UTF-8"?&gt;

&lt;Persons&gt;

&lt;Person&gt;

Nguyễn Văn Cường

&lt;/Person&gt;

&lt;/Persons&gt;

</Code>

</Post>

</Tutorials>

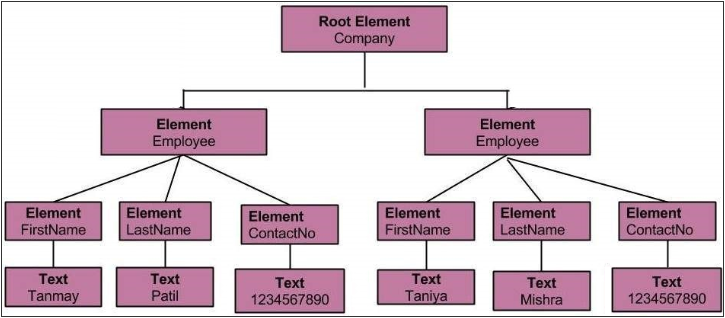
Chạy file XML này lên và mọi thứ đều ổn.

Cả hai cách trên đều dùng để xử lý nội dung bên trong thẻ XML, tuy nhiên người ta vẫn hay sử dụng CDATA hơn bởi vì nó đơn giản nhưng lại rất hiệu quả.

### Mô hình đối tượng tài liệu - Document Object Model (DOM):

DOM - Document Object Model ("Mô hình Đối tượng Tài liệu"), là một giao diện lập trình ứng dụng (API). DOM thông thường có có dạng một cây cấu trúc dữ liệu, được dùng để truy xuất các tài liệu dạng HTML và XML. Khái niệm này được định nghĩa bởi tổ chức W3C và sử dụng chung cho tất cả các ngôn ngữ có áp dụng DOM.

Mô hình DOM độc lập với hệ điều hành và dựa theo kỹ thuật lập trình hướng đối tượng để mô tả tài liệu.

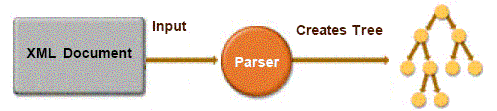


#### Đặc điểm của DOM:

* Các đặc điểm của mô hình DOM:
* Truy cập tài liệu XML như là một cấu trúc cây
* Được bao gồm hầu hết các nút element và các nút text(trừ Header, text nằm ngoài Root Element)
* Có thể "rà xoát" (Traversing) cây từ sau ra trước
* Đòi hỏi bộ nhớ lớn hơn
  + Khá nặng nề trong việc tải và lưu trữ
* Sử dụng nó khi rà xoát và hiệu chỉnh cây

#### Hành động của DOM và cấu trúc cây tạo thành:

Hình minh họa dưới cho bạn thấy đầu vào là một tài liệu XML được bộ phân tích bởi mô hình DOM và một sẽ cây được tạo ra trong bộ nhớ mang thông tin của tài liệu đó .Việc phân tích tài liệu XML bây giờ đưa về phân tích ,sử lý các nút của cây.’



Cấu trúc của một cây DOM:

* Tài liệu XML được hình dung như là một cây
* Một cây được làm từ các nút cây(nodes), Có 12 kiểu Node:
  + Document (Mô tả một nút lớn nhất đó là toàn bộ tài liệu XML)
  + DocumentFragment (Một đoạn tài liệu XML)
  + Element
  + Attr (Nút thuộc tính)
  + Text (Nút chứa text)
  + Comment (Ghi chú trong tài liệu XML)
  + ProcessingInstruction (Tương ứng với chỉ lệnh trong XML)
  + DocumentType (Định nghĩa XML)
  + Entity (Tương ứng với thực thể trong XML)
  + EntityReference (Tương ứng với các thực thể tham chiếu trong XML)
  + CDATASection (Tương ứng với các phân đoạn trong XML)
  + Notation (Tương ứng với các chú thích NOTATION trong XML)
* Các nút cây có thể chứa các nút cây khác (Phụ thuộc vào kiểu nút cây là gì)
  + Nút cha gồm có các nút con

Phân cấp cây DOM:

* Một nút Document bao gồm.
* Một nút Element (Đây chính là Element gốc của tài liệu)
* Một hoặc nhiều các nút chỉ lệnh (ProcessingInstruction)
* Một hoặc nhiều nút Comment
* Một nút Element có thể chứa
* Các nút Element khác
* Một hoặc nhiều nút Text
* Một hoặc nhiều nút thuộc tính (Attr)
* Một nút thuộc tính (Attr) chứa
* Một nút Text

### Namespace System.Xml và đọc ghi file XML trong C#:

Cấu trúc một số class cơ bản trong namespace System.Xml:

|  |  |
| --- | --- |
| Tên lớp | Giải thích |
| XmlReader | Một lớp đọc - trừu tượng nhanh và non-cached dữ liệu XML. |
| XmlWriter | Một lớp viết - trừu tượng nhanh và non-cached dữ liệu XML trong một dòng hoặc định dạng file. |
| XmlTexReader | Mở rộng của XmlReader. Cung cấp chuỗi truy cập nhanh dữ liệu XML. |
| XmlTexWriter | Mở rộng của XmlWriter. Phát nhanh các dòng XML. |
| XmlNode | Một lớp trừu tượng miêu tả một nút đơn trong một tài liệu XML. Lớp cơ sở cho các lớp khác trong namespace XML. |
| XmlDocument | Mở rộng của XmlNode. Nó cung cấp một cây miêu tả tài liệu XML trong bộ nhớ, cho phép điều hướng và soạn thảo trước khi Save thành file. Hỗ trợ Xpath. |
| XmlDataDocument | Mở rộng của XmlDocument. Đây là một tài liệu có thể được tải từ dữ liệu XML hoặc từ dữ liệu trong một ADO.NET DataSet. Cho phép hòa trộn XML và dữ liệu quan hệ trong cùng một view |

Trong số các class này, XmlDocument được xem là linh động và tiện dụng nhất do có áp dụng mô hình DOM để dễ dàng xử lý và không thao tác trực tiếp trên file XML cũng như hỗ trợ Xpath để làm việc với các node dễ dàng hơn.

#### Đọc XML file:

Để đọc một XML file trong lớp này, chúng sẽ sử dụng phương thức Load(TenFile) để sao chép cấu trúc của XML file vào một đối tượng XMLDocument. Từ đây mọi thao tác, thay đổi của chúng ta sẽ chỉ xảy ra trên đối tượng này mà không ảnh hưởng đến file XML gốc

XmlDocument docXML = new XmlDocument();  
docXML.Load("D:\book.xml");

Lúc này, biến docXML là một bản sao hoàn hảo từ XML file. Việc còn lại là chúng ta sẽ tham chiếu đến các node và lấy ra các dữ liệu mà chúng ta muốn.

Nếu bạn chỉ cần đọc và show tất cả các node thì s khuyên bạn nên dùng các XmlTextReader, XmlReader Class thay vì làm theo bài viết này. Vì nó không tận dụng được ưu điểm của XMLDocument mà ngược lại có thể gây phiền phức cho bạn. nerd

Chúng ta nên biết một chút về XPATH trước khi thao tác trên đối tượng XMLDocument. XPATH giúp chúng ta có thể "đi bất cứ đâu" trong một tài liệu XML để select ra các node, thuộc tính cần thiết. Xpath đơn giản chỉ là một chuỗi biểu thức để căn cứ vào đó các node, thuộc tính được chọn ra.

Để chọn ra các Node (bao gồm các node con), cơ bản chúng ta sẽ có 2 phương thức sau:

SelectNodes(): Chọn ra một tập các Node thõa điều kiện Xpath.

SelectSingleNode(): Chọn ra Node đầu tiên thõa điều kiện.

string xpath = "//book";  
// Lấy ra node book đầu tiên.  
XmlNode book = Xml.SelectSingleNode(xpath);  
// Lấy ra tất cả node book hiện có.  
XmlNodeList bookList = Xml.SelectNodes(xpath);

Để lấy giá trị từ các Node đơn trả về, chúng ta sử dụng các thuộc tính sau

InnerText: Lấy giá trị của node hiện tại.

InnerXml: Lấy node con của node hiện tại dạng XML.

OuterXml: Lấy node hiện tại và các node con dạng XML.

string value = book.innerText;  
// Lấy thuộc tính đầu tiên của node book  
string value = book.Attributes[0].InnerText;

Ngoài ra chúng ta còn thể tham chiếu đến rất nhiều thứ thông qua node hiện tại. Xem phần thuộc tính của XmlNode ở cuối bài nhé. Đến đây xem như việc đọc file Xml của ta đã hoàn tất.

#### Tạo và ghi XML file:

1. Tạo và thêm Node mới trong XML:

Để tạo nhanh nội dung cho một XML file mới, chúng ta có thể dùng phương thức LoadXml() và dùng đối tượng XmlTextWriter để lưu chúng thành XML file.

string contentXML = "" +  
 "wrench" +  
 "salipro" +  
 "";  
 XmlDocument docXML = new XmlDocument();  
 docXML.LoadXml(contentXML);  
  
 // Save thành file.  
 XmlTextWriter writer = new XmlTextWriter("data.xml",null);  
 writer.Formatting = Formatting.Indented;  
 docXML.Save(writer);

Thông thường tạo một Node mới để chèn chúng vào một tài liệu Xml có sẵn, chúng ta sẽ sử dụng những phương thức bên dưới đây.

CreateNode(): Tạo node mới.

CreateElement(): Tạo element mới cho node.

CreateAttribute(): Tạo thuộc tính mới cho node hoặc element.

node.PrependChild(element): Thêm element vào trước những element đã có.

node.AppendChild(element): Thêm element vào sau những element đã có.

node.Attributes.Append(attribute): Thêm thuộc tính vào node.

element.Attributes.Append(attribute): Thêm thuộc tính vào element.

root.AppendChild(node): Thêm node vừa tạo vào node khác, cụ thể là root.

// Lấy node gốc đã có  
 XmlNode root = Xml.SelectSingleNode("/bookstore");  
// Tạo một node mới.  
 XmlNode node = Xml.CreateNode(XmlNodeType.Element, "book", null);  
 XmlAttribute attCat = Xml.CreateAttribute("category");  
 node.Attributes.Append(attCat);  
   
 XmlElement = Xml.CreateElement("title");  
 element1.InnerText = "Tên sách";  
 XmlElement = Xml.CreateElement("price");  
 element2.InnerText = "Giá";  
 node.AppendChild(element1);  
 node.AppendChild(element2);  
// Thêm node vào bookstore  
 root.AppendChild(node);

1. Cập nhật một Node:

Chúng ta sẽ dùng phương thức ReplaceChild() để cập nhật một node. Xem ví dụ bên dưới

XmlNode root = Xml.SelectSingleNode("/bookstore");  
 XmlNodeList titleList = Xml.SelectNodes("//book/title");  
  
//Giả sử chúng ta đã tạo ra một node mới từ TaoNodeMoi()  
 XmlNode newNode = TaoNodeMoi(title, price);  
 XmlNode oldNode = null;  
 foreach (XmlNode nameBook in titleList)  
 {  
 if (nameBook.InnerText == "AAA")  
 {  
 //Xác định Node cần cập nhật.  
 oldNode = nameBook.ParentNode;  
 //Cập nhật Node  
 root.ReplaceChild(newNode, oldNode);  
 break;  
 }  
 }

Ví dụ trên mục tiêu là cập nhật Node "book" có title = "AAA". Đầu tiên ta cần tham chiếu đến node Cha của Node book (cụ thể root), vì chỉ có node Cha mới đủ quyền thao tác, điều này đúng cho cả phần Xóa node.

1. Xóa một Node:

Hoàn toàn tương tự nhứ Cập nhật node, ở đây chúng ta sử dung phương thức RemoveChild().

namespace System.Xml đã cung cấp một hướng tiếp cận mới khi xây dựng project với CSDL thuộc kiểu XML. Hướng tiếp cận này càng được củng cố khi ở phiên bản .NET 3.5 Microsoft đã nâng cấp bổ trợ thêm truy vấn LINQ to namespace này với System.Xml.Linq.

### LINQ to XML:

LINQ to XML là một cách tiếp cận được cập nhật, được thiết kế lại để lập trình với XML. Nó cung cấp các khả năng sửa đổi tài liệu trong bộ nhớ của Mô hình đối tượng tài liệu (DOM) và hỗ trợ các biểu thức truy vấn LINQ. Mặc dù các biểu thức truy vấn này khác về mặt cú pháp so với XPath, chúng cung cấp các chức năng tương tự.

#### Tổng quan:

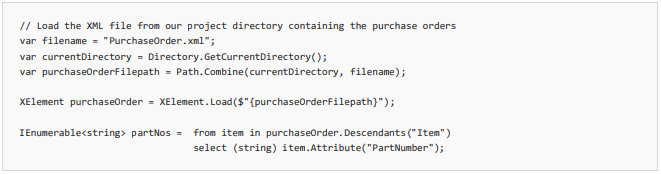
LINQ to XML là giao diện lập trình XML trong bộ nhớ hỗ trợ LINQ to phép chúng ta làm việc với XML trực tiếp bằng các ngôn ngữ lập trình .NET Framework.

LINQ to XML giống như Mô hình Đối tượng Tài liệu (DOM) ở chỗ nó đưa XML file vào bộ nhớ. Chúng ta có thể truy vấn và sửa đổi file, và sau khi sửa đổi nó xong, ta có thể lưu nó vào một tệp hoặc sắp xếp lại và gửi nó qua Internet. Tuy nhiên, LINQ to XML khác với DOM: Nó cung cấp một mô hình đối tượng mới có dung lượng nhẹ hơn và dễ làm việc hơn, tận dụng các tính năng của ngôn ngữ C#.

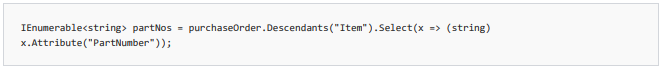
Ưu điểm quan trọng nhất của LINQ to XML là tích hợp với Truy vấn LINQ. Việc tích hợp này cho phép ta viết các truy vấn trên tài liệu XML trong bộ nhớ để truy xuất tập hợp các phần tử và thuộc tính. Khả năng truy vấn của LINQ to XML có thể so sánh tương tự về chức năng (mặc dù không cùng cú pháp) với XPath và XQuery. Việc tích hợp LINQ trong C# cung cấp khả năng làm việc mạnh hơn, thời gian biên dịch và hỗ trợ debug được cải thiện.

Một ưu điểm khác của LINQ to XML là khả năng sử dụng các kết quả truy vấn làm tham số cho các hàm tạo đối tượng **XElements** và **XAttribution** xây dựng một cách tiếp cận mạnh mẽ để tạo cây XML. Cách tiếp cận này được gọi là xây dựng chức năng, cho phép các nhà phát triển dễ dàng chuyển đổi các cây XML từ hình dạng này sang hình dạng khác.

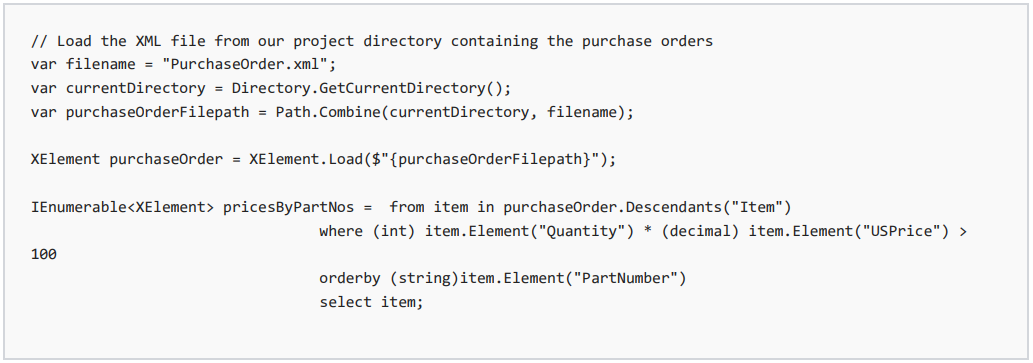
Ví dụ: ta có thể có một Order XML điển hình như được mô tả trong Tệp XML mẫu: PurchaseOrder.xml. Bằng cách sử dụng LINQ to XML, ta có thể chạy truy vấn sau để có được giá trị thuộc tính PartNumber cho mọi thành phần item trong Order:



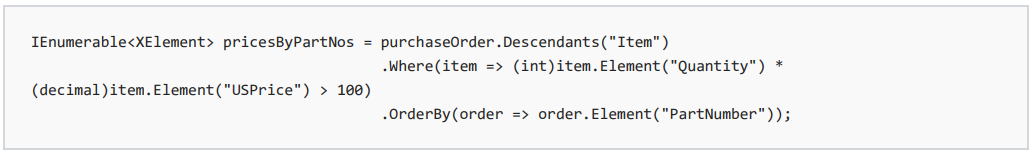
Viết lại dưới dạng cú pháp phương thức:

****

Một ví dụ khác, nếu ta muốn một danh sách được sắp xếp theo PartNumber của các item có giá trị lớn hơn $100. Để có được thông tin này, ta có thể chạy truy vấn sau:

****

Một lần nữa, điều này có thể được viết lại dưới dạng cú pháp phương thức:

****

Ngoài các khả năng LINQ này, LINQ to XML còn cung cấp giao diện lập trình XML cải tiến. Sử dụng LINQ to XML, chúng ta có thể:

* Lấy dữ liệu XML từ **Files** và **streams.**
* Chuyển dữ liệu XML thành **Files** và **streams.**
* Tạo XML từ đầu bằng cách sử dụng chức năng xây dựng.
* Truy vấn XML bằng các trục giống như XPath.
* Thao tác với cây XML trong bộ nhớ bằng cách sử dụng các phương thức như **Add**, **Remove**, **ReplaceWith** và **SetValue**.
* Xác thực cây XML bằng XSD.
* Sử dụng kết hợp các tính năng này để chuyển đổi cây XML từ hình dạng này sang hình dạng khác.

**Tạo cây XML**

Một trong những lợi thế quan trọng nhất của việc lập trình với LINQ to XML là dễ dàng tạo các cây XML. Ví dụ: để tạo một cây XML nhỏ, ta có thể viết code như sau:



#### Lợi ích của LINQ to XML:

LINQ to XML nhắm vào nhiều nhà phát triển. Đối với một nhà phát triển trung bình chỉ muốn hoàn thành công việc, LINQ to XML giúp làm việc với XML dễ dàng hơn bằng cách cung cấp trải nghiệm truy vấn tương tự như SQL. Chỉ cần một chút nghiên cứu, các lập trình viên có thể học cách viết các truy vấn ngắn gọn và mạnh mẽ với ngôn ngữ lập trình(.NET) mà họ đã chọn.

Các nhà phát triển chuyên nghiệp có thể sử dụng LINQ to XML để tăng đáng kể năng suất dự án của họ. Với LINQ to XML, họ có thể viết ít code hơn, dễ hiểu hơn, gọn hơn và mạnh mẽ hơn. Họ có thể sử dụng các biểu thức truy vấn từ nhiều miền dữ liệu cùng một lúc.

#### LINQ to SQL so với DOM:

**Những cách mới để xây dựng cây XML**

Trong DOM W3C, bạn xây dựng cây XML từ dưới lên; nghĩa là bạn tạo một tài liệu, bạn tạo các phần tử và sau đó bạn thêm các phần tử vào tài liệu.

Ví dụ, sau đây sẽ là một cách điển hình để tạo cây XML bằng cách sử dụng DOM với class **XmlDocument** của Microsoft:



Kiểu xây dựng này không cung cấp trực quan thông tin về cấu trúc của cây XML. LINQ to XML có hỗ trợ cách tiếp cận này để xây dựng cây XML, nhưng cũng hỗ trợ một cách tiếp cận khác tiến bộ hơn, Khởi tạo có cấu trúc. Khởi tạo có cấu trúc sử dụng các hàm tạo **XElement** và **XAttribute** để xây dựng cây XML.

Chúng ta sẽ xây dựng lại cây XML bằng cách sử dụng chức năng khởi tạo có cấu trúc của LINQ to XML:



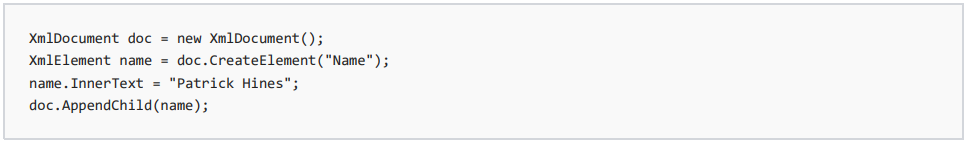
Lưu ý rằng việc thụt lề code trong hàm xây dựng cây XML trong nhằm mục đích cho ta thấy cấu trúc XML của biến.

**Làm việc trực tiếp với các phần tử (Elements) của XML**

Khi ta lập trình với XML, trọng tâm chính của sẽ thường là các phần tử (Elements) XML và có thể là các thuộc tính (Attributes). Trong LINQ to XML, ta có thể làm việc trực tiếp với các thành phần và thuộc tính này. Ví dụ: ta có thể làm như sau:

* Tạo các phần tử XML mà không sử dụng một đối tượng Document nào cả. Điều này đơn giản hóa việc lập trình khi ta phải làm việc với các phần của cây XML.
* Nhận các đối tượng T:System.Xml.Linq.XElement trực tiếp từ file XML.
* Sắp xếp các đối tượng T:System.Xml.Linq.XElement vào một file hoặc luồng (stream).

So sánh điều này với W3C DOM, trong đó XML Document được sử dụng làm nơi chứa cấu trúc logic của cây XML.Trong DOM, các XML Node, bao gồm các Element và Attribute, phải được tạo trong ngữ cảnh của file XML. Đây là một đoạn code để tạo thành phần tên trong DOM:

****

Nếu muốn sử dụng một Element trên nhiều tài liệu, ta phải lại nhập các Node trên các tài liệu đó. LINQ to XML hỗ trợ tránh sự phức tạp này.

Khi sử dụng LINQ to XML, ta chỉ sử dụng lớp **XDocument** nếu muốn thêm một nhận xét hoặc hướng dẫn xử lý ở cấp Root của tài liệu.

**Xử lý đơn giản hóa name và namespace**

Xử lý name, namespace và tiền tố (prefix) của namespace nói chung là một phần rất phức tạp của lập trình XML. LINQ to XML đơn giản hóa các name và namespace bằng cách loại bỏ yêu cầu xử lý các tiền tố namespace. Chúng ta vẫn có thể kiểm soát tiền tố namespace nếu muốn. Nhưng nếu quyết định không kiểm soát rõ ràng các tiền tố namespace, LINQ to XML sẽ gán các tiền tố namespace trong quá trình tuần tự hóa nếu chúng được yêu cầu hoặc sẽ tuần tự hóa bằng cách sử dụng các namespace mặc định nếu không có bất kì yêu cầu nào đến chúng. Nếu namespace mặc định được sử dụng, sẽ không có tiền tố namespace trong tài liệu kết quả.

Một vấn đề khác với DOM là nó không cho phép ta thay đổi tên của một Node. Thay vào đó, chúng ta phải tạo một Node mới và sao chép tất cả các Node con vào nó, qua đó làm mất tính đơn nhất của Node gốc. LINQ to XML tránh được vấn đề này bằng cách cho phép người dùng đặt thuộc tính **XName** trên một Node.

**Hỗ trợ phương thức tĩnh (Static Methods) để tải XML**

LINQ to XML cho phép ta tải XML bằng cách sử dụng các phương thức tĩnh, thay vì các phương thức cá thể (Instance Methods). Điều này đơn giản hóa việc tải và phân tích cú pháp.

**Loại bỏ sự hỗ trợ các cấu trúc DTD (Document type Definition/Declaration - DOCTYPE)**

LINQ to XML còn đơn giản hóa việc lập trình XML bằng cách loại bỏ hỗ trợ cho các thực thể và tham chiếu thực thể. Việc quản lý các thực thể rất phức tạp và hiếm khi được sử dụng. Loại bỏ hỗ trợ của chúng làm tăng hiệu suất và đơn giản hóa không gian lập trình. Khi cây LINQ to XML được xây dựng, tất cả các thực thể DTD sẽ được mở rộng.

**Hỗ trợ cho các Fragment**

LINQ to XML không cung cấp hỗ trợ cho lớp XmlDocumentFragment. Tuy nhiên, trong nhiều trường hợp, cấu trúc XmlDocumentFragment có thể được xử lý bằng kết quả của một truy vấn kiểu **IEnumerable<T>** của **XNode** hoặc **IEnumerable<T>** của **XElement**.

**Hỗ trợ cho XPathNavigator**

LINQ to XML cung cấp hỗ trợ cho **XPathNavigator** thông qua các phương thức mở rộng trong namespace **System.Xml.XPath**.

**Hỗ trợ cho dấu cách và thụt lề**

LINQ to XML xử lý khoảng trắng đơn giản hơn DOM.

Một kịch bản phổ biến là đọc XML kiểu răng cưa, tạo cây XML trong bộ nhớ mà không có bất kỳ khoảng trắng nào (nghĩa là không bảo toàn khoảng trắng), thực hiện một số thao tác trên XML và sau đó lưu XML theo cấu trúc răng cưa. Khi ta sắp xếp XML theo định dạng, chỉ có khoảng trắng quan trọng trong cây XML được giữ nguyên. Đây là phương thức mặc định trong LINQ to XML.

Một kịch bản phổ biến khác là đọc và sửa đổi XML đã được thụt lề có cấu trúc. Chúng ta có thể không muốn thay đổi cấu trúc này theo bất kỳ cách nào. Trong LINQ to XML, ta có thể thực hiện điều này bằng cách duy trì khoảng trắng khi tải hoặc phân tích cú pháp XML và vô hiệu hóa định dạng khi tuần tự hóa XML.

LINQ to XML lưu trữ khoảng trắng dưới dạng nút **XText**, thay vì có loại nút **Whitespace** chuyên dụng, như DOM thực hiện.

**Hỗ trợ chú thích**

Các phần tử LINQ to XML hỗ trợ một bộ chú thích mở rộng. Điều này giúp theo dõi thông tin đa dạng về một phần tử, chẳng hạn như thông tin lược đồ, thông tin về việc phần tử đó có bị ràng buộc với UI hay bất kỳ loại thông tin cụ thể nào khác của ứng dụng hay không.

**Hỗ trợ thông tin lược đồ**

LINQ to XML cung cấp hỗ trợ xác thực XSD (lược đồ XML) thông qua các phương thức mở rộng trong namespace **System.Xml.Schema**. Chúng ta có thể xác định rằng một cây XML tuân thủ XSD hay không. Và thể điền vào cây XML bằng post-schema-validation infoset(PSVI).

#### LINQ to XML so với các công nghệ XML khác:

Chúng ta sẽ so sánh LINQ với XML với các công nghệ XML sau: **XmlReader**, XSLT, MSXML và XmlLite.

Để so sánh LINQ với XML với Mô hình đối tượng tài liệu (DOM), hãy xem **LINQ với XML so với DOM (C #).**

**LINQ to XML so với XmlReader**

**XmlReader** là một trình phân tích cú pháp nhanh, không chuyển tiếp, không bộ đệm.

LINQ to XML được triển khai trên **XmlReader** và chúng được tích hợp chặt chẽ. Tuy nhiên, ta cũng có thể sử dụng **XmlReader** riêng.

Ví dụ: giả sử ta đang xây dựng một dịch vụ Web sẽ phân tích hàng trăm tài liệu XML mỗi giây và các tài liệu có cùng cấu trúc, nghĩa là ta chỉ phải viết một cấu trúc code để phân tích cú pháp XML. Trong trường hợp này, ta nên sử dụng **XmlReader**.

Ngược lại, nếu ta đang xây dựng một hệ thống phân tích nhiều tài liệu XML nhỏ hơn và mỗi tài liệu khác nhau, chúng ta cần tận dụng các cải tiến năng suất mà LINQ sang XML cung cấp.

**LINQ to XML so với XSLT**

Cả LINQ sang XML và XSLT đều cung cấp các khả năng chuyển đổi tài liệu XML mở rộng. XSLT là một cách tiếp cận khai báo dựa trên quy tắc. Các lập trình viên XSLT nâng cao viết XSLT theo kiểu lập trình chức năng nhấn mạnh cách tiếp cận không trạng thái. Các biến đổi có thể được viết bằng các hàm thuần túy được thực hiện mà không có tác dụng phụ. Cách tiếp cận dựa trên quy tắc hoặc chức năng này không quen thuộc với nhiều nhà phát triển và có thể khó học và tốn thời gian để tìm hiểu.

XSLT có thể là một hệ thống rất năng suất mang lại các ứng dụng hiệu suất cao. Ví dụ: một số công ty web lớn sử dụng XSLT như một cách để tạo HTML từ XML được lấy từ nhiều kho dữ liệu khác nhau. Công cụ XSLT được quản lý biên dịch mã XSLT thành CLR và thậm chí còn hoạt động tốt hơn trong một số trường hợp so với công cụ XSLT gốc. Tuy nhiên, XSLT không tận dụng kiến thức C # và Visual Basic mà nhiều nhà phát triển có. Nó yêu cầu các nhà phát triển viết mã bằng một ngôn ngữ lập trình phức tạp và khác biệt. Sử dụng hai hệ thống phát triển không tích hợp như C # (hoặc Visual Basic) và XSLT dẫn đến các hệ thống phần mềm khó phát triển và bảo trì hơn.

Sau khi bạn đã thành thạo các biểu thức truy vấn LINQ to XML, các phép biến đổi LINQ sang XML là một công nghệ mạnh mẽ, dễ sử dụng. Về cơ bản, bạn hình thành tài liệu XML của mình bằng cách sử dụng chức năng xây dựng, lấy dữ liệu từ nhiều nguồn khác nhau, xây dựng các đối tượng **Xlement** một cách linh hoạt và lắp ráp toàn bộ thành một cây XML mới. Việc chuyển đổi có thể tạo ra một tài liệu hoàn toàn mới. Xây dựng các phép biến đổi trong LINQ thành XML tương đối dễ dàng và trực quan và mã kết quả có thể đọc được. Điều này làm giảm chi phí phát triển và bảo trì.

LINQ to XML không nhằm thay thế XSLT. XSLT vẫn là công cụ được lựa chọn cho các phép biến đổi XML phức tạp và tập trung vào tài liệu, đặc biệt nếu cấu trúc của tài liệu không được xác định rõ.

XSLT có lợi thế là tiêu chuẩn của World Wide Web Consortium (W3C). Nếu bạn có một yêu cầu là bạn chỉ sử dụng các công nghệ là tiêu chuẩn, XSLT có thể phù hợp hơn.

XSLT là XML và do đó có thể được lập trình thao tác.

**LINQ to XML so với MSXML**

MSXML là công nghệ dựa trên COM để xử lý XML được bao gồm trong Microsoft Windows. MSXML cung cấp một triển khai nguyên gốc của DOM với sự hỗ trợ cho XPath và XSLT. Nó cũng chứa bộ phân tích cú pháp dựa trên sự kiện, bộ nhớ đệm SAX2.

MSXML hoạt động tốt, được bảo mật theo mặc định trong hầu hết các kịch bản và có thể được truy cập trong Internet Explorer để thực hiện xử lý XML phía máy khách trong các ứng dụng kiểu AJAX. MSXML có thể được sử dụng từ bất kỳ ngôn ngữ lập trình nào hỗ trợ COM, bao gồm C ++, JavaScript và Visual Basic 6.0.

MSXML không được khuyến nghị sử dụng trong mã được quản lý dựa trên thời gian chạy ngôn ngữ chung (CLR).

**LINQ to XML so với XmlLite**

XmlLite là một bộ đệm không lưu trữ, chỉ chuyển tiếp, gọi trình phân tích cú pháp. Các lập trình viên chủ yếu sử dụng XmlLite với C++. Lập trình viên không nên sử dụng XmlLite với mã được quản lý.

Ưu điểm chính của XmlLite là nó là một trình phân tích cú pháp XML nhẹ, nhanh, an toàn trong hầu hết các tình huống. Diện tích bề mặt đe dọa của nó là rất nhỏ. Nếu bạn phải phân tích các tài liệu không đáng tin cậy và bạn muốn bảo vệ chống lại các cuộc tấn công như từ chối dịch vụ hoặc tiếp xúc với dữ liệu, XmlLite có thể là một lựa chọn tốt.

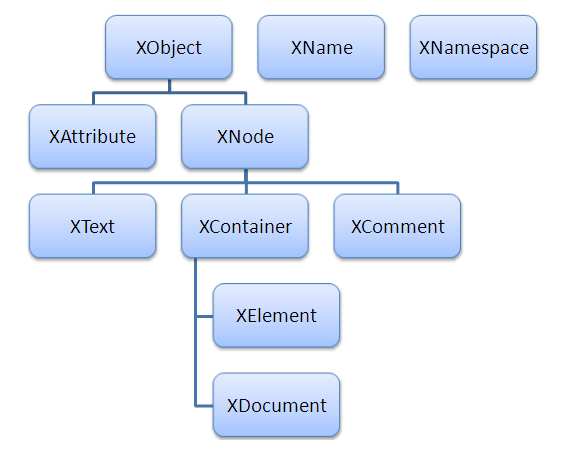
XmlLite không được tích hợp với Truy vấn tích hợp ngôn ngữ (LINQ). Nó không mang lại sự cải thiện năng suất lập trình viên là động lực thúc đẩy đằng sau LINQ.

### Namespace System.Xml.Linq:

Sức mạnh của LINQ to XML chứa trong namespace System.Xml.Linq với 19 lớp cơ bản sau:

Xattribute, XCData, Xcomment, XContainer, XDeclaration, XDocument, XDocumentType, XElement, XName, XNamespace, XNode, XNodeDocumentOrderComparer, XNodeEqualityComparer, XObject, XObjectChange, XobjectChangeEventArgs, XObjectEventHandler, XProcessingInstruction, Xtext.

#### Cây phân cấp lớp trong LINQ to XML API:



Công dụng của mỗi lớp trên có thể được nhận biết ngay từ tên gọi của chúng, tuy nhiên để rõ ràng hơn, chúng ta có thể xem bảng sau đây:

|  |  |
| --- | --- |
| **Class** | **Mô tả** |
| XName | Đại diện cho tên của một XML element hoặc attribute |
| XNamespace | Đại diện cho một XML namespace. Lớp này không thể kế thừa |
| XObject | Đại diện cho một Node hoặc một attribute trên cây XML |
| XAttribute | Đại diện cho một XML attribute |
| XNode | Đại diện cho cấu trúc trừu tượng của một Node (element, comment, document type, processing instruction, or text node) trên cây XML |
| XText | Đại diện cho một Node text |
| XComment | Đại diện cho một XML comment |
| XContainer | Đại diện cho một Node mà có thể chứa Node khác |
| XElement | Đại diện cho một XML element |
| XDocument | Đại diện cho một đoạn văn bản XML |

1. XNamespace:

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

XNamespace ns=”https://yinyangit.wordpress.com/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=”https://yinyangit.wordpress.com/namespace&#8221;

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

1. 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 chúng ta 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=”{https://yinyangit.wordpress.com}Test”;

1. XObject:

Được thừa kế từ System.Object và IXmlLineInfo (System.Xml), XObject là một lớp abstract nằm ở đỉnh của cây phân cấp như minh họa ở trên. Lớp này cung cấp các property 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 (đối tên), Value (đổi giá trị).

using System;

using System.Xml.Linq;

class LinqToXmlEx

{

static void Main(string[] args)

{

XElement root = new XElement("Root");

XElement lchild = new XElement("LChild","Foo");

XElement rchild = new XElement("RChild","Bar");

root.Changing += (sender, e) =>

{

Console.WriteLine("Changing type: " + e.ObjectChange);

};

root.Add(lchild);

root.Add(rchild);

lchild.Name = "LeftChild";

rchild.Add("rel");

lchild.Remove();

Console.WriteLine("----------");

Console.WriteLine(root);

Console.Read();

}

}

Output:

Changing type: Add

Changing type: Add

Changing type: Name

Changing type: Value

Changing type: Remove

———-

<Root>

<RChild>Barrel</RChild>

</Root>

Thêm vào đó, XObject cho phép ta 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. Các phương thức làm việc với annotation trong XObject:

|  |  |
| --- | --- |
| **Name** | **Description** |
| [**AddAnnotation**](http://msdn.microsoft.com/en-us/library/system.xml.linq.xobject.addannotation.aspx) | Thêm một đối tượng vào danh sách chú thích (Annotation) của XObject. |
| [**Annotation(Type)**](http://msdn.microsoft.com/en-us/library/bb350955.aspx) | Lấy đối tượng chú thích đầu tiên của một kiểu xác định trong XObject. |
| [**Annotation<T>()**](http://msdn.microsoft.com/en-us/library/bb292552.aspx) | Tương tự như Annotation(Type) |
| [**Annotations(Type)**](http://msdn.microsoft.com/en-us/library/bb156218.aspx) | Lấy danh sách đối tượng chú thích của một kiểu xác định trong XObject. |
| [**Annotations<T>()**](http://msdn.microsoft.com/en-us/library/bb302164.aspx) | Tương tự như Annotations(Type) |
| [**RemoveAnnotations(Type)**](http://msdn.microsoft.com/en-us/library/bb351439.aspx) | Xóa các chú thích thuộc một kiểu xác định trong XObject. |
| [**RemoveAnnotations<T>()**](http://msdn.microsoft.com/en-us/library/bb301198.aspx) | Tương tự như RemoveAnnotations(Type) |

1. XAttribute:

Một phần tử XML có thể chứa nhiều attribute, mỗi attribute này được đại diện bởi một đối tượng XAttribute. Chúng ta 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.

1. XNode:

Đại diện cho các phần tử (element, comment, document type, processing instruction, text node). XNode cho phép ta 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 file XML. Các phần tử được thêm vào sẽ có cùng level với nhau và có cùng phần tử cha.

|  |  |
| --- | --- |
| **Name** | **Description** |
| [**AddAfterSelf(Object)**](http://msdn.microsoft.com/en-us/library/bb154665.aspx) | Thêm một đối tượng vào phía sau Node này |
| [**AddAfterSelf(Object[])**](http://msdn.microsoft.com/en-us/library/bb338557.aspx) | Thêm một nhóm đối tượng vào sau Node này |
| [**AddBeforeSelf(Object)**](http://msdn.microsoft.com/en-us/library/bb343247.aspx) | Thêm một đối tượng vào phía trước Node này |
| [**AddBeforeSelf(Object[])**](http://msdn.microsoft.com/en-us/library/bb310533.aspx) | Thêm một nhóm đối tượng vào trước Node này |
| [**Remove**](http://msdn.microsoft.com/en-us/library/system.xml.linq.xnode.remove.aspx) | Xóa Node này |
| [**ReplaceWith(Object)**](http://msdn.microsoft.com/en-us/library/bb302711.aspx) | Thay thế Node này bằng một đối tượng khác |
| [**ReplaceWith(Object[])**](http://msdn.microsoft.com/en-us/library/bb336624.aspx) | Thay thế Node này bằng một nhóm đối tượng khác |

Ngoài ra còn có các phương thức để duyệt các phần tử trong tài liệu XML như Ancestors() dùng để lấy về một collection các phần tử có level cao hơn (tổ tiên):

|  |  |
| --- | --- |
| **Name** | **Description** |
| [**Ancestors()**](http://msdn.microsoft.com/en-us/library/bb339717.aspx) | Trả về danh sách(Collection) các Element cấp trên của Node này |
| [**Ancestors(XName)**](http://msdn.microsoft.com/en-us/library/bb348268.aspx) | Trả về danh sách các Element cấp trên đã được lọc của Node này. Các Element này sẽ phải thỏa mãn kiểu XName |
| [**ElementsAfterSelf()**](http://msdn.microsoft.com/en-us/library/bb345729.aspx) | Trả về danh sách Element cùng cấp với Node này, theo thứ tự tài liệu. |
| [**ElementsAfterSelf(XName)**](http://msdn.microsoft.com/en-us/library/bb354563.aspx) | Trả về danh sách Element cùng cấp với Node này, theo thứ tự tài liệu. Các Element này sẽ phải thỏa mãn kiểu XName |
| [**IsAfter**](http://msdn.microsoft.com/en-us/library/system.xml.linq.xnode.isafter.aspx) | Xác định Node hiện tại có đứng sau một Node nhập vào hay không |
| [**IsBefore**](http://msdn.microsoft.com/en-us/library/system.xml.linq.xnode.isbefore.aspx) | Xác định Node hiện tại có đứng trước một Node nhập vào hay không |
| [**NodesAfterSelf**](http://msdn.microsoft.com/en-us/library/system.xml.linq.xnode.nodesafterself.aspx) | Trả về danh sách các Node cùng cấp liền trước của Node hiện tại |
| [**NodesBeforeSelf**](http://msdn.microsoft.com/en-us/library/system.xml.linq.xnode.nodesbeforeself.aspx) | Trả về danh sách các Node cùng cấp liền sau của Node hiện tại |

Ngoài ra, Lớp này còn chứa các phương thức để đọc, ghi nội dung XML.

1. XText và XComment:

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

1. 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.

Các phương thức mà ta cần quan tâm của lớp này:

|  |  |
| --- | --- |
| **Name** | **Description** |
| [**Add(Object)**](http://msdn.microsoft.com/en-us/library/bb354061.aspx) | Thêm một đối tượng con cho XContainer. |
| [**Add(Object[])**](http://msdn.microsoft.com/en-us/library/bb345077.aspx) | Thêm một danh sách đối tượng con cho XContainer. |
| [**AddFirst(Object)**](http://msdn.microsoft.com/en-us/library/bb344365.aspx) | Thêm một đối tượng con vào đầu XContainer này |
| [**AddFirst(Object[])**](http://msdn.microsoft.com/en-us/library/bb340917.aspx) | Thêm một danh sách đối tượng con vào đầu XContainer này |
| [**DescendantNodes**](http://msdn.microsoft.com/en-us/library/system.xml.linq.xcontainer.descendantnodes.aspx) | Trả về danh sách đối tượng con (Document, Element) của XContainer này |
| [**Descendants()**](http://msdn.microsoft.com/en-us/library/bb360635.aspx) | Trả về danh sách các đối tượng con của một Element con xác định |
| [**Descendants(XName)**](http://msdn.microsoft.com/en-us/library/bb353813.aspx) | Trả về danh sách các đối tượng con của một Element con xác định. Các Element này sẽ phải thỏa mãn kiểu Xname. |
| [**Element**](http://msdn.microsoft.com/en-us/library/system.xml.linq.xcontainer.element.aspx) | Trả về Element con đầu tiên với kiểu XName xác định |
| [**Elements()**](http://msdn.microsoft.com/en-us/library/bb342765.aspx) | Trả về danh sách các Element con của XContainer này |
| [**Elements(XName)**](http://msdn.microsoft.com/en-us/library/bb348975.aspx) | Trả về danh sách các Element con của XContainer này. Các Element này sẽ phải thỏa mãn kiểu XName |
| [**Nodes**](http://msdn.microsoft.com/en-us/library/system.xml.linq.xcontainer.nodes.aspx) | Trả về danh sách các Node con của XContainer này |
| [**RemoveNodes**](http://msdn.microsoft.com/en-us/library/system.xml.linq.xcontainer.removenodes.aspx) | Xóa các Node con trong XContainer này |
| [**ReplaceNodes(Object)**](http://msdn.microsoft.com/en-us/library/bb156423.aspx) | Thay thế một Node con trong XContainer này bằng một đối tượng |
| [**ReplaceNodes(Object[])**](http://msdn.microsoft.com/en-us/library/bb350830.aspx) | Thay thế một Node con trong XContainer này bằng một danh sách đối tượng |

1. 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 file thành một phần tử.

Ví dụ ta sử dụng Parse() để tạo một phần tử XML từ một đoạn văn bản. Hai đối tượng x1 và x2 sau sẽ tạo ra một phần tử giống nhau:

XElement x1=XElement.Parse("<Blog title=\"Let me know what you think\"><Name>Yin Yang</Name></Blog>");

XElement x2=new XElement("Blog",

new XAttribute("title","Let me know what you think"),

new XElement("Name","Yin Yang"));

Tài liệu XML được tạo ra từ x1 và x2 trên sẽ có cùng nội dung như sau:

<Blog title="Let me know what you think">

<Name>Yin Yang</Name>

</Blog>

1. XDocument:

XDocument là lớp đại diện cho toàn bộ một tài liệu XML. Sau đây là một ví dụ tương đối hoàn chỉnh từ MSDN cho những gì ta mới nghiên cứu:

using System;

using System.Linq;

using System.Xml.Linq;

public class Linq2XmlExample {

public static void Main() {

XDocument srcTree = new XDocument(

new XComment("This is a comment"),

new XElement("Root",

new XElement("Child1", "data1"),

new XElement("Child2", "data2"),

new XElement("Child3", "data3"),

new XElement("Child2", "data4"),

new XElement("Info5", "info5"),

new XElement("Info6", "info6"),

new XElement("Info7", "info7"),

new XElement("Info8", "info8")

)

);

XDocument doc = new XDocument(

new XComment("This is a comment"),

new XElement("Root",

from el in srcTree.Element("Root").Elements()

where ((string)el).StartsWith("data")

select el

)

);

Console.WriteLine(doc);

doc.Save("C:\\example.xml");

Console.Read();

}

}

Output:

<!–This is a comment–>

<Root>

<Child1>data1</Child1>

<Child2>data2</Child2>

<Child3>data3</Child3>

<Child2>data4</Child2>

</Root>

#### Đọc/ghi một file XML dùng LINQ:

Đọc dữ liệu từ file XML:

// tạo một chuỗi chứa định dạng XML

string myXML = @"

<Departments>

<Department>Account</Department>

<Department>Sales</Department>

<Department>Pre-Sales</Department>

<Department>Marketing</Department>

</Departments>";

// chuyển chuỗi thành tài liệu XML

XDocument xdoc = new XDocument();

xdoc = XDocument.Parse(myXML);

// duyệt tài liệu XML

var result = xdoc.Element("Departments").Descendants();

foreach (XElement item in result)

{

Console.WriteLine("Department Name - " + item.Value);

}

Kết quả:

Department Name - Account

Department Name - Sales

Department Name - Pre-Sales

Department Name – Marketing

Thêm một node mới vào file XML

// tạo một chuỗi chứa định dạng XML

string myXML = @"

<Departments>

<Department>Account</Department>

<Department>Sales</Department>

<Department>Pre-Sales</Department>

<Department>Marketing</Department>

</Departments>";

// chuyển chuỗi thành tài liệu XML

XDocument xdoc = new XDocument();

xdoc = XDocument.Parse(myXML);

//thêm phần tử mới vào cuối tài liệu XML (sau Marketing)

xdoc.Element("Departments").Add(new XElement("Department", "Finance"));

//thêm một phần tử mới vào đầu (trước Account

xdoc.Element("Departments").AddFirst(new XElement("Department", "Support"));

// duyệt tài liệu XML

var result = xdoc.Element("Departments").Descendants();

foreach (XElement item in result)

{

Console.WriteLine("Department Name - " + item.Value);

}

Kết quả:

Department Name - Support

Department Name - Account

Department Name - Sales

Department Name - Pre-Sales

Department Name - Marketing

Department Name - Finance

Xoá một phần tử từ file XML

// tạo một chuỗi chứa định dạng XML

string myXML = @"<Departments>

<Department>Account</Department>

<Department>Sales</Department>

<Department>Pre-Sales</Department>

<Department>Marketing</Department>

</Departments>";

// chuyển chuỗi thành tài liệu XML

XDocument xdoc = new XDocument();

xdoc = XDocument.Parse(myXML);

//Xoá Sales

xdoc.Descendants().Where(s =>s.Value == "Sales").Remove();

// duyệt tài liệu XML

var result = xdoc.Element("Departments").Descendants();

foreach (XElement item in result)

{

Console.WriteLine("Department Name - " + item.Value);

}

Kết quả:

Department Name - Account

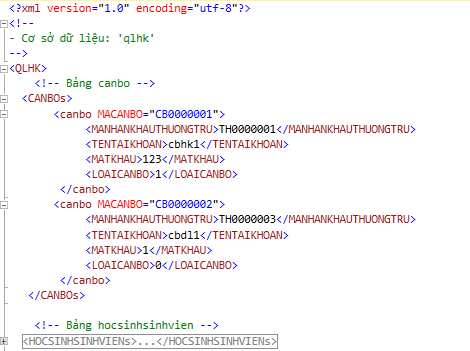
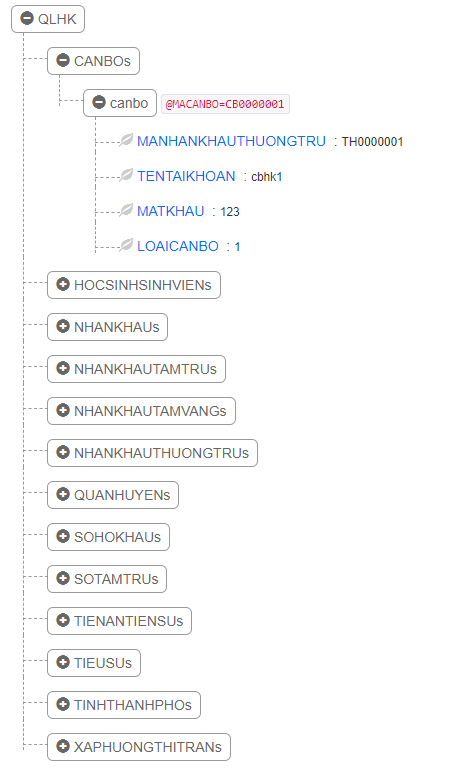
Department Name - Pre-Sales

Department Name - Marketing

## Xây dựng chương trình với LINQ to XML:

### Định nghĩa các lớp mô hình dữ liệu:

Cấu trúc của file XML được trình bày theo cấu trúc cây như phần VI.1.1, cụ thể như sau:



Trong đó, Root Node là tên database (Ví dụ: “<QLHK>”), các Node con đầu tiên của Root Node là các bảng. Mỗi bảng là một Element với tên Element là tên in hoa của bảng đó và kết thúc bằng chữ “s” (“<CANBOs>”). Phía trong của các Element bảng là các Record, tên của mỗi record là tên viết thường của tên bảng (“<canbo>”), các thông tin của mỗi Record được lưu trong các Element (“<MANHANKHAUTHUONGTRU>”, “<TENTAIKHOAN>”, ...) ngoại trừ khóa chính được lưu trong trường Attribute của Record đó.

### Truy vấn cơ sở dữ liệu:

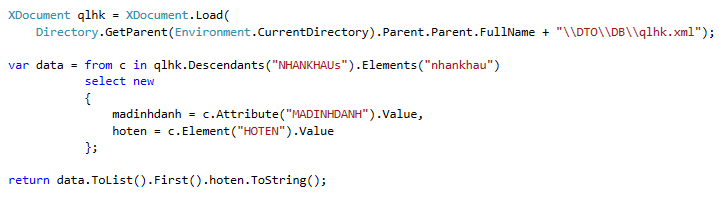
1. Lấy các nhân khẩu:

Để truy xuất thông tin CSDL từ file XML ta sẽ khởi tạo biến như sau:



Đoạn code: Directory.GetParent…\\qlhk.xml” sẽ trả về chuỗi string đường dẫn đến vị trí chứa file XML CSDL, trong thực tế, nếu chuỗi đường dẫn đến file XML là cố định, chúng ta có thể truyền trực tiếp đường dẫn này thay vì đoạn code trên. Phương thức Xdocument.Load() sẽ nhận hỗ trợ việc lấy dữ liệu của file XML và ánh xạ nó vào biến thuộc kiểu Xdocument. Mặc khác, chúng ta cũng có thể dùng phương thức XElement.Load() nhận dữ liệu từ file XML tương tự như Xdocument.Load()

Như ta đã cấu trúc ở phần VI.1.1, tên CSDL sẽ là RootNode, sau đó tên các Bảng sẽ là các Node con đầu tiên của RootNode, nên ta có truy vấn sau để lấy thông tin các nhân khẩu của bảng NHANKHAU:



Phương thức Descendants() sẽ duyệt qua các Node con của qlhk (RootNode) cho đến khi tìm thấy một danh sách Node con có tên NHANKHAUs, sau đó phương thức Elements() sẽ tiếp tục lấy ra các Element con có tên nhankhau của trong các NHANKHAUs này. Cách truy vấn trên cũng có thể rút gọn lại thành:



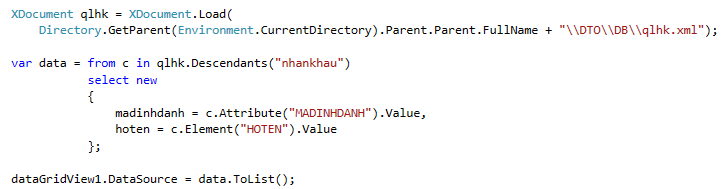
Lúc này phương thức Desendants() cũng sẽ duyệt qua các Node con của qlhk (theo trình tự kí tự trong XDocument) và cũng lấy ra danh sách các Node nhankhau. Tuy nhiên, hạn chế trong truy vấn kiểu này đó là Desendants cũng sẽ lấy ra các node nhankhau không thuộc Element (bảng) NHANKHAUs, do đó yêu cầu chúng ta cần phải đảm bảo cấu trúc cây trong file XML để có thể có các truy vấn hợp lý. Do khóa chính của các bảng được lưu trữ trong Attribute của mỗi bảng ghi (record), do đó ta cần phải lấy giá trị này thông qua phương thức XElement.Attribute().

1. Gắn các câu truy vấn vào các Control C#:

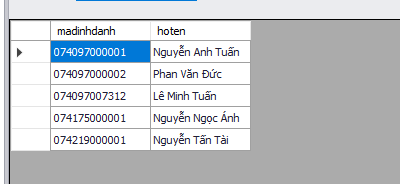
Như ta đã biết, truy vấn LINQ trả về kết quả mà nó sẽ implement interrface IEnumerable - đây cũng là interface mà các control trong C# dùng để hỗ trợ gắn nối các đối tượng. Do đó, ta có thể tùy ý truyền các giá trị truy vấn này vào các Control C# tùy ý như ví dụ sau, ta khai báo một control trong một Form C# như sau:



Chúng ta cũng có thể gắn nối kết quả của câu LINQ to XML đã viết trước đây trực tiếp vào DataGridView giống như sau:

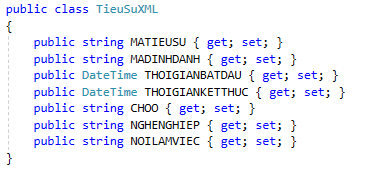


Tuy nhiên khác với LINQ to SQL, giá trị trả về của câu truy vấn LINQ ở đây sẽ là một IEnumerable thuần túy và thông thường DataGridView trong C# sẽ không nhận được dữ liệu đầu vào kiểu này, do đó chúng ta cần dùng phương thức ToList() để chuyển giá trị trả về thuộc kiểu List, màn hình kết quả sẽ trông như sau:

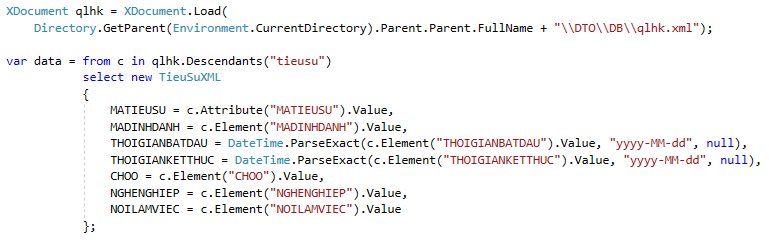


1. DataSharping:

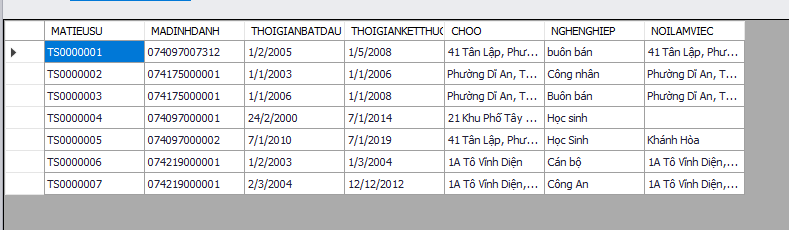
LINQ to XML không tự động tạo các bảng ánh xạ (do cấu trúc file XML đầu vào là không cố định), do đó để nhận các dữ liệu trả về của câu truy vấn LINQ to XML, chúng ta cần sử dụng Anounymous Type như ví dụ trên, hoặc xây các Lớp ánh xạ các thành phần của các bảng một cách thủ công. Xét Ví dụ sau, ta tạo một Lớp TieuSuXML để nhận các thông tin tiểu sử từ CSDL:



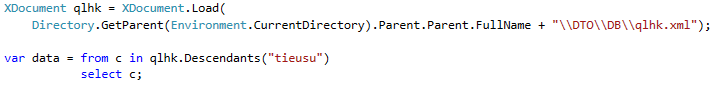
Sau đó thực hiện truy vấn theo lớp vừa tạo:



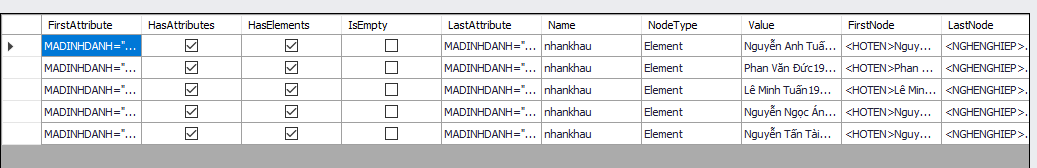
Kết quả trả về khi đó sẽ như sau:



Trong trường hợp chúng ta trả về trực tiếp biến truy vấn:



Thì kết quả trả về sẽ như sau:

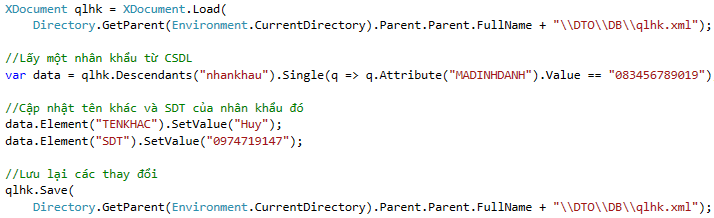


Kiểu dữ liệu trả về của truy vấn này là một IEnumerable<XElement> do đó có cấu trúc dữ liệu tuân thủ theo cấu trúc cây XML với các phương thức đặc trưng của XElement như IsEmpty(), FirstAttribute(), LastAttribute(), Value, FirstNode(),…

### Cập nhật cơ sở dữ liệu:

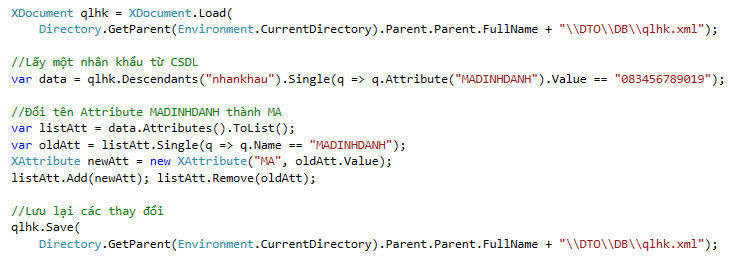
1. Cập nhật dữ liệu sử dụng SetValue() và Save():

LINQ to XML cũng hỗ trợ lưu vết các thao tác thay đổi hay cập nhật mà chúng ta thực hiện trên các đối tượng đó (change tracking) tương tự như LINQ to XML. Chúng ta có thể thoải mái thay đổi các giá trị (Value) của các Element, Attribute, thêm các comment, text, … Và sau đó gọi phương thức Save() để lưu lại. Xét ví dụ sau, ta lấy ra một nhân khẩu, cập nhật thông tin tên khác và SDT của nhân khẩu đó, sau đó lưu kết quả cập nhật vào file qlhk.xml:



Phương thức qlhk.Save(/\*đường dẫn\*/) sẽ lưu ngược dữ liệu từ Xdocument về lại file XML.

Nhờ tính linh động hướng văn bản này của LINQ to XML, chúng ta thậm chí còn có thể thêm bớt các Attribute, Element con của Node bất kì. Ở ví dụ sau, ta sẽ thay đổi tên của Attribute MADINHDANH bằng cách xóa Attribute này rồi thêm một Attribute khác với tên MA và giá trị là giá trị của MADINHDANH:

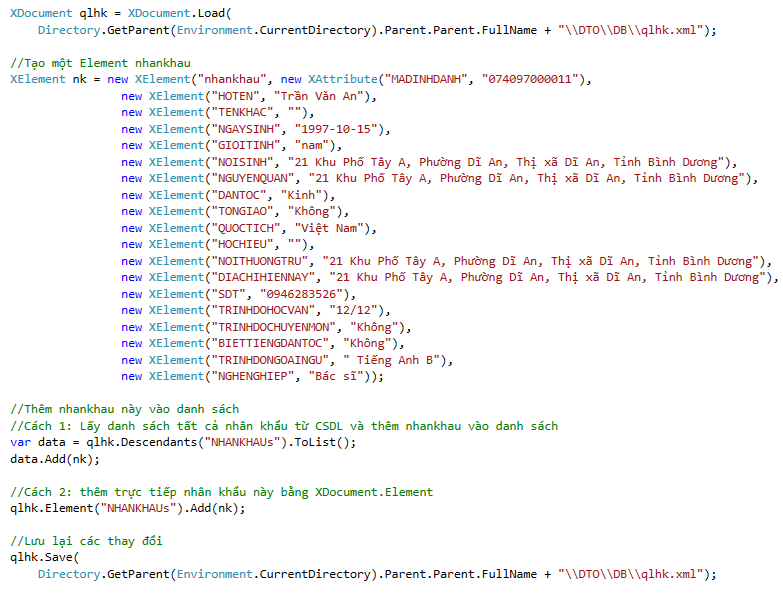


1. Thêm và xóa dữ liệu:

Ngoài việc cập nhật các dòng đã có trong CSDL, LINQ to XML còn cho phép chúng ta thêm và xóa dữ liệu. Ta có thể làm được điều này bằng việc thêm/bớt các đối tượng Element từ các kết quả truy vấn. LINQ to XML sẽ lưu vết lại các thao tác này, và sẽ lưu ngược lại các giá trị thay đổi này vào file XML khi phương thức Save() được gọi.

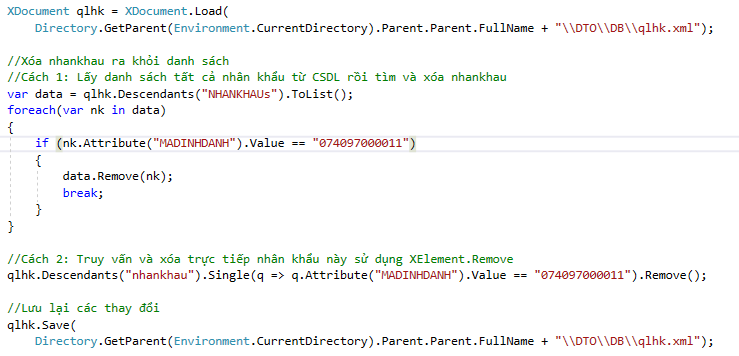
1. Thêm một nhankhau:

Chúng ta thực hiện thêm một nhankhau bằng cách lấy ra danh sách các nhankhau trong bảng NHANKHAUs và tạo một Element nhankhau rồi thêm nó vào danh sách đó:



1. Xóa một nhankhau:

Để xóa một nhankhau ta cũng lấy danh sách các nhankhau trong Elements NHANKHAUs, tìm và xóa nhankhau đó hoặc ta có thể truy vấn tới nhankhau đó rồi dùng phương thức Remove() để xóa bản thân nó ra khỏi XDocument:



## Tổng kết:

# LINQ to Object:

using **System.Linq**; **System.Collections.Generic**;

Thuật ngữ "LINQ to Objects" đề cập đến việc sử dụng các truy vấn LINQ với bất kỳ tập hợp IEnumerable hay IEnumerable (T),

## Các khái niệm trong LINQ to Object:

### Object:

Obj là…..

Sau đây là các đối tượng khác nhau mà chúng ta có thể sử dụng với LINQ.

* Linq to Strings
* Linq to String Array
* Linq to Int Array
* Linq to Files
* Linq to Lists

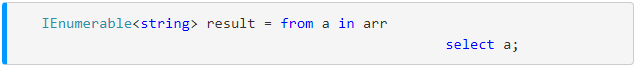
## Xây dựng chương trình với LINQ to Object:

### LINQ to String:

LINQ to String Array có nghĩa là viết các truy vấn LINQ trên mảng chuỗi để nhận dữ liệu cần thiết. Nếu chúng ta sử dụng truy vấn LINQ trên mảng chuỗi, chúng ta có thể dễ dàng nhận được các phần tử cần thiết mà không cần viết nhiều mã.

**Cú pháp**

Cú pháp truy vấn lấy ra phần tử của mảng string

****

**Ví dụ**

Lấy ra những phần tử từ mảng tuần tự string với tên bắt đầu bằng “s”

### :

## Tổng kết:

# Tổng quan về LinQ to XML:

|  |  |
| --- | --- |
| **Chủ đề** | **Miêu tả** |
| **Tổng quan LINQ to XML (C#)** | Cung cấp tổng quan về công nghệ LINQ to XML. |
| **LINQ to XML so với DOM (C#)** | So sánh LINQ to XML với Mô hình đối tượng tài liệu (DOM). |
| **LINQ to XML so với Các công nghệ XML khác** | So sánh LINQ to XML với các công nghệ phân tích và xử lý phân tích XML khác: XmlReader, XSLT, MSXML và XmlLite. |

XML đã được áp dụng rộng rãi như một cách để định dạng dữ liệu trong nhiều ngữ cảnh. Ví dụ: bạn có thể tìm thấy XML trên Web, trong các tệp cấu hình, trong các tệp Microsoft Office Word và trong cơ sở dữ liệu.

LINQ to XML là một cách tiếp cận được cập nhật, được thiết kế lại để lập trình với XML. Nó cung cấp các khả năng sửa đổi tài liệu trong bộ nhớ của Mô hình đối tượng tài liệu (DOM) và hỗ trợ các biểu thức truy vấn LINQ. Mặc dù các biểu thức truy vấn này khác về mặt cú pháp so với XPath, chúng cung cấp chức năng tương tự.

**Các lập trình viên LINQ to XML**

LINQ to XML nhắm vào nhiều lập trình viên. Đối với một lập trình viên trung bình chỉ muốn hoàn thành công việc, LINQ to XML giúp XML dễ dàng hơn bằng cách cung cấp trải nghiệm truy vấn tương tự như SQL. Chỉ cần một chút nghiên cứu, các lập trình viên có thể học cách viết các truy vấn ngắn gọn và mạnh mẽ bằng ngôn ngữ lập trình mà họ lựa chọn.

Các lập trình viên chuyên nghiệp có thể sử dụng LINQ to XML để tăng đáng kể năng suất của họ. Với LINQ to XML, họ có thể viết ít mã hơn, biểu cảm hơn, gọn hơn và mạnh mẽ hơn. Họ có thể sử dụng các biểu thức truy vấn từ nhiều miền dữ liệu cùng một lúc.

**LINQ to XML là gì?**

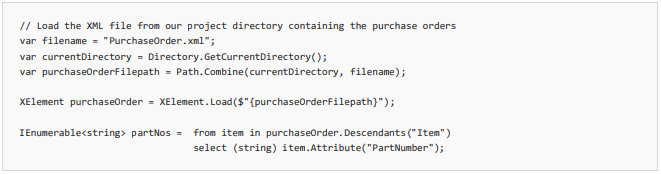
LINQ to XML là giao diện lập trình XML trong bộ nhớ hỗ trợ LINQ to phép bạn làm việc với XML từ bên trong các ngôn ngữ lập trình .NET Framework.

LINQ to XML giống như Mô hình Đối tượng Tài liệu (DOM) ở chỗ nó đưa tài liệu XML vào bộ nhớ. Bạn có thể truy vấn và sửa đổi tài liệu, và sau khi bạn sửa đổi nó, bạn có thể lưu nó vào một tệp hoặc tuần tự hóa nó và gửi nó qua Internet. Tuy nhiên, LINQ to XML khác với DOM: Nó cung cấp một mô hình đối tượng mới có trọng lượng nhẹ hơn và dễ làm việc hơn và tận dụng các tính năng ngôn ngữ trong C #.

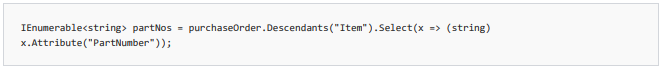
Ưu điểm quan trọng nhất của LINQ to XML là tích hợp với Truy vấn tích hợp ngôn ngữ (LINQ). Việc tích hợp này cho phép bạn viết các truy vấn trên tài liệu XML trong bộ nhớ để truy xuất các bộ sưu tập các phần tử và thuộc tính. Khả năng truy vấn của LINQ to XML có thể so sánh về chức năng (mặc dù không theo cú pháp) với XPath và XQuery. Việc tích hợp LINQ trong C # cung cấp khả năng gõ mạnh hơn, kiểm tra thời gian biên dịch và hỗ trợ trình gỡ lỗi được cải thiện.

Một ưu điểm khác của LINQ to XML là khả năng sử dụng các kết quả truy vấn làm tham số cho các hàm tạo đối tượng **XEuity** và **XAttribution** cho phép một cách tiếp cận mạnh mẽ để tạo cây XML. Cách tiếp cận này, được gọi là xây dựng chức năng, cho phép các lập trình viên dễ dàng chuyển đổi các cây XML từ hình dạng này sang hình dạng khác.

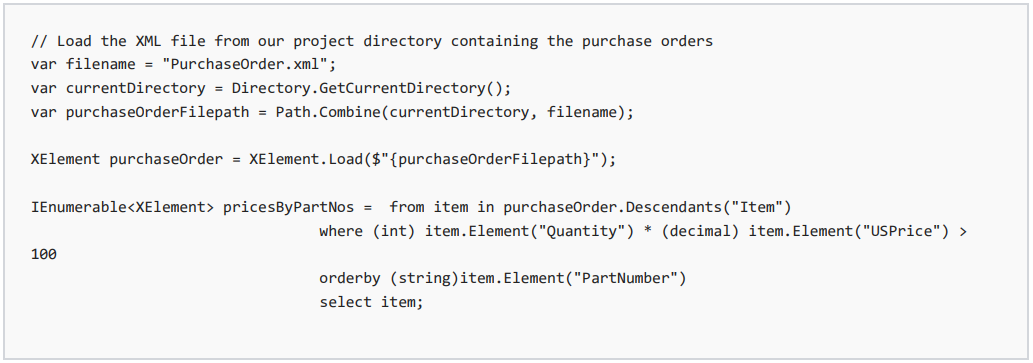
Ví dụ: bạn có thể có một đơn đặt hàng XML điển hình như được mô tả trong Tệp XML mẫu: Đơn đặt hàng tiêu biểu (LINQ to XML). Bằng cách sử dụng LINQ to XML, bạn có thể chạy truy vấn sau để có được giá trị thuộc tính số phần cho mọi thành phần vật phẩm trong đơn đặt hàng:



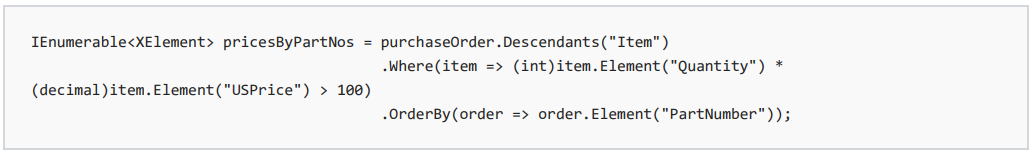
Viết lại dưới dạng cú pháp phương thức:

****

Một ví dụ khác, bạn có thể muốn một danh sách, được sắp xếp theo số phần, của các mục có giá trị lớn hơn $ 100. Để có được thông tin này, bạn có thể chạy truy vấn sau:

****

Một lần nữa, điều này có thể được viết lại dưới dạng cú pháp phương thức:

****

Ngoài các khả năng LINQ này, LINQ to XML cung cấp giao diện lập trình XML được cải tiến. Sử dụng LINQ to XML, bạn có thể:

* Tải XML từ **Files** và **streams.**
* Nối tiếp XML thành **Files** và **streams.**
* Tạo XML từ đầu bằng cách sử dụng chức năng xây dựng.
* Truy vấn XML bằng các trục giống như XPath.
* Thao tác với cây XML trong bộ nhớ bằng cách sử dụng các phương thức như **Add**, **Remove**, **ReplaceWith** và **SetValue**.
* Xác thực cây XML bằng XSD.
* Sử dụng kết hợp các tính năng này để chuyển đổi cây XML từ hình dạng này sang hình dạng khác.

**Tạo cây XML**

Một trong những lợi thế quan trọng nhất của việc lập trình với LINQ to XML là dễ dàng tạo các cây XML. Ví dụ: để tạo một cây XML nhỏ, bạn có thể viết mã như sau:



# LINQ to XML so với DOM (C #):

**Những cách mới để xây dựng cây XML**

Trong DOM W3C, bạn xây dựng cây XML từ dưới lên; nghĩa là bạn tạo một tài liệu, bạn tạo các phần tử và sau đó bạn thêm các phần tử vào tài liệu.

Ví dụ, sau đây sẽ là một cách điển hình để tạo cây XML bằng cách sử dụng triển khai DOM, **XmlDocument** của Microsoft:



Kiểu mã hóa này không cung cấp trực quan nhiều thông tin về cấu trúc của cây XML. LINQ to XML hỗ trợ cách tiếp cận này để xây dựng cây XML, nhưng cũng hỗ trợ một cách tiếp cận khác, xây dựng chức năng. Xây dựng chức năng sử dụng các hàm tạo **XEuity** và **XAttribution** để xây dựng cây XML.

Đây là cách bạn sẽ xây dựng cùng một cây XML bằng cách sử dụng chức năng xây dựng chức năng LINQ to XML:



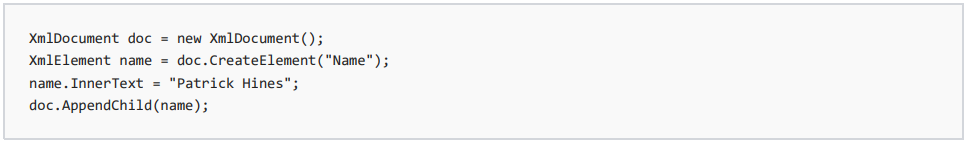
Lưu ý rằng việc thụt lề mã để xây dựng cây XML cho thấy cấu trúc của XML bên dưới. Để biết thêm thông tin, hãy xem **Tạo Cây XML (C #).**

**Làm việc trực tiếp với các phần tử XML**

Khi bạn lập trình với XML, trọng tâm chính của bạn thường là các phần tử XML và có lẽ là các thuộc tính. Trong LINQ to XML, bạn có thể làm việc trực tiếp với các thành phần và thuộc tính XML. Ví dụ: bạn có thể làm như sau:

* Tạo các phần tử XML mà không sử dụng một đối tượng tài liệu nào cả. Điều này đơn giản hóa việc lập trình khi bạn phải làm việc với các đoạn cây XML.
* Tải các đối tượng T: System.Xml.Linq.XElament trực tiếp từ tệp XML.
* Nối tiếp các đối tượng T: System.Xml.Linq.XElement vào một tệp hoặc một luồng.

So sánh điều này với W3C DOM, trong đó tài liệu XML được sử dụng làm vùng chứa logic cho cây XML.Trong DOM, các nút XML, bao gồm các thành phần và thuộc tính, phải được tạo trong ngữ cảnh của tài liệu XML. Đây là một đoạn mã để tạo thành phần tên trong DOM:

****

Nếu bạn muốn sử dụng một yếu tố trên nhiều tài liệu, bạn phải nhập các nút trên các tài liệu. LINQ to XML tránh lớp phức tạp này.

Khi sử dụng LINQ to XML, bạn chỉ sử dụng lớp **XDocument** nếu bạn muốn thêm một nhận xét hoặc hướng dẫn xử lý ở cấp gốc của tài liệu.

**Xử lý đơn giản tên và namespace**

Xử lý tên, namespace và tiền tố namespace nói chung là một phần phức tạp của lập trình XML. LINQ to XML đơn giản hóa các tên và namespace bằng cách loại bỏ yêu cầu xử lý các tiền tố namespace. Nếu bạn muốn kiểm soát tiền tố namespace, bạn có thể. Nhưng nếu bạn quyết định không kiểm soát rõ ràng các tiền tố namespace, LINQ to XML sẽ gán các tiền tố namespace trong quá trình tuần tự hóa nếu chúng được yêu cầu hoặc sẽ tuần tự hóa bằng cách sử dụng các namespace mặc định nếu không. Nếu namespace mặc định được sử dụng, sẽ không có tiền tố namespace trong tài liệu kết quả. Để biết thêm thông tin, hãy xem **Làm việc với Namespace XML (C #).**

Một vấn đề khác với DOM là nó không cho phép bạn thay đổi tên của một nút. Thay vào đó, bạn phải tạo một nút mới và sao chép tất cả các nút con vào nó, làm mất danh tính nút gốc. LINQ to XML tránh được vấn đề này bằng cách cho phép bạn đặt thuộc tính **XName** trên một nút.

**Hỗ trợ phương thức tĩnh để tải XML**

LINQ to XML cho phép bạn tải XML bằng cách sử dụng các phương thức tĩnh, thay vì các phương thức cá thể. Điều này đơn giản hóa việc tải và phân tích cú pháp. Để biết thêm thông tin, hãy xem Cách: **Tải XML từ một tệp (C #).**

**Xóa hỗ trợ cho các công trình xây dựng DTD**

LINQ to XML tiếp tục đơn giản hóa việc lập trình XML bằng cách loại bỏ hỗ trợ cho các thực thể và tham chiếu thực thể. Việc quản lý các thực thể rất phức tạp và hiếm khi được sử dụng. Loại bỏ hỗ trợ của họ làm tăng hiệu suất và đơn giản hóa giao diện lập trình. Khi cây LINQ to XML được điền, tất cả các thực thể DTD được mở rộng.

**Hỗ trợ cho các Fragment**

LINQ to XML không cung cấp tương đương cho lớp XmlDocumentFragment. Tuy nhiên, trong nhiều trường hợp, khái niệm XmlDocumentFragment có thể được xử lý bằng kết quả của một truy vấn được nhập cho **IEnumerable <T>** của **XNode** hoặc **IEnumerable <T>** của **XEuity**.

**Hỗ trợ cho XPathNavigator**

LINQ to XML cung cấp hỗ trợ cho **XPathNavigator** thông qua các phương thức mở rộng trong namespace **System.Xml.XPath**. Để biết thêm thông tin, hãy xem **System.Xml.XPath.Extensions**.

**Hỗ trợ cho không gian trắng và thụt lề**

LINQ to XML xử lý khoảng trắng đơn giản hơn DOM. Một kịch bản phổ biến là đọc XML thụt lề, tạo cây XML trong bộ nhớ mà không có bất kỳ nút văn bản khoảng trắng nào (nghĩa là không bảo toàn khoảng trắng), thực hiện một số thao tác trên XML và sau đó lưu XML bằng cách thụt lề. Khi bạn tuần tự hóa XML với định dạng, chỉ có khoảng trắng đáng kể trong cây XML được giữ nguyên. Đây là hành vi mặc định cho LINQ to XML.

Một kịch bản phổ biến khác là đọc và sửa đổi XML đã được thụt lề có chủ ý. Bạn có thể không muốn thay đổi vết lõm này theo bất kỳ cách nào. Trong LINQ to XML, bạn có thể thực hiện điều này bằng cách duy trì khoảng trắng khi bạn tải hoặc phân tích cú pháp XML và vô hiệu hóa định dạng khi bạn tuần tự hóa XML.

LINQ to XML lưu trữ khoảng trắng dưới dạng nút **XText**, thay vì có loại nút **Whitespace** chuyên dụng, như DOM thực hiện.

**Hỗ trợ chú thích**

Các phần tử LINQ to XML hỗ trợ một bộ chú thích mở rộng. Điều này hữu ích để theo dõi thông tin linh tinh về một phần tử, chẳng hạn như thông tin lược đồ, thông tin về việc phần tử đó có bị ràng buộc với UI hay bất kỳ loại thông tin cụ thể nào khác của ứng dụng hay không. Để biết thêm thông tin, hãy xem **Chú thích LINQ to XML.**

**Hỗ trợ thông tin lược đồ**

LINQ to XML cung cấp hỗ trợ xác thực XSD thông qua các phương thức mở rộng trong namespace **System.Xml.Schema**. Bạn có thể xác thực rằng một cây XML tuân thủ XSD. Bạn có thể điền vào cây XML bằng infoset xác thực sau lược đồ (PSVI). Để biết thêm thông tin, hãy xem Cách: **Xác thực bằng XSD** và **Tiện ích mở rộng**.

# LINQ to XML so với các công nghệ XML khác:

Chúng ta sẽ so sánh LINQ với XML với các công nghệ XML sau: **XmlReader**, XSLT, MSXML và XmlLite. Thông tin này có thể giúp bạn quyết định sử dụng công nghệ nào.

Để so sánh LINQ với XML với Mô hình đối tượng tài liệu (DOM), hãy xem **LINQ với XML so với DOM (C #).**

**LINQ to XML so với XmlReader**

**XmlReader** là một trình phân tích cú pháp nhanh, không chuyển tiếp, bộ đệm.

LINQ to XML được triển khai trên **XmlReader** và chúng được tích hợp chặt chẽ. Tuy nhiên, bạn cũng có thể sử dụng **XmlReader**.

Ví dụ: giả sử bạn đang xây dựng một dịch vụ Web sẽ phân tích hàng trăm tài liệu XML mỗi giây và các tài liệu có cùng cấu trúc, nghĩa là bạn chỉ phải viết một triển khai mã để phân tích cú pháp XML. Trong trường hợp này, có lẽ bạn sẽ muốn sử dụng **XmlReader**.

Ngược lại, nếu bạn đang xây dựng một hệ thống phân tích nhiều tài liệu XML nhỏ hơn và mỗi tài liệu khác nhau, bạn sẽ muốn tận dụng các cải tiến năng suất mà LINQ sang XML cung cấp.

**LINQ to XML so với XSLT**

Cả LINQ sang XML và XSLT đều cung cấp các khả năng chuyển đổi tài liệu XML mở rộng. XSLT là một cách tiếp cận khai báo dựa trên quy tắc. Các lập trình viên XSLT nâng cao viết XSLT theo kiểu lập trình chức năng nhấn mạnh cách tiếp cận không trạng thái. Các biến đổi có thể được viết bằng các hàm thuần túy được thực hiện mà không có tác dụng phụ. Cách tiếp cận dựa trên quy tắc hoặc chức năng này không quen thuộc với nhiều lập trình viên và có thể khó học và tốn thời gian để tìm hiểu.

XSLT có thể là một hệ thống rất năng suất mang lại các ứng dụng hiệu suất cao. Ví dụ: một số công ty web lớn sử dụng XSLT như một cách để tạo HTML từ XML được lấy từ nhiều kho dữ liệu khác nhau. Công cụ XSLT được quản lý biên dịch mã XSLT thành CLR và thậm chí còn hoạt động tốt hơn trong một số trường hợp so với công cụ XSLT gốc. Tuy nhiên, XSLT không tận dụng kiến thức C # và Visual Basic mà nhiều lập trình viên có. Nó yêu cầu các lập trình viên viết mã bằng một ngôn ngữ lập trình phức tạp và khác biệt. Sử dụng hai hệ thống phát triển không tích hợp như C # (hoặc Visual Basic) và XSLT dẫn đến các hệ thống phần mềm khó phát triển và bảo trì hơn.

Sau khi bạn đã thành thạo các biểu thức truy vấn LINQ to XML, các phép biến đổi LINQ sang XML là một công nghệ mạnh mẽ, dễ sử dụng. Về cơ bản, bạn hình thành tài liệu XML của mình bằng cách sử dụng chức năng xây dựng, lấy dữ liệu từ nhiều nguồn khác nhau, xây dựng các đối tượng **Xlement** một cách linh hoạt và lắp ráp toàn bộ thành một cây XML mới. Việc chuyển đổi có thể tạo ra một tài liệu hoàn toàn mới. Xây dựng các phép biến đổi trong LINQ thành XML tương đối dễ dàng và trực quan và mã kết quả có thể đọc được. Điều này làm giảm chi phí phát triển và bảo trì.

LINQ to XML không nhằm thay thế XSLT. XSLT vẫn là công cụ được lựa chọn cho các phép biến đổi XML phức tạp và tập trung vào tài liệu, đặc biệt nếu cấu trúc của tài liệu không được xác định rõ.

XSLT có lợi thế là tiêu chuẩn của World Wide Web Consortium (W3C). Nếu bạn có một yêu cầu là bạn chỉ sử dụng các công nghệ là tiêu chuẩn, XSLT có thể phù hợp hơn.

XSLT là XML và do đó có thể được lập trình thao tác.

**LINQ to XML so với MSXML**

MSXML là công nghệ dựa trên COM để xử lý XML được bao gồm trong Microsoft Windows. MSXML cung cấp một triển khai nguyên gốc của DOM với sự hỗ trợ cho XPath và XSLT. Nó cũng chứa bộ phân tích cú pháp dựa trên sự kiện, bộ nhớ đệm SAX2.

MSXML hoạt động tốt, được bảo mật theo mặc định trong hầu hết các kịch bản và có thể được truy cập trong Internet Explorer để thực hiện xử lý XML phía máy khách trong các ứng dụng kiểu AJAX. MSXML có thể được sử dụng từ bất kỳ ngôn ngữ lập trình nào hỗ trợ COM, bao gồm C ++, JavaScript và Visual Basic 6.0.

MSXML không được khuyến nghị sử dụng trong mã được quản lý dựa trên thời gian chạy ngôn ngữ chung (CLR).

**LINQ to XML so với XmlLite**

XmlLite là một bộ đệm không lưu, chỉ chuyển tiếp, kéo trình phân tích cú pháp. Các lập trình viên chủ yếu sử dụng XmlLite với C ++. Lập trình viên không nên sử dụng XmlLite với mã được quản lý.

Ưu điểm chính của XmlLite là nó là một trình phân tích cú pháp XML nhẹ, nhanh, an toàn trong hầu hết các tình huống. Diện tích bề mặt đe dọa của nó là rất nhỏ. Nếu bạn phải phân tích các tài liệu không đáng tin cậy và bạn muốn bảo vệ chống lại các cuộc tấn công như từ chối dịch vụ hoặc tiếp xúc với dữ liệu, XmlLite có thể là một lựa chọn tốt.

XmlLite không được tích hợp với Truy vấn tích hợp ngôn ngữ (LINQ). Nó không mang lại sự cải thiện năng suất lập trình viên là động lực thúc đẩy đằng sau LINQ.

# Tài liệu tham khảo:

* Lịch sử của truy vấn tích hợp ngôn ngữ: ***Data Access API of the Day****,**Mike Pizzo* <https://blogs.msdn.microsoft.com/data/2006/12/05/data-access-api-of-the-day-part-i/>
* MSDN .Net Language-Integrated Query: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/>
* Hướng dẫn khởi tạo và sử dụng Object Relational Designer(O/R Designer): <https://docs.microsoft.com/en-us/previous-versions/bb384429(v=vs.120)>
* LINQ to SQL cơ bản: <https://yinyangit.wordpress.com/2011/08/11/linq-to-sql-basic-concepts-object-relational-mapping-entity-class-association-and-datacontext/>
* MSDN LINQ to SQL: <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/>
* Mẫu truy vấn LINQ: <https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b>
* ***C# in Depth Third Edition****, JohnSkeet*
* ***Practical Database Programming With Visual C#.NET***, *Ying Bai*
* ***Professional LINQ****, Scott Klein*