

Sử dụng LINQ to SQL (LINQ to SQL phần 1)

LINQ to SQL là gì?

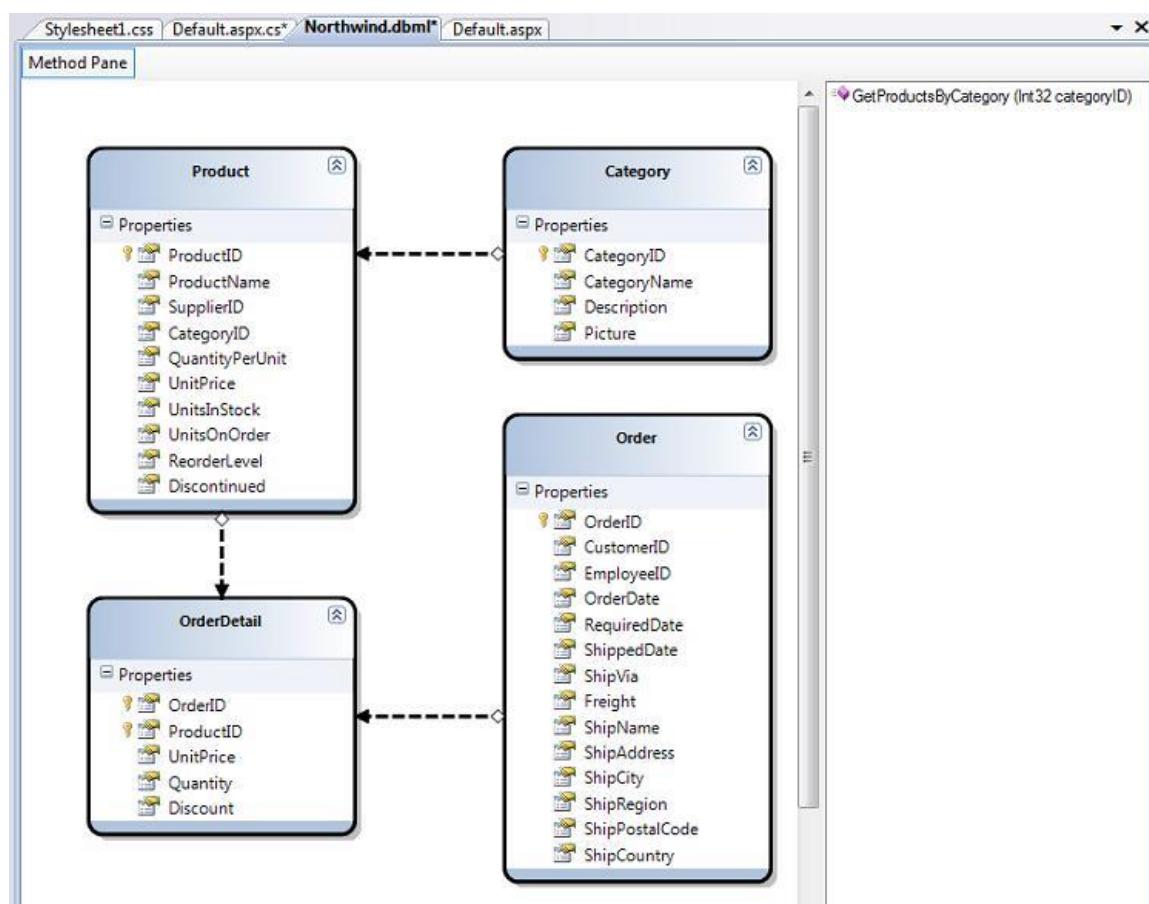
LINQ to SQL là một phiên bản hiện thực hóa của O/RM (object relational mapping) có bên trong .NET Framework bản “Orcas” (nay là .NET 3.5), nó cho phép bạn mô hình hóa một cơ sở dữ liệu dùng các lớp .NET. Sau đó bạn có thể truy vấn cơ sở dữ liệu (CSDL) dùng LINQ, cũng như cập nhật/thêm/xóa dữ liệu từ đó.

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

Mô hình hóa CSDL dùng LINQ to SQL:

Visual Studio “Orcas” 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. Bài viết sau sẽ đi sâu hơn vào cách dùng trình thiết kế này (bạn cũng có thể xem đoạn video này để xem cách tôi tạo một mô hình LINQ to SQL).

Bằng cách dùng trình thiết kế LINQ to SQL, tôi có thể 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 thể: 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.

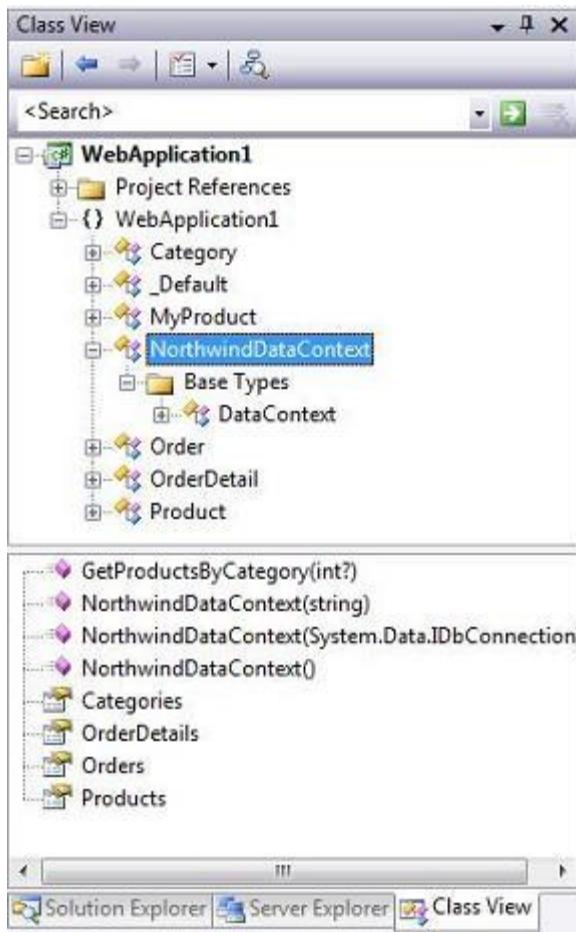
Các mũi tên giữa bốn lớp thực thể trên biểu diễn quan hệ giữa các thực thể khác nhau, chúng được tạo ra dựa trên các mối quan hệ primary-key/foreign-key trong CSDL. Hướng của mũi tên chỉ ra mối quan hệ là một – một hay một – nhiều. Các thuộc tính tương ứng sẽ được thêm vào các lớp thực thể trong các 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 phải bên trong trình thiết kế LINQ to SQL ở 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 categoryID và trả về một chuỗi các Product. Chúng ta sẽ xem bằng cách nào có thể gọi được thủ tục này trong một đoạn code bên dưới.

Tìm hiểu 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 thể và quan hệ bên trong CSDL mà chúng ta vừa mô hình hóa. Cứ mỗi 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 đổi. Lớp DataContext được tạo sẽ có các thuộc tính để biểu diễn mỗi 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 ví dụ LINQ to SQL

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

1) Lấy các Product từ CSDL

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

```
NorthwindDataContext db = new NorthwindDataContext();
var products = from p in db.Products
               where p.Category.CategoryName == "Beverages"
               select p;
```

2) Cập nhật một sản phẩm trong CSDL

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

```
NorthwindDataContext db = new NorthwindDataContext();
Product product = db.Products.Single(p => p.ProductName == "Toy 1");
product.UnitPrice = 99;
product.UnitsInStock = 5;
db.SubmitChanges();
```

3) Chèn thêm một phân loại mới và hai sản phẩm vào CSDL

Đoạn mã dưới đây biểu diễn cách tạo một phân loại mới, và tạo hai sản phẩm mới và đưa chúng vào trong phân loại đã tạo. Cả ba sau đó sẽ được đưa vào cơ sở dữ liệu.

Chú ý rằng tôi không cần phải tự quản 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 đổi bằng InsertOnSubmit trong phiên bản hiện tại)

```
NorthwindDataContext db = new NorthwindDataContext();
// Create new Category and Products
Category category = new Category();
category.CategoryName = "Scott's Toys";
Product product1 = new Product();
product1.ProductName = "Toy 1";
Product product2 = new Product();
product2.ProductName = "Toy 2";
// Associate Products with Category
category.Products.Add(product1);
category.Products.Add(product2);
// Add category to database and save changes
db.categories.Add(category);
db.SubmitChanges();
```

4) Xóa các sản phẩm

Đoạn mã sau sẽ biểu diễn cách xóa tất cả các sản phẩm Toy khỏi CSDL:

(RemoveAll đã được thay đổi bằng DeleteAllOnSubmit trong phiên bản hiện tại)

```

NorthwindDataContext db = new NorthwindDataContext();

var toyProducts = from p in db.Products
                  where p.ProductName.Contains("Toy")
                  select p;

db.Products.RemoveAll(toyProducts);

db.SubmitChanges();

```

5) Gọi một thủ tục

Đoạn mã dưới đây biểu diễn cách lấy các thực thể Product mà không dùng cú pháp của LINQ, mà gọi đến thủ tục “GetProductsByCategory” chúng ta đã thêm vào trước đây. Nhớ rằng một khi đã lấy về kết quả, tôi có thể cập nhật/xóa và sau đó gọi db.SubmitChanges() để cập nhật các thay đổi trở lại CSDL.

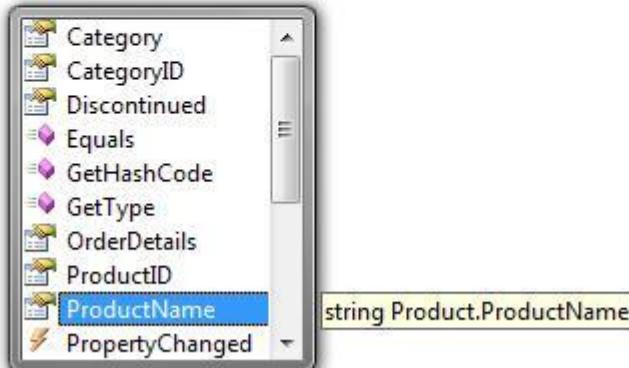
```

NorthwindDataContext db = new NorthwindDataContext();

var products = db.GetProductsByCategory(5);

foreach (Product product in products)
{
    product.
}

```



6) Lấy các sản phẩm và phân trang

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

```

NorthwindDataContext db = new NorthwindDataContext();

var products = (from p in db.Products
                 where p.Category.CategoryName.StartsWith("c")
                 select p).Skip(200).Take(10);

```

Tổng kết

LINQ to SQL cung cấp một cách hay, rõ ràng để mô hình hóa lớp dữ liệu trong ứng dụng của bạn. Một khi đã định nghĩa mô hình dữ liệu, bạn có thể dễ dàng thực hiện các câu truy vấn cũng như cập nhật, xóa, sửa dữ liệu một cách hiệu quả.

Hãy vọng những hướng dẫn và ví dụ mẫu ở trên đã giúp bạn làm quen với LINQ. Tôi sẽ tiếp tục các bài viết này để giúp bạn khám phá LINQ to SQL một cách chi tiết hơn.

Định nghĩa các lớp mô hình dữ liệu (LINQ to SQL phần 2)

Trong phần một, tôi 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 bài viết đầu tiên, tôi 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

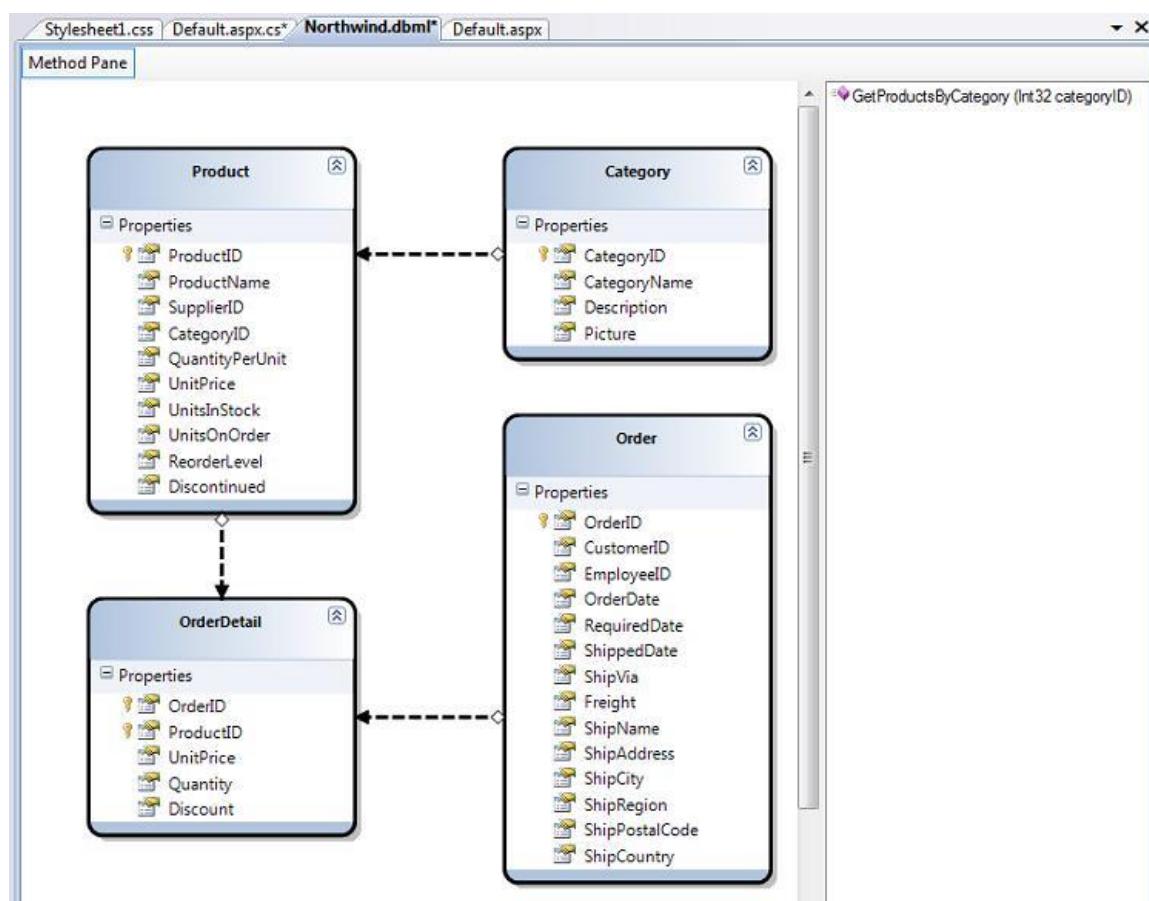
Cách chèn và tạo quan hệ các dòng trong một CSDL

Cách xóa các dòng trong một CSDL

Cách gọi một thủ tục

Cách lấy dữ liệu và phân trang trên server

Tôi 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:



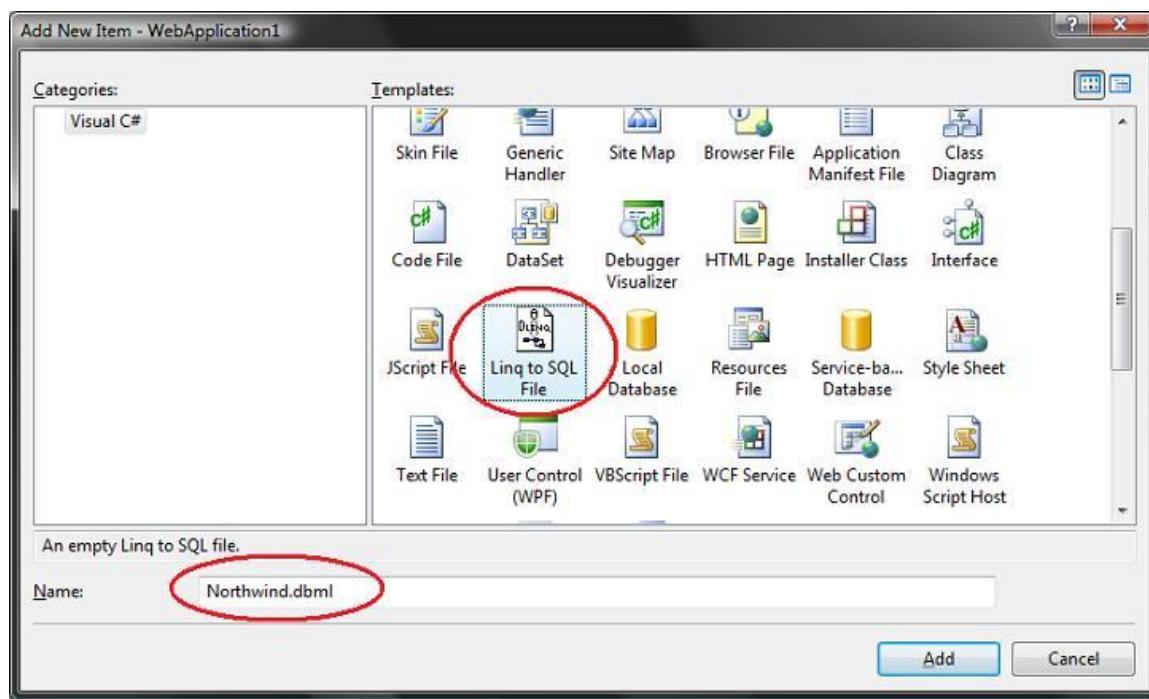
Trong bài này, tôi sẽ đi vào chi tiết cách tạo ra một mô hình dữ liệu LINQ to SQL giống như trên.

LINQ to SQL, cũng như LINQ to SQL, và tất cả các tính năng khác mà tôi đã nói đến trong loạt bài này sẽ được coi như một phần của .NET 3.5 và Visual Studio “Orcas” (nay là Visual Studio 2008).

Bạn có thể làm theo tất cả các bước dưới đây bằng cách tải về hoặc Visual Studio 2008 hoặc Visual Web Developer Express. Cả hai đều có thể được cài đặt và dùng đồng thời với Visual Studio 2005.

Tạo ra một mô hình dữ liệu LINQ to SQL

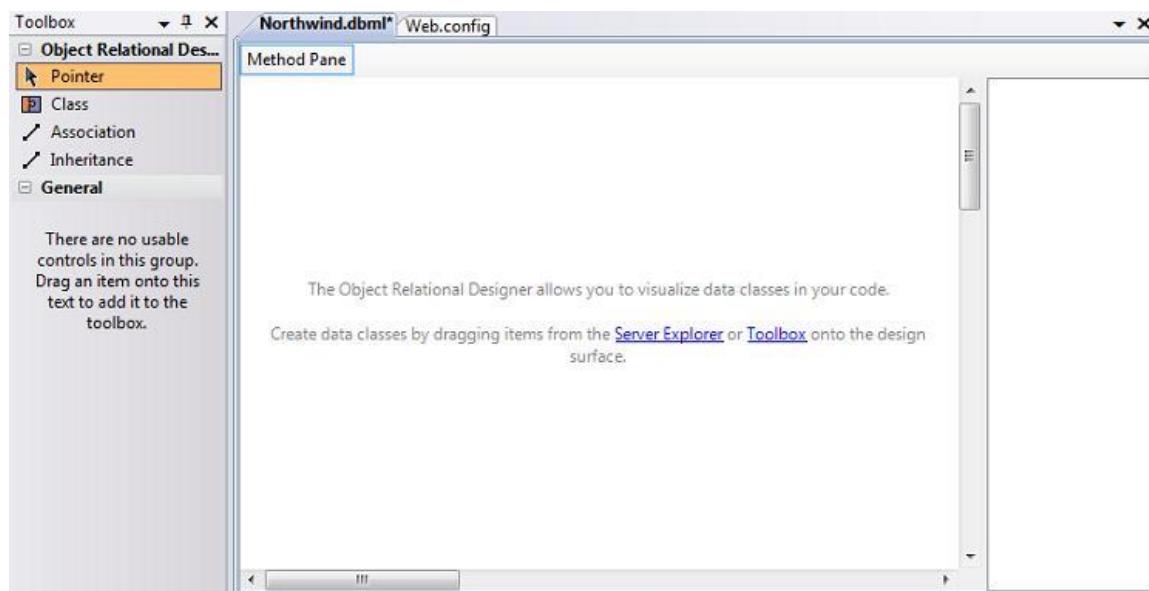
Bạn có thể thêm một mô hình dữ liệu LINQ to SQL và một dự án ASP.NET, Class Library hay Windows bằng cách dùng tùy chọn “Add New Item” bên trong Visual Studio và chọn “LINQ to SQL”:



Việc chọn mục “LINQ to SQL” sẽ khởi chạy LINQ to SQL designer, và cho phép bạn mô hình hóa các lớp mà nó biểu diễn một CSDL quan hệ. Nó cũng sẽ tạo ra một lớp kiểu “DataContext”, trong đó có các thuộc tính để biểu diễn mỗi bảng mà chúng ta mô hình hóa trong CSDL, cũng như các phương thức cho mỗi Stored Procedure mà chúng ta mô hình hóa. Như tôi đã mô tả trong phần 1 của loạt bài này, 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 cửa sổ thiết kế LINQ to SQL, và cũng là cái mà bạn sẽ thấy ngay khi tạo ra một mô hình dữ liệu LINQ to SQL:



Các lớp thực thể

LINQ to SQL cho phép bạn mô hình hóa các lớp ánh xạ vào 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 CSDL. Các thuộc tính của các lớp thông thường ánh xạ vào các cột trong bảng. Mỗi instance của một lớp thực thể biểu diễn một dòng trong bảng.

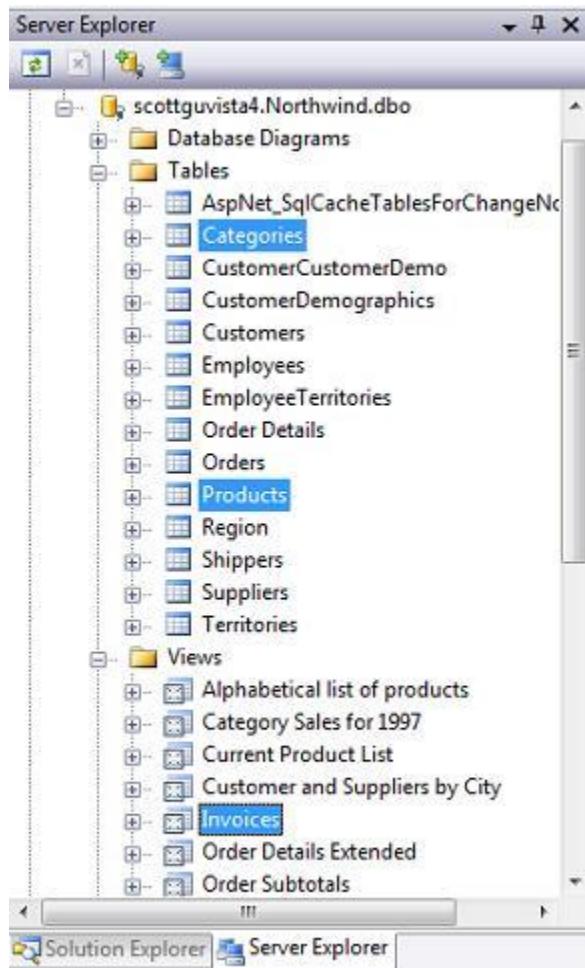
Các lớp thực thể trong LINQ to SQL không cần phải kế thừa từ một lớp đặc biệt nào khác, điều đó cho phép bạn có thể cho phép chúng thừa kế từ bất cứ đối tượng nào bạn muốn. Tất cả các lớp được tạo ra dùng LINQ to SQL designer đều được định nghĩa như “partial class” – có nghĩa là bạn có thể viết thêm code để thêm vào các thuộc tính, phương thức và sự kiện cho chúng.

Không giống như chức năng DataSet/TableAdapter có trong VS 2005, khi dùng LINQ to SQL designer, bạn không cần chỉ ra câu truy vấn SQL được dùng để tạo ra mô hình và lớp truy xuất dữ liệu.

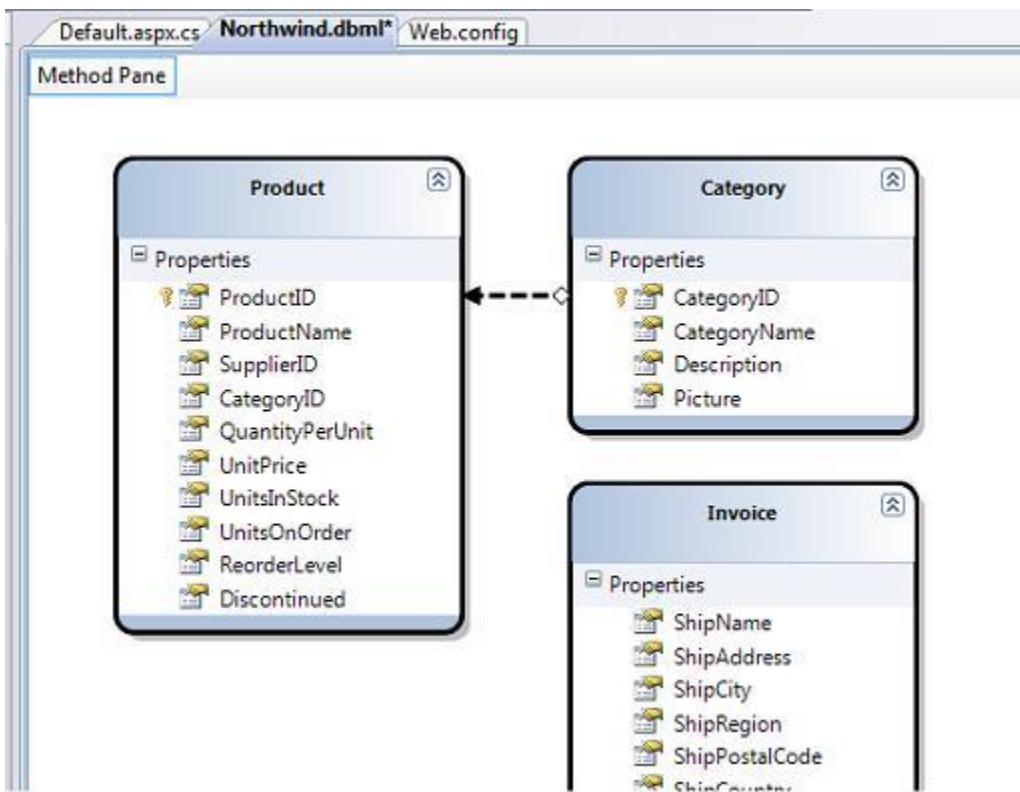
Thay vào đó, bạn tập trung chủ yếu vào việc định nghĩa các lớp thực thể, cách chúng ánh xạ vào CSDL, và mối quan hệ giữa chúng. Trình LINQ to SQL cung cấp mà bạn dùng sẽ đảm bảo việc sinh ra các lệnh SQL thích hợp vào lúc chạy khi bạn tương tác và làm việc với các thực thể dữ liệu. Bạn có thể dùng cú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

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

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



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



Dùng các lớp mô hình hóa dữ liệu ở trên, bạn có thể chạy tất cả các đoạn lệnh mẫu được một tả trong phần 1 của loạt bài này. Tôi không cần thêm bất kỳ đoạn code nào hay cấu hình để có thể thực hiện được các thao tác query, insert, update, delete và phân trang.

Cách đặt tên và ngữ pháp số nhiều

Một trong những thứ bạn đã nghe nhắc đến khi dùng LINQ to SQL là nó có thể tự động chuyển tên bảng và cột thành dạng số nhiều khi tạo các lớp thực thể. Lấy ví dụ: Bảng “Products” trong ví dụ của chúng ta tạo ra lớp “Product”, cũng như bảng “Categories” tạo ra lớp “Category”. Cách đặt tên này giúp mô hình của bạn thống nhất với quy ước đặt tên trong .NET.

Nếu không thích tên lớp hay tên thuộc tính do trình designer sinh ra, bạn vẫn có thể sửa lại thành bất cứ tên nào bạn 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.

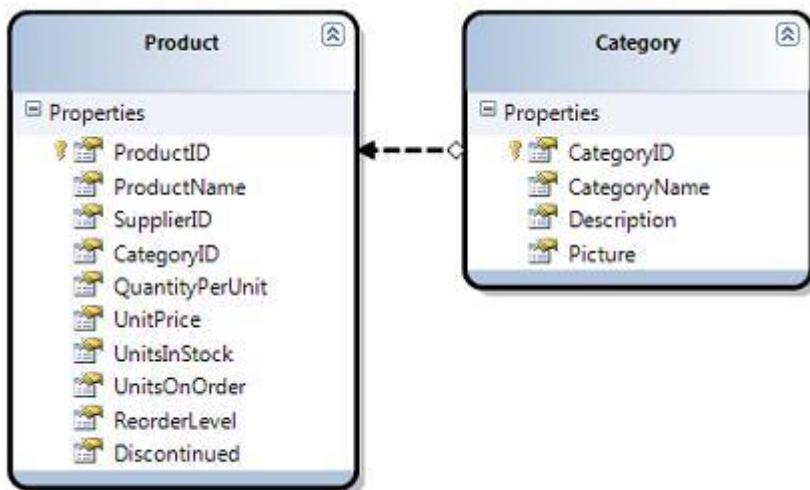


Khả năng đặt tên cho các thực thể/thuộc tính/quan hệ khác với tên trong CSDL rất hữu dụng trong một số trường hợp, ví dụ:

1. Khi tên bảng/cột trong CSDL bị thay đổi. Bởi vì mô hình thực thể của bạn có thể có tên khác với tên trong CSDL, do vậy bạn có thể 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, bạn có thể đặt tên chúng thành “LastName” và “FirstName” trong lớp thực thể và viết các lệnh để dùng với nó (mà không cần đổi tên các cột trong CSDL).

Quan hệ giữa các thực thể

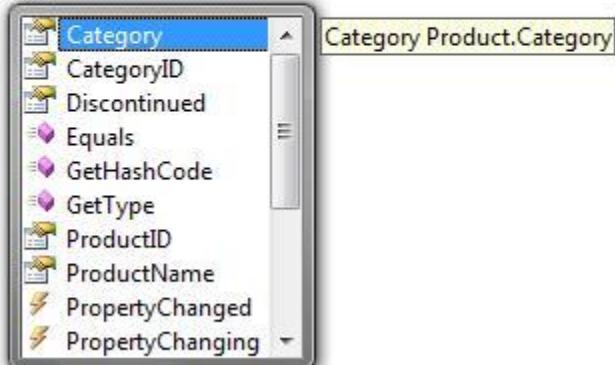
Khi bạn kéo thả các đối tượng từ Server Explorer lên trên cửa sổ LINQ to SQL designer, VS sẽ tự động xác định các mối quan hệ primary key/foreign key giữa các đối tượng, và tự động tạo các quan hệ mặc nhiên giữa các lớp thực thể khác nhau mà nó đã tạo. Ví dụ, khi bạn thêm cả hai bảng Products và Categories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



Mỗi quan hệ trên sẽ làm lớp thực thể Product có thêm một thuộc tính là Category, bạn có thể dùng để truy cập vào thực thể Category của một Product. Nó cũng làm lớp Category có thêm thuộc tính “Products”, đây là một tập hợp cho phép bạn lấy ra tất cả các Product có trong Category đó.

```

NorthwindDataContext db = new NorthwindDataContext();
Product product = db.Products.single(p => p.ProductID == 17);
string catName = product.
  
```

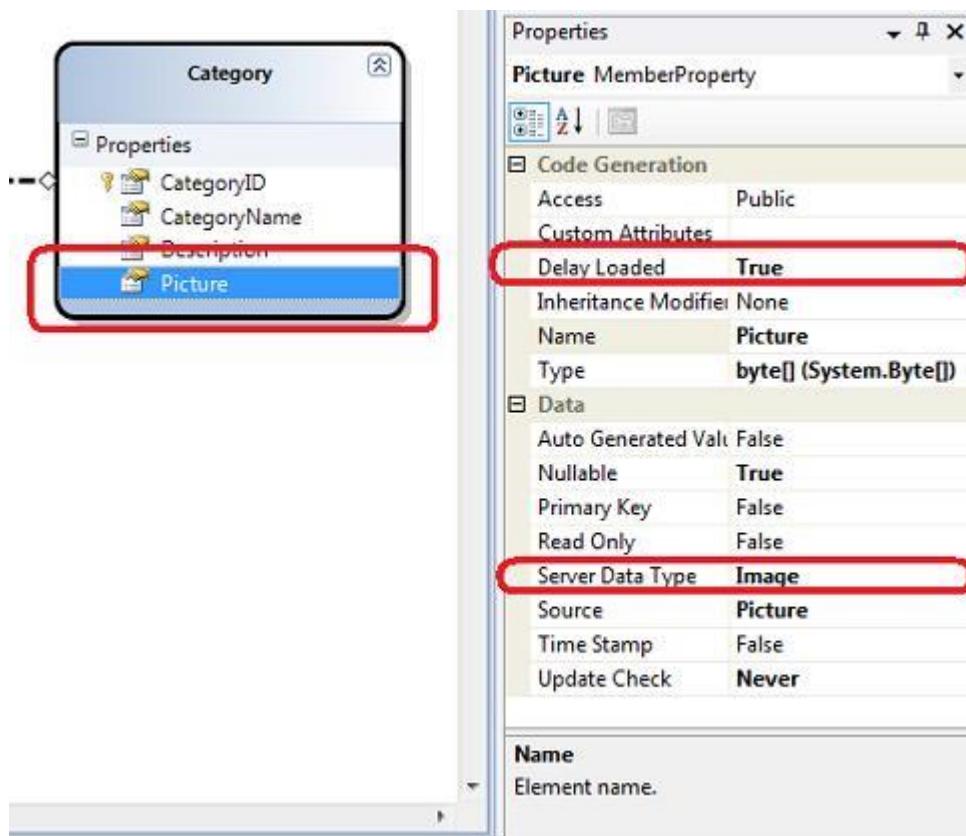


Nếu bạn không thích cách mà trình thiết kế đã mô hình hóa hoặc đặt tên, bạn hoàn toàn có thể chỉnh sửa lại. Chỉ cần click lên mũi tên chỉ ra quan hệ trên của sổ soạn thảo và truy cập vào các thuộc tính của nó thông qua bảng thuộc tính để đổi tên, chỉnh sửa hoặc thậm chí xóa nó.

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 delay/lazy loading). Bạn có thể 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.

Tôi có thể cấu hình thuộc tính Picture để nó chỉ được nạp khi dùng đến bằng cách đặt thuộc tính Delay Loaded thành true:



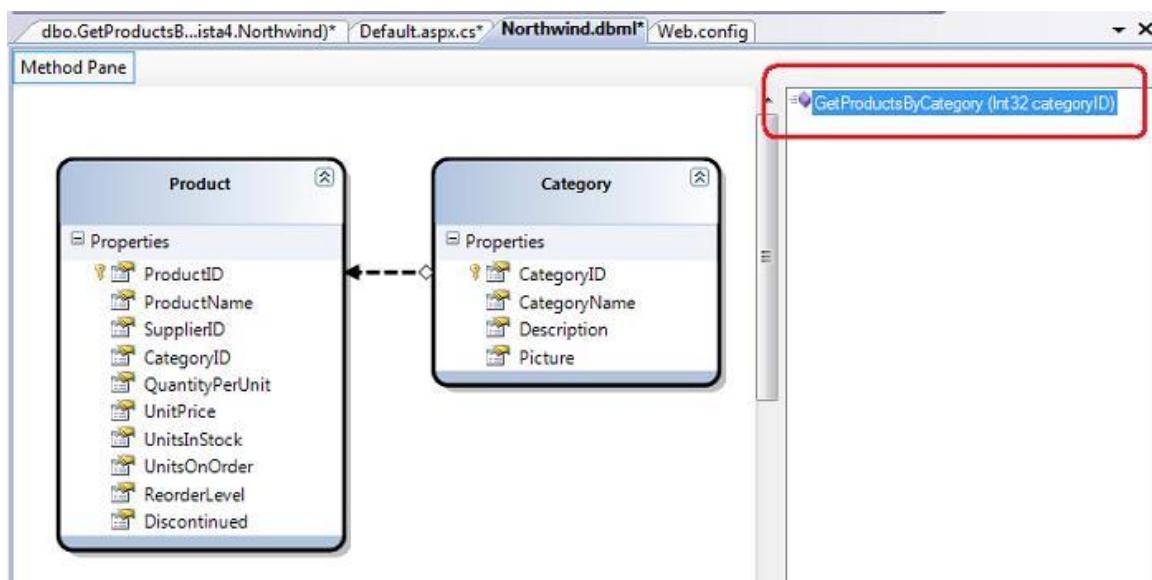
Ghi chú: Thay vì cấu hình prefetch/delay load trên các thực thể, bạn cũng có thể đặt lại thông qua các lệnh khi bạn thực hiện các câu truy vấn LINQ trên lớp thực thể đó (tôi sẽ hướng dẫn cách làm điều này trong bài viết sau của loạt bài này).

Dùng các Stored Procedure

LINQ to SQL cho phép bạn có thể 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 sản phẩm dựa trên một CategoryID:

```
dbo.GetProductsB...ista4.Northwind)* Default.aspx.cs* Northwind.dbml Web.config
ALTER PROCEDURE dbo.GetProductsByCategory
(
    @categoryID int
)
AS
SELECT * from Products where CategoryID=@categoryID
```

Tôi có thể 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 tôi thả SPROC lên trên thực thể “Product”, LINQ to SQL designer sẽ khai báo SPROC để trả về một tập kết quả có kiểu IEnumerable <product></product>:



Sau đó tôi có thể dùng cú pháp LINQ to SQL hay gọi thẳng phương thức ở trên để lấy về các thực thể từ CSDL:

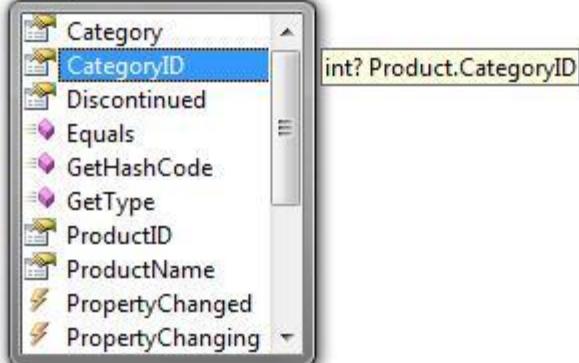
```

NorthwindDataContext db = new NorthwindDataContext();

// Retrieve products based on an adhoc query
IEnumerable<Product> products = from p in db.Products
                                   where p.CategoryID == 1
                                   select p;

// Retrieve products instead using a SPROC method
products = db.GetProductsByCategory(1);

// Iterate over results
foreach (Product product in products)
{
    product.|
```



Dùng SPROCS để cập nhật/xóa/thêm dữ liệu

Mặc nhiên LINQ to SQL sẽ tự động tạo ra các biểu thức SQL phù hợp cho bạn mỗi khi 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”:

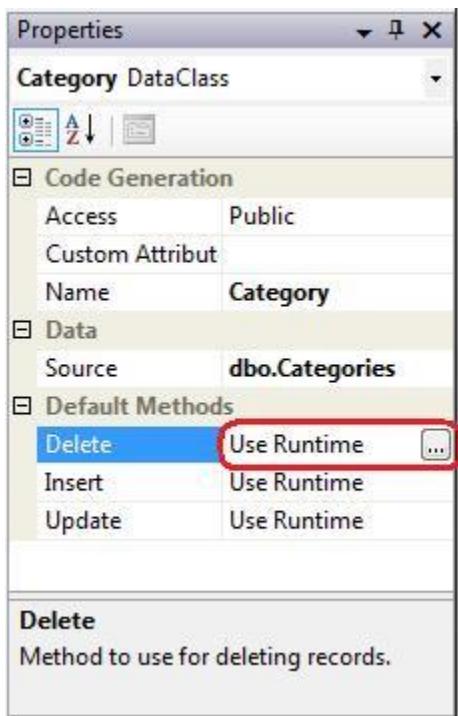
```

NorthwindDataContext db = new NorthwindDataContext();

Product product = db.Products.Single(p => p.ProductName == "Toy 1");
product.UnitPrice = 99;
product.UnitsInStock = 5;
db.SubmitChanges();
```

Mặc nhiên, LINQ to SQL sẽ tạo và thực thi phát biểu UPDATE tương ứng khi bạn xác nhận thay đổi (tôi sẽ nói thêm về vấn đề này trong những bài viết khác).

Bạn cũng có thể định nghĩa và dùng các thủ tục INSERT, UPDATE, DELETE nếu muốn. Để cấu hình, click lên một lớp thực thể trong cửa sổ LINQ to SQL và trong bảng thuộc tính, nhấn chuột lên nút “...” trên các giá trị Delete/Insert/Update, và chọn SPROC mà bạn đã định nghĩa.



Có một điều hay là những thay đổi ở trên hoàn toàn được thực hiện ở lớp ánh xạ LINQ to SQL – có nghĩa là tất cả những đoạn lệnh mà tôi đã viết trước đây đều có thể tiếp tục làm việc mà không cần thay đổi bất kỳ điều gì. Điều này giúp tránh phải thay đổi lại code ngay cả nếu sau này bạn muốn dùng một hàm SPROC tối ưu hơn sau này.

Tổng kết

LINQ to SQL cung cấp một cách thức đơn giản, sáng sửa để mô hình hóa lớp dữ liệu trong ứng dụng của bạn. Một khi bạn đã định nghĩa mô hình dữ liệu, bạn có thể thực hiện các câu truy vấn, thêm, cập nhật và xóa dữ liệu một cách dễ dàng và hiệu quả.

Dùng trình thiết kế LINQ to SQL có sẵn trong Visual Studio và Visual Web Developer Express, bạn có thể tạo và quản lý mô hình dữ liệu cực kỳ nhanh. Trình LINQ to SQL designer cũng vô cùng mềm dẻo để bạn có thể tùy biến các hành vi mặc nhiên và ghi đè hoặc mở rộng hệ thống sao cho phù hợp với những yêu cầu cụ thể nào đó.

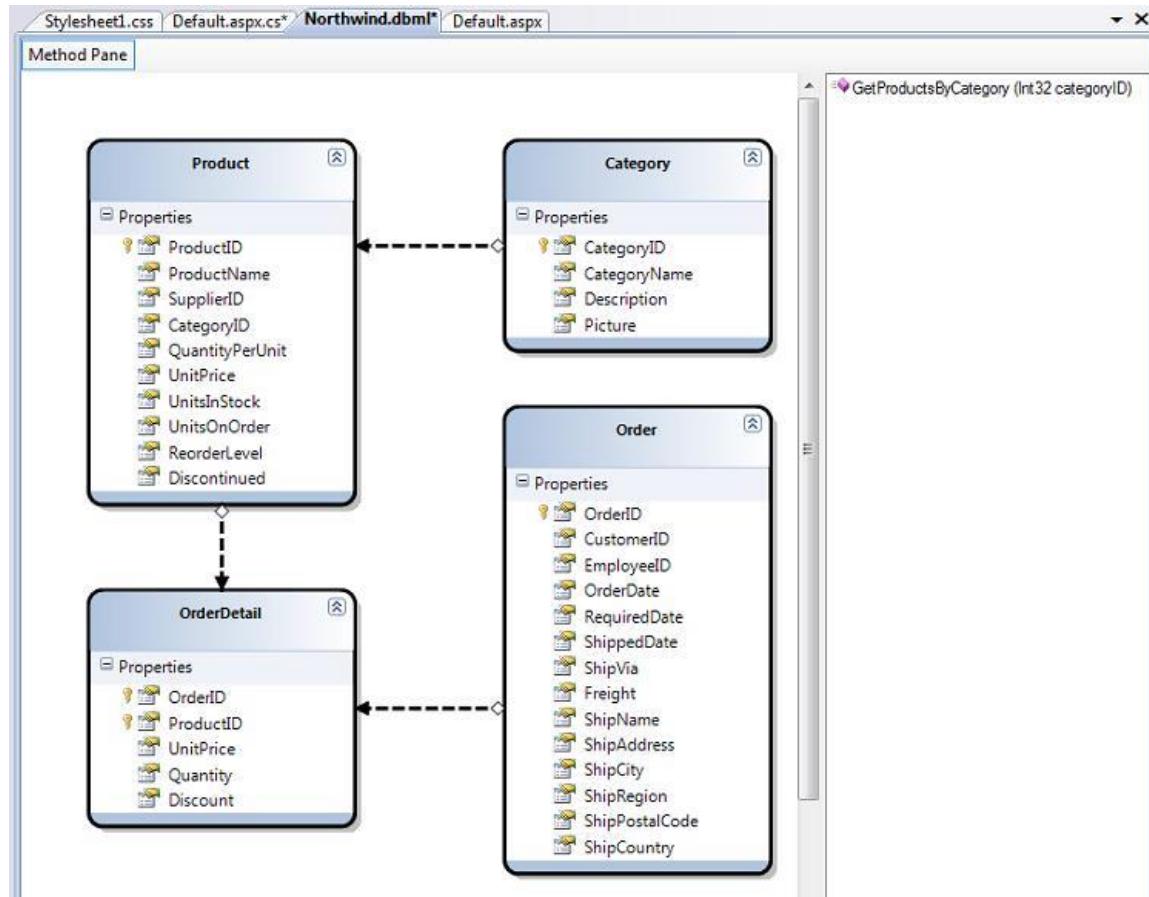
Trong những bài tiếp theo tôi sẽ dùng mô hình dữ liệu chúng ta đã tạo ra trong bài này để đào sâu hơn vào việc truy vấn, thêm, cập nhật và xóa dữ liệu. Trong các bài viết về cập nhật, thêm, xóa tôi cũng sẽ thảo luận về cách thêm các đoạn lệnh để kiểm tra dữ liệu cũng như các quy tắc vào các lớp thực thể chúng ta đã định nghĩa ở trên.

Mike Taulty cũng có một số đoạn video rất hay về LINQ to SQL mà bạn nên xem tại đây. Chúng cung cấp một cách tuyệt vời để học bằng cách xem những người khác từng bước sử dụng LINQ to SQL.

Truy vấn Cơ sở dữ liệu (LINQ to SQL phần 3)

Mô hình hóa CSDL Northwind dùng LINQ to SQL

Trong phần 2 của loạt bài này, tôi đã đi qua các bước để tạo một mô hình các lớp LINQ to SQL bằng cách dùng trình LINQ to SQL có sẵn trong VS 2008. Dưới đây là một hình mà tôi đã tạo dùng CSDL mẫu Northwind:



Lấy các sản phẩm

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 bạn làm điều này bằng cách viết các câu truy vấn dùng cú pháp LINQ với lớp NorthwindDataContext mà chúng ta đã tạo dùng trình thiết kế LINQ to SQL designer ở trên.

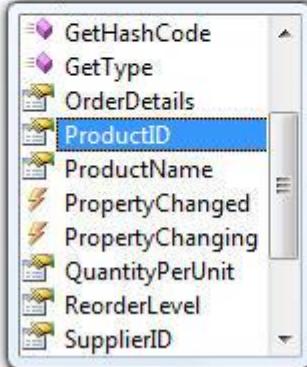
Ví dụ, để lấy và duyệt qua một tập các đối tượng Product, tôi có thể viết code như dưới đây:

```

NorthwindDataContext db = new NorthwindDataContext();

var products = from p in db.Products
               where p.CategoryID == 2
               select p;

foreach (Product product in products)
{
    product.|
```



int Product.ProductID

Trong câu truy vấn trên, tôi đã dùng một mệnh đề “where” trong cú pháp LINQ để chỉ trả về các sản phẩm trong một category cho trước. Tôi hiện đang dùng CategoryID của Product để thực hiện lọc ra các dòng mong muốn.

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

```

NorthwindDataContext db = new NorthwindDataContext();

var products = from p in db.Products
               where p.Category.CategoryName == "Beverages"
               select p;
```

Chú ý cách tôi dùng thuộc tính “Category” trên mỗi đối tượng Product để lọc theo CategoryName của Category chứa Product đó. Thuộc tính này được tự động tạo ra bởi LINQ to SQL vì chúng ta đã mô hình hóa các lớp Category và Product như một mối quan hệ một-nhiều.

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 để lấy về chỉ những Product có 5 hoặc hơn đơn đặt hàng:

```
NorthwindDataContext db = new NorthwindDataContext();  
var products = from p in db.Products  
where p.OrderDetails.Count > 5  
select p;
```

Chú ý cách chúng ta đã dùng tập hợp “OrderDetails” mà LINQ to SQL đã tạo trên mỗi lớp Product (nhờ vào mối quan hệ một-nhiều mà chúng ta đã mô hình hóa trong trình thiết kế LINQ to SQL).

Trực quan hóa các câu truy vấn LINQ to SQL trong trình gõ lỗi

Các trình ánh xạ O/R (Object relational mapper) như LINQ to SQL tạo ra và thực thi các câu lệnh SQL một cách tự động mỗi khi bạn thực hiện một câu truy vấn hay cập nhật mô hình đối tượng của nó.

Một trong những điều quan tâm lớn nhất mà các lập trình viên mới quen với ORM là: “Câu lệnh SQL thực sự được thực thi là gì?”. Một điều thực sự thú vị về LINQ to SQL là nó cho phép xem rất dễ dàng câu lệnh SQL được thực thi thực sự khi bạn chạy ứng dụng trong chế độ gõ lỗi.

Bắt đầu từ bản Beta2 của VS 2008, bạn có thể dùng một LINQ to SQL visualizer plug-in để xem một cách dễ dàng (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 biểu tượng chiếc kính lúp để xem giá trị của câu lệnh một cách trực quan:

```
NorthwindDataContext db = new NorthwindDataContext();  
var products = from p in db.Products  
select p;
```

Một cửa sổ sẽ hiện lên cho phép bạn xem một cách chính xác câu lệnh LINQ to SQL mà LINQ to SQL sẽ dùng để lấy về các đối tượng Product:

SQL Server Query Visualizer

```
Table(Product).Where(p => (p.Category.CategoryName = "Beverages"))

SELECT [t0].[ProductID], [t0].[ProductName], [t0].[SupplierID], [t0].[CategoryID], [t0].[QuantityPerUnit], [t0].[UnitPrice],
[t0].[UnitsInStock], [t0].[UnitsOnOrder], [t0].[ReorderLevel], [t0].[Discontinued]
FROM [dbo].[Products] AS [t0]
LEFT OUTER JOIN [dbo].[Categories] AS [t1] ON [t1].[CategoryID] = [t0].[CategoryID]
WHERE [t1].[CategoryName] = @p0
-----
@p0 [String]: 'Beverages'
```

Original query

Nếu bạn nhấn nút “Execute” trên cửa sổ này, nó sẽ cho phép bạn chạy câu lệnh SQL trực tiếp trong trình debugger và xem một cách chính xác dữ liệu được trả về:

QueryResult

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice
▶	1	Chai	1	1	10 boxes x 20 bags	66.0000
	2	Chang	1	1	24 - 12 oz bottles	19.0000
	24	Guaraná Fantástico	10	1	12 - 355 ml cans	4.5000
	34	Sasquatch Ale	16	1	24 - 12 oz bottles	14.0000
	35	Steeleye Stout	16	1	24 - 12 oz bottles	18.0000
	38	Côte de Blaye	18	1	12 - 75 cl bottles	263.5000
	39	Chartreuse verte	18	1	750 cc per bottle	18.0000
	43	Ipoh Coffee	20	1	16 - 500 g tins	46.0000
	67	Laughing Lumberjack	16	1	24 - 12 oz bottles	14.0000
	70	Outback Lager	7	1	24 - 355 ml bottles	15.0000
	75	Rhönbräu Kloster	12	1	24 - 0.5 l bottles	7.7500
	76	Lakkalikööri	23	1	500 ml	18.0000

Điều này rõ ràng làm cho việc xem những gì LINQ to SQL làm cho bạn trở thành cực kỳ dễ dàng. Nhờ rằng bạn có thể dễ dàng thay thế câu SQL mà LINQ to SQL thực thi nếu muốn - mặc dù trong 98% trường hợp tôi nghĩ bạn sẽ thấy rằng câu lệnh mà LINQ to SQL thực thi là thực sự, thực sự tốt.

Gắn nối các câu truy vấn LINQ to SQL vào các control LINQ to SQL

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

Lấy ví dụ, bạn có thể khai báo một control <asp:gridview> trong một trang .aspx giống như sau:

```
<h3>Products</h3>
<asp:GridView ID="GridView1"
    CssClass="gridview"
    AlternatingRowStyle-CssClass="even"
    runat="server" />
```

Tôi cũng có thể gắn nối kết quả của câu LINQ to SQL đã viết trước đây vào GridView giống như sau:

```
NorthwindDataContext db = new NorthwindDataContext();
var products = from p in db.Products
               where p.Category.CategoryName == "Beverages"
               select p;
GridView1.DataSource = products;
GridView1.DataBind();
```

Nó sẽ sinh ra một trang trông như sau:

The screenshot shows a Windows Internet Explorer window displaying a table of products. The table has four columns: ProductID, ProductName, QuantityPerUnit, and Discontinued. The Discontinued column contains checkboxes. The data is as follows:

ProductID	ProductName	QuantityPerUnit	Discontinued
1	Chai	10 boxes x 20 bags	<input type="checkbox"/>
2	Chang	24 - 12 oz bottles	<input type="checkbox"/>
24	Guaraná Fantástica	12 - 355 ml cans	<input checked="" type="checkbox"/>
34	Sasquatch Ale	24 - 12 oz bottles	<input type="checkbox"/>
35	Steeleye Stout	24 - 12 oz bottles	<input type="checkbox"/>

Data Sharpening

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 Product:

Ví dụ, câu truy vấn sau lấy về các sản phẩm:

```
NorthwindDataContext db = new NorthwindDataContext();
var products = from p in db.Products
               where p.Category.CategoryName == "Beverages"
               select p;
```

Và toàn bộ kết quả được trả về:

QueryResult

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice
▶	1	Chai	1	1	10 boxes x 20 bags	66.0000
	2	Chang	1	1	24 - 12 oz bottles	19.0000
	24	Guaraná Fantásti...	10	1	12 - 355 ml cans	4.5000
	34	Sasquatch Ale	16	1	24 - 12 oz bottles	14.0000
	35	Steeleye Stout	16	1	24 - 12 oz bottles	18.0000
	38	Côte de Blaye	18	1	12 - 75 cl bottles	263.5000
	39	Chartreuse verte	18	1	750 cc per bottle	18.0000
	43	Ipoh Coffee	20	1	16 - 500 g tins	46.0000
	67	Laughing Lumber...	16	1	24 - 12 oz bottles	14.0000
	70	Outback Lager	7	1	24 - 355 ml bottles	15.0000
	75	Rhönbräu Kloster...	12	1	24 - 0.5 l bottles	7.7500
	76	Lakkalikööri	23	1	500 ml	18.0000

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

```
NorthwindDataContext db = new NorthwindDataContext();
var products = from p in db.Products
               where p.Category.CategoryName == "Beverages"
               select new
               {
                   ID = p.ProductID,
                   Name = p.ProductName
               };
```

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

QueryResult

	ProductID	ProductName
▶	1	Chai
	2	Chang
	24	Guaraná Fantástico
	34	Sasquatch Ale
	35	Steeleye Stout
	38	Côte de Blaye
	39	Chartreuse verte
	43	Ipoh Coffee
	67	Lauding Lumber
	70	Outback Lager
	75	Rhönbräu Kloster
	76	Lakkalikööri

Một điều thực sự thú vị về LINQ to SQL là tôi 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 tôi biểu 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 Product, tổng số đơn hàng đã được đặt cho sản phẩm đó, và rồi lấy tổng giá trị của từng đơn hàng:

```
NorthwindDataContext db = new NorthwindDataContext();
var products = from p in db.Products
               where p.Category.CategoryName == "Beverages"
               select new
               {
                   ID = p.ProductID,
                   Name = p.ProductName,
                   NumOrders = p.OrderDetails.Count,
                   Revenue = p.OrderDetails.Sum(o=>o.UnitPrice * o.Quantity)
               };

```

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

SQL Server Query Visualizer

```
Table(Product).Where(p => (p.Category.CategoryName = "Beverages")).Select(p => new
<>f__AnonymousType0`4(ID = p.ProductID, Name = p.ProductName, NumOrders =
p.OrderDetails.Count, Revenue = p.OrderDetails.Sum(o => (o.UnitPrice * Convert(o.Quantity))))))
```

```
SELECT [t0].[ProductID], [t0].[ProductName], (
    SELECT COUNT(*)
    FROM [dbo].[Order Details] AS [t2]
    WHERE [t2].[ProductID] = [t0].[ProductID]
) AS [value], (
    SELECT SUM([t3].[UnitPrice] * (CONVERT(Decimal(38,9),[t3].[Quantity]))))
    FROM [dbo].[Order Details] AS [t3]
    WHERE [t3].[ProductID] = [t0].[ProductID]
) AS [value2]
FROM [dbo].[Products] AS [t0]
LEFT OUTER JOIN [dbo].[Categories] AS [t1] ON [t1].[CategoryID] = [t0].[CategoryID]
WHERE [t1].[CategoryName] = @p0
-----
@p0 [String]: 'Beverages'
```

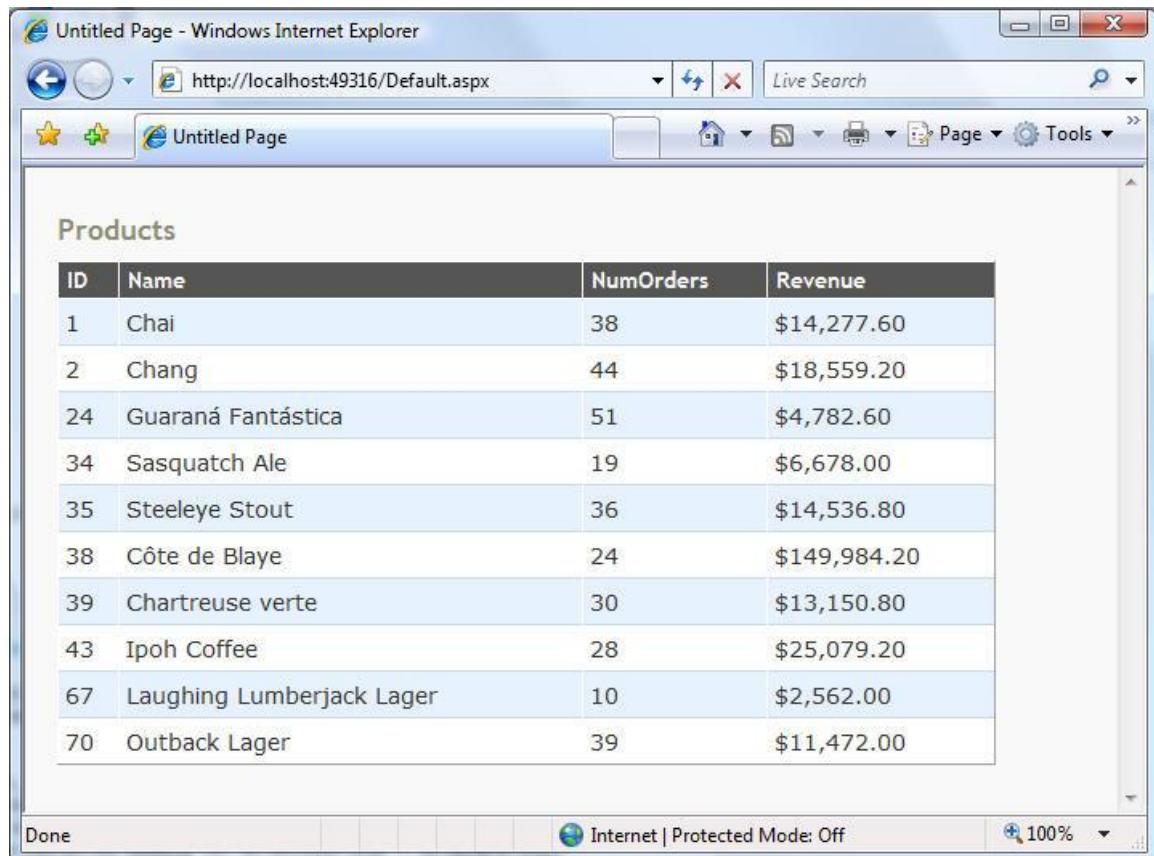
Original query **Execute**

Câu SQL ở trên cho phép tính toán tất cả các giá trị của NumOrders và Revenue từ ngay trên SQL server, và trả về chỉ những dữ liệu như dưới đây (làm cho việc thực thi được nhanh chóng):

QueryResult

	ProductID	ProductName	value	value2
▶	75	Rhönbräu Kloster...	46	8650.550000
	35	Steeleye Stout	36	14536.800000
	43	Ipoh Coffee	28	25079.200000
	38	Côte de Blaye	24	149984.200000
	67	Laughing Lumber...	10	2562.000000
	1	Chai	38	14277.600000
	24	Guaraná Fantásti...	51	4782.600000
	70	Outback Lager	39	11472.000000
	76	Lakkalikööri	39	16794.000000
	2	Chang	44	18559.200000
	39	Chartreuse verte	30	13150.800000
	34	Sasquatch Ale	19	6678.000000

Chúng ta có thể gắn nối tập kết quả vào control GridView để tạo ra một giao diện đẹp hơn:



The screenshot shows a Microsoft Internet Explorer window with the title "Untitled Page - Windows Internet Explorer". The address bar displays "http://localhost:49316/Default.aspx". The main content area contains a GridView control with the header "Products". The grid has four columns: "ID", "Name", "NumOrders", and "Revenue". The data rows are as follows:

ID	Name	NumOrders	Revenue
1	Chai	38	\$14,277.60
2	Chang	44	\$18,559.20
24	Guaraná Fantástica	51	\$4,782.60
34	Sasquatch Ale	19	\$6,678.00
35	Steeleye Stout	36	\$14,536.80
38	Côte de Blaye	24	\$149,984.20
39	Chartreuse verte	30	\$13,150.80
43	Ipoh Coffee	28	\$25,079.20
67	Laughing Lumberjack Lager	10	\$2,562.00
70	Outback Lager	39	\$11,472.00

The status bar at the bottom of the browser window shows "Done", "Internet | Protected Mode: Off", and "100%".

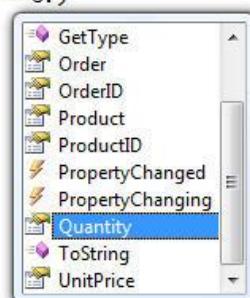
Bạn cũng có thể được hỗ trợ đầy đủ bởi tính năng intellisense bên trong VS 2008 khi viết các câu truy vấn LINQ:

```

NorthwindDataContext db = new NorthwindDataContext();

var products = from p in db.Products
               where p.Category.CategoryName == "Beverages"
               select new
               {
                   ID = p.ProductID,
                   Name = p.ProductName,
                   NumOrders = p.OrderDetails.Count,
                   Revenue = p.OrderDetails.Sum(o => o.UnitPrice * o.Quantity)
               };

```



Trong ví dụ trên, tôi đang sử dụng một kiểu vô danh (anonymous type) và dùng object initialization để gọt giũa và định nghĩa cấu trúc trả về. Một điều thực sự tuyệt vời là VS 2008 cung cấp intellisense đầy đủ, kiểm tra lúc dịch và cả refactoring khi làm việc cả với các tập kết quả có kiểu vô danh:

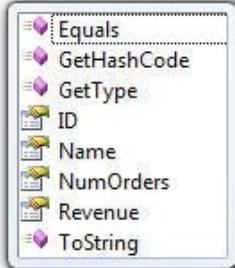
```

NorthwindDataContext db = new NorthwindDataContext();

var products = from p in db.Products
               where p.Category.CategoryName == "Beverages"
               select new
               {
                   ID = p.ProductID,
                   Name = p.ProductName,
                   NumOrders = p.OrderDetails.Count,
                   Revenue = p.OrderDetails.Sum(o=>o.UnitPrice * o.Quantity)
               };

foreach(var product in products)
{
    product.
}

```



Phân trang kết quả truy vấn

Một trong những yêu cầu chung khi viết các trang web là bạn phải có khả năng phân trang một cách hiệu quả. LINQ cung cấp sẵn hai hàm mở rộng cho phép bạn có thể làm điều đó một cách dễ dàng và hiệu quả – hàm Skip() và Take().

Bạn có thể dùng Skip() và Take() như dưới đây để chỉ ra rằng bạn chỉ muốn lấy về 10 đối tượng sản phẩm – bắt đầu từ một sản phẩm cho trước mà chúng ta chỉ ra trong tham số truyền vào:

```
void BindProducts(int startRow)
{
    NorthwindDataContext db = new NorthwindDataContext();
    var products = from p in db.Products
                   where p.OrderDetails.Count > 2
                   select new
                   {
                       ID = p.ProductID,
                       Name = p.ProductName,
                       NumOrders = p.OrderDetails.Count,
                       Revenue = p.OrderDetails.Sum(o => o.UnitPrice * o.Quantity)
                   };
    GridView1.DataSource = products.Skip(startRow).Take(10);
    GridView1.DataBind();
}
```

Chú ý ở trên tôi đã không dùng Skip() và Take() trong câu khai báo truy vấn các sản phẩm – mà chỉ dùng tới khi gắn kết dữ liệu vào GridView. Mọi người hay hỏi “Có phải làm như vậy thì câu lệnh đầu tiên sẽ lấy toàn bộ dữ liệu từ CSDL về lớp giữa, rồi sau đó mới thực hiện việc phân trang ?”. Câu trả lời là “Không”. Lý do là vì LINQ chỉ thực sự thực thi các câu truy vấn khi bạn lấy kết quả từ nó mà thôi.

Một trong những ưu điểm của mô hình này là nó cho phép bạn 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 bạn tạo ra các câu truy vấn từ các câu khác, giúp bạn có thể xây dựng các câu truy vấn rất phức tạp cũng như có thể dùng lại được các câu truy vấn khác.

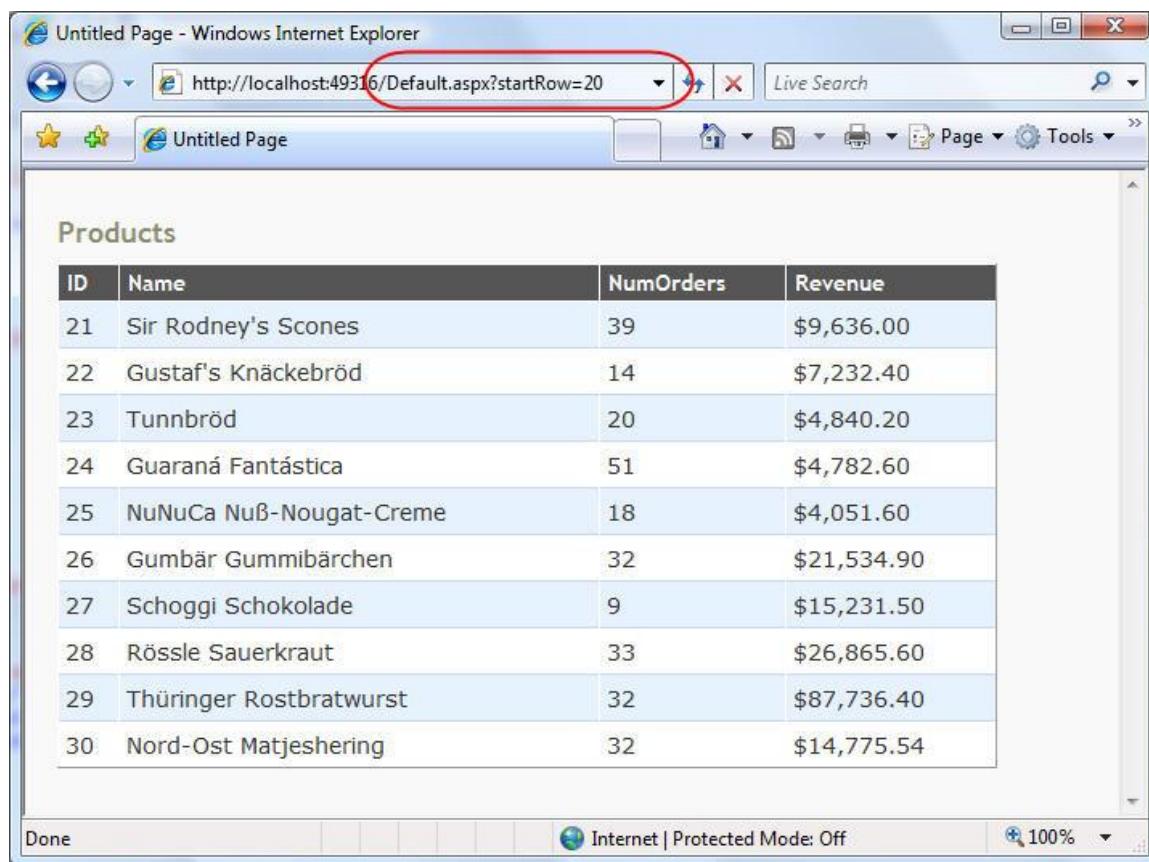
Một khi tôi đã có phương thức BindProduct() định nghĩa ở trên, tôi có thể viết lệnh như dưới đây để lấy về chỉ số đầu từ query string, và cho phép danh sách sản phẩm có thể được hiện phân trang và hiển thị:

```

void Page_Load(object sender, EventArgs e)
{
    int startRow = Convert.ToInt32(Request.QueryString["startRow"]);
    BindProducts(startRow);
}

```

Nó sẽ cho chúng ta một trang hiển thị các sản phẩm có nhiều hơn 5 đơn đặt hàng, cùng với doanh thu tương ứng, và được phân trang dựa trên tham số truyền vào qua query string:



The screenshot shows a Windows Internet Explorer window titled "Untitled Page - Windows Internet Explorer". The address bar contains the URL "http://localhost:49316/Default.aspx?startRow=20". The main content area displays a table titled "Products" with 10 rows of data. The columns are labeled "ID", "Name", "NumOrders", and "Revenue". The data is as follows:

ID	Name	NumOrders	Revenue
21	Sir Rodney's Scones	39	\$9,636.00
22	Gustaf's Knäckebröd	14	\$7,232.40
23	Tunnbröd	20	\$4,840.20
24	Guaraná Fantástica	51	\$4,782.60
25	NuNuCa Nuß-Nougat-Creme	18	\$4,051.60
26	Gumbär Gummibärchen	32	\$21,534.90
27	Schoggi Schokolade	9	\$15,231.50
28	Rössle Sauerkraut	33	\$26,865.60
29	Thüringer Rostbratwurst	32	\$87,736.40
30	Nord-Ost Matjeshering	32	\$14,775.54

Ghi chú: Khi làm việc với SQL 2005, LINQ to SQL sẽ dùng hàm ROW_NUMBER() để thực hiện việc phân trang logic trong CSDL. Nó đảm bảo rằng chỉ 10 dòng dữ liệu được trả về khi chúng ta thực hiện các câu lệnh trên:

QueryResult

	ID	Name	NumOrders	Revenue
▶	21	Sir Rodney's Sco...	39	9636.000000
	22	Gustaf's Knäcke...	14	7232.400000
	23	Tunnbröd	20	4840.200000
	24	Guaraná Fantásti...	51	4782.600000
	25	NuNuCa Nuß-No...	18	4051.600000
	26	Gumbär Gummib...	32	21534.900000
	27	Schoggi Schokol...	9	15231.500000
	28	Rössle Sauerkraut	33	26865.600000
	29	Thüringer Rostbr...	32	87736.400000
	30	Nord-Ost Matjesh...	32	14775.540000

Nó làm cho việc phân trang hiệu quả và dễ dàng hơn, đặc biệt là với các tập dữ liệu lớn.

Tổng kết

Hi vọng các bước trên đã cung cấp một cái nhìn đầy đủ về những đặc tính mà LINQ to SQL cung cấp, để tìm hiểu thêm về các biểu thức LINQ và cú pháp mới được dùng trong C# và VB.NET trong VS 2008, xin hãy tham khảo thêm các bài viết sau:

[Automatic Properties, Object Initializer and Collection Initializers](#)

[Extension Methods](#)

[Lambda Expressions](#)

[Query Syntax](#)

[Anonymous Types](#)

Trong bài viết tiếp theo trong loạt bài này, tôi sẽ cho thấy cách thêm các phép kiểm tra vào mô hình dữ liệu của chúng ta, và biểu diễn cách chúng ta có thể dùng để đưa logic chương trình vào mỗi lần thực thi các câu lệnh update, insert, hay delete dữ liệu. Tôi cũng sẽ cho các bạn thấy các tính năng cao cấp hơn của lazy loading và eager loading, cách dùng control mới `<asp:LINQDataSource>` để hỗ trợ việc khai báo databinding trong ASP.NET, cách giải quyết xung đột...

<http://my.opera.com/hivietnam/blog/show.dml/4395413>

Cập nhật cơ sở dữ liệu (LINQ to SQL phần 4)

Từ vài tuần trước, tôi đã bắt đầu một loạt bài nói về LINQ to SQL. LINQ to SQL là một O/RM có sẵn trong bản .NET Framework 3.5, và nó cho phép bạn dễ dàng mô hình hóa các CSDL cùng các lớp .NET. Bạn có thể dùng các biểu thức LINQ để truy vấn CSDL, cũng như để thêm/xóa/sửa dữ liệu.

Dưới đây là 3 bài đầu tiên trong loạt bài này:

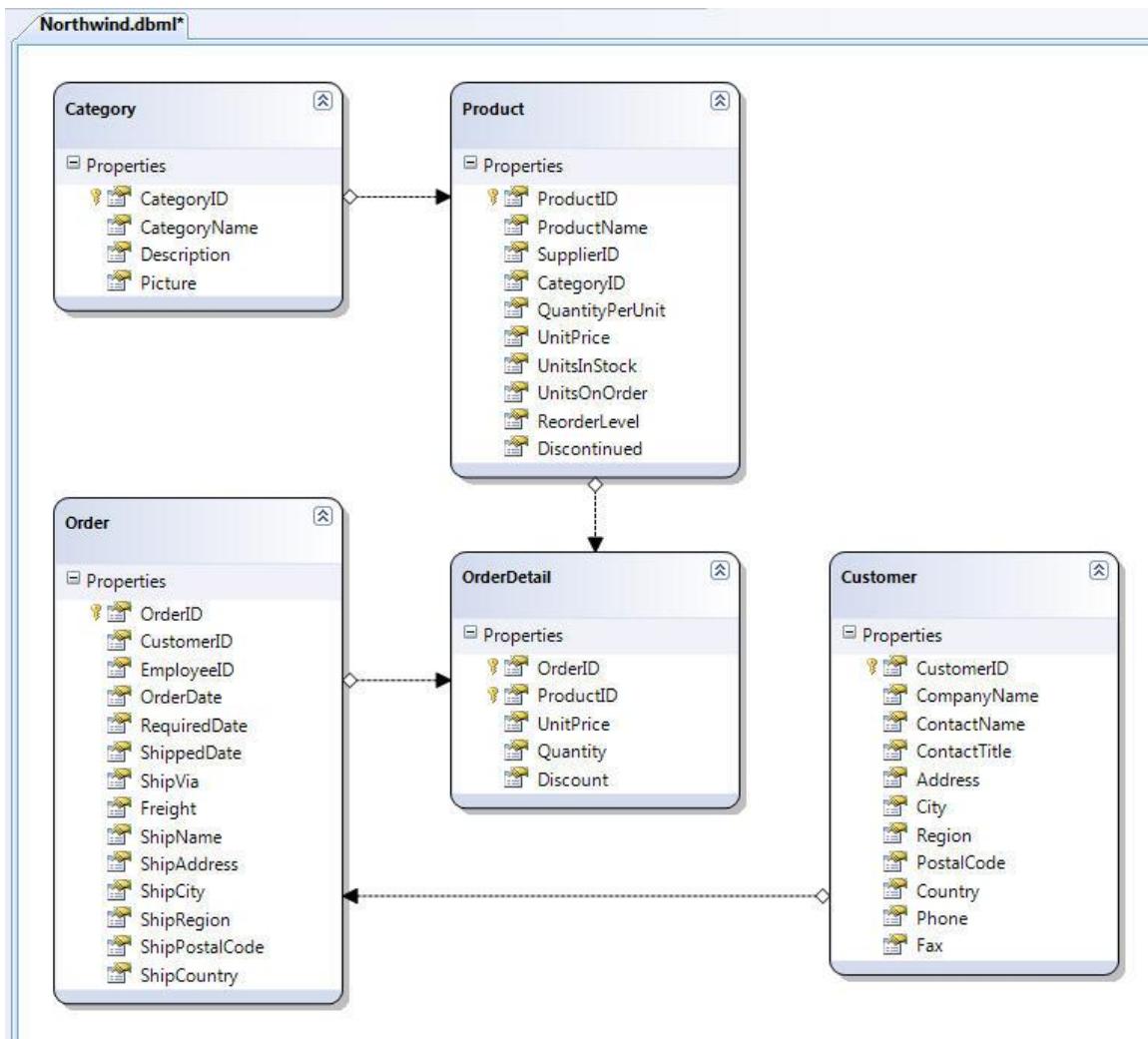
- Sử dụng LINQ to SQL (phần 1)
- Định nghĩa các lớp mô hình dữ liệu (phần 2)
- Truy vấn Cơ sở dữ liệu (phần 3)

Trong bài hôm nay, tôi sẽ nói rõ hơn về cách chúng ta dùng CSDL đã được mô hình hóa trước đây, và dùng nó để cập nhật, chỉnh sửa và xóa dữ liệu.

Tôi cũng sẽ cho các bạn thấy các chúng ta có thể thêm các quy tắc (business rule – sau này trở đi tôi sẽ để nguyên từ business rule, vì từ này rõ nghĩa hơn) và tùy biến cách xác thực tính hợp lệ của dữ liệu.

CSDL Northwind được mô hình hóa dùng LINQ to SQL

Trong phần 2 của loạt bài này, tôi đã đi qua các bước để tạo nên mô hình các lớp LINQ to SQL dùng LINQ to SQL designer có trong VS 2008. Dưới đây là sơ đồ lớp đã được tạo cho CSDL mẫu Northwind và cũng sẽ là mô hình được dùng trong bài viết này:



ra 5 lớp mô hình: Product, Category, Customer, Order and OrderDetail. Các thuộc tính của mỗi lớp ánh xạ vào các cột tương ứng trong bảng dữ liệu. Mỗi đối tượng thuộc lớp thực thể sẽ biểu diễn một dòng trong bảng CSDL.

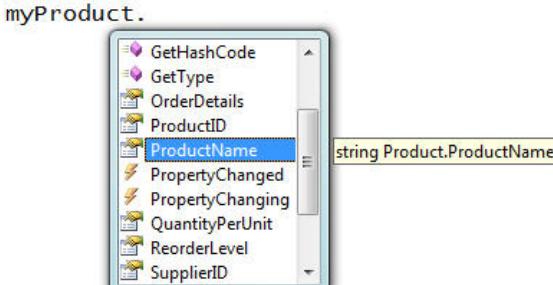
Khi định nghĩa mô hình dữ liệu, LINQ to SQL designer cũng tạo ra một lớp `DataContext` cung cấp các cách thức để truy vấn và cập nhật lại dữ liệu.

Trong mô hình mẫu chúng ta đã định nghĩa ở trên, lớp này được đặt tên là “`NorthwindDataContext`”. Lớp `NorthwindDataContext` có các thuộc tính biểu diễn các bảng chúng ta đã định nghĩa trong CSDL (`Products`, `Categories`, `Customers`, `Orders`, `OrderDetails`).

Như chúng ta đã xem trong phần 3, chúng ta cũng dễ dàng dùng các biểu thức LINQ để truy vấn và lấy dữ liệu từ CSDL bằng cách dùng lớp `NorthwindDataContext`. 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 Product đơn bằng cách tìm dựa trên tên sản phẩm:

```
NorthwindDataContext northwind = new NorthwindDataContext();
Product myProduct = northwind.Products.Single(p => p.ProductName == "Chai");
```



Tôi cũng có thể viết thêm một câu truy vấn LINQ dưới đây để lấy về tất cả các sản phẩm từ CSDL mà hiện tại chưa có đơn đặt hàng, và giá tiền nhiều hơn \$100:

```
NorthwindDataContext northwind = new NorthwindDataContext();
var products = from p in northwind.Products
               where p.OrderDetails.Count == 0 && p.UnitPrice > 100
               select p;
```

Chú ý cách tôi đang dùng “OrderDetails” kết hợp với mỗi sản phẩm như một phần của câu truy vấn để chỉ lấy về các sản phẩm không có đơn đặt hàng.

Change Tracking và DataContext.SubmitChanges()

When we perform queries and retrieve objects like the product instances above, LINQ to SQL will by default keep track of any changes or updates we later make to these objects. We can make any number of queries and changes we want using a LINQ to SQL DataContext, and these changes will all be tracked together.

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 lại vết của 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 mà chúng ta 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 gọi, và không liên quan gì đến CSDL. Có nghĩa là bạn không hề dùng tài nguyên trên CSDL, hoặc bạn không cần cài đặt thêm hay thay đổi bất kỳ thứ gì trên CSDL để 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 đổi 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.

Lấy ví dụ, bạn có thể viết câu lệnh dưới đây để cập nhật lại giá tiền và số lượng đơn vị còn lại của sản phẩm “Chai”:

```
NorthwindDataContext northwind = new NorthwindDataContext();
Product myProduct = northwind.Products.Single(p => p.ProductName == "Chai");
myProduct.UnitPrice = 2;
myProduct.UnitsInStock = 4;
northwind.SubmitChanges();
```

Khi tôi gọi northwind.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 sản phẩm mà chúng ta đã sửa lại như ở trên.

Tôi có thể viết đoạn lệnh dưới đây để duyệt qua danh sách các sản phẩm ít phổ biến và giá cao, sau đó đặt lại thuộc tính “ReorderLevel” = 0:

```
NorthwindDataContext northwind = new NorthwindDataContext();
var expensiveUnpopularProducts = from p in northwind.Products
                                   where p.OrderDetails.Count == 0 && p.UnitPrice > 100
                                   select p;
foreach (Product product in expensiveUnpopularProducts) {
    product.ReorderLevel = 0;
}
northwind.SubmitChanges();
```

Khi tôi gọi `northwind.SubmitChanges()` như trên, LINQ to SQL sẽ tính toán và thực thi một tập thích hợp các phát biểu UPDATE để cập nhật các sản phẩm có thuộc tính ReorderLevel đã bị thay đổi.

Hãy nhớ là nếu giá trị của các thuộc tính của đối tượng Product không bị thay đổi 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 đơn giá của đối tượng “Chai” đã là 2 và số sản phẩm còn lại là 4, thì việc gọi `SubmitChanges()` sẽ chẳng làm thực thi bất kỳ câu SQL nào. Cũng vậy, chỉ các sản phẩm trong ví dụ thứ hai có `ReorderLevel` không bằng 0 mới được cập nhật khi gọi `SubmitChanges()`.

Các ví dụ Insert và Delete

Ngoài việc cập nhật các dòng đã có trong CSDL, LINQ to SQL còn cho phép bạn thêm và xóa dữ liệu. Bạn 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.

Thêm một sản phẩm

Bạn có thể thêm một sản phẩm mới vào CSDL bằng việc tạo ra một đối tượng thuộc lớp “`Product`”, gán các giá trị thuộc tính, và sau đó thêm nó vào tập hợp “`Products`” của `DataContext`:

```
NorthwindDataContext northwind = new NorthwindDataContext();
Product myProduct = new Product();
myProduct.ProductName = "Scott's Special Product";
myProduct.UnitPrice = 999;
myProduct.UnitsInStock = 1;
myProduct.CategoryID = 1;
northwind.Products.Add(myProduct);
northwind.SubmitChanges();
```

Khi gọi “`SubmitChanges`” như trên, một dòng mới sẽ được thêm vào bảng `Product`.

Xóa các sản phẩm

Cũng như tôi đã nói về việc thêm một sản phẩm mới bằng cách đối tượng Product vào tập hợp Products của DataContext, tôi cũng có thể làm một cách ngược lại khi muốn xóa một sản phẩm từ CSDL bằng cách xóa nó khỏi tập hợp này:

```
NorthwindDataContext northwind = new NorthwindDataContext();
var lameProducts = from p in northwind.Products
                   where p.OrderDetails.Count > 0 && p.Discontinued == true
                   select p;
northwind.Products.RemoveAll(lameProducts);
northwind.SubmitChanges();
```

(RemoveAll đã được thay đổi bằng DeleteOnSubmit trong phiên bản hiện tại)

Chú ý cách tôi lấy một tập hợp các sản phẩm không còn được sản xuất và cũng không có đơn đặt hàng nào bằng cách dùng một câu truy vấn LINQ, rồi sau đó truyền nó cho phương thức RemoveAll của tập hợp Products trong DataContext. Khi gọi SubmitChanges(), tất cả các sản phẩm đó sẽ bị xóa khỏi CSDL.

Cập nhật thông qua các quan hệ

Điều làm cho các trình ORM như LINQ to SQL cực kỳ mềm dẻ là nó cho phép chúng ta dễ dàng mô hình hóa mối quan hệ giữa các bảng trong mô hình dữ liệu. Ví dụ, tôi có thể mô hình hóa mỗi Product trong một Category, mỗi Order để chứa các OrderDetails cho từng mục, kết hợp các OrderDetail với một Product, và làm cho mỗi Customer kết hợp với một tập các Order. Tôi đã biểu diễn cách xây dựng và mô hình hóa các mối quan hệ trong phần 2 của loạt bài này.

LINQ to SQL cho phép tôi tận dụng được ưu điểm của các mối quan hệ trong việc truy vấn và cập nhật dữ liệu. Ví dụ, tôi có thể viết đoạn lệnh dưới đây để tạo một Product mới và kết hợp nó với một category “Beverages” trong CSDL như dưới đây:

```

NorthwindDataContext northwind = new NorthwindDataContext();

// Retrieve beverages category
Category beverages = northwind.Categories.Single(c => c.CategoryName == "Beverages");

// Create new Product
Product myProduct = new Product();
myProduct.ProductName = "Scott's Special Product";
myProduct.UnitPrice = 999;
myProduct.UnitsInStock = 1;

// Associate product with beverage category
beverages.Products.Add(myProduct); ←

// Update database
northwind.SubmitChanges();

```

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

Hãy chú ý cách tôi thêm một đối tượng Product vào tập hợp Products của một Category. Nó sẽ chỉ ra rằng 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 tôi gọi SubmitChanges.

Một ví dụ khác cho thấy LINQ to SQL có thể giúp quản lý quan hệ giữa các bảng như thế nào và giúp cho việc lập trình sáng sủa hơn, hãy xem một ví dụ dưới đây khi tôi tạo một Order mới cho một khách hàng đã có. Sau khi đặt giá trị cho ngày chuyển hàng và chi phí cho việc đặt hàng, tôi sẽ tạo tiếp 2 mục chi tiết trong đơn đặt hàng để chỉ đến các sản phẩm mà khách hàng đang muốn mua. Sau đó, tôi sẽ kết hợp đơn đặt hàng với khách hàng, và cập nhật các thay đổi vào CSDL.

```

NorthwindDataContext northwind = new NorthwindDataContext();

// Retrieve product details
Product chai = northwind.Products.Single(p => p.ProductName == "Chai");
Product tofu = northwind.Products.Single(p => p.ProductName == "Tofu");

// Create new Order and Order Line items for products
Order myOrder = new Order();
myOrder.OrderDate = DateTime.Now;
myOrder.RequiredDate = DateTime.Now.AddDays(1);
myOrder.Freight = 34;

OrderDetail myItem1 = new OrderDetail();
myItem1.Product = chai;
myItem1.Quantity = 23;

OrderDetail myItem2 = new OrderDetail();
myItem2.Product = tofu;
myItem2.Quantity = 3;

// Associate new order and order line items together
myOrder.OrderDetails.Add(myItem1);
myOrder.OrderDetails.Add(myItem2);

// Retrieve customer details
Customer myCustomer = northwind.Customers.Single(c => c.CompanyName == "B's Beverages");

// Add the order to the customer's Orders collection
myCustomer.Orders.Add(myOrder);

// Update database
northwind.SubmitChanges();

```

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

Như bạn thấy, mô hình lập trình trên cho phép thực hiện tất cả các công việc này một cách cực kỳ sáng sủa theo phong cách hướng đối tượng.

Transactions

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

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

Nếu không có một transaction đang diễn ra, DataContext của LINQ to SQL sẽ tự động bắt đầu một transaction để bảo vệ các thao tác cập nhật khi gọi

SubmitChanges(). Thêm vào đó, LINQ to SQL còn cho phép bạn tự định nghĩa và dùng đối tượng TransactionScope của riêng bạn. Điều này làm cho việc tích hợp các lệnh LINQ to SQL vào các đoạn mã truy cập dữ liệu đã có dễ dàng hơn. Nó cũng có nghĩa là bạn có thể đưa cả các tài nguyên không phải của CSDL vào trong cùng transaction. Ví dụ: bạn có thể gửi đi một thông điệp MSMQ, cập nhật hệ thống file (sử dụng khả năng hỗ trợ transaction cho hệ thống file),... và nhóm tất cả các thao tác đó vào trong cùng một transaction mà bạn dùng để cập nhật CSDL dùng LINQ to SQL.

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 dữ 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 hỗ trợ nhiều cách để các nhà phát triển có thể dễ dàng tích hợp chúng vào với các mô hình dữ liệu của họ.

LINQ to SQL cho phép bạn thêm khả năng xác thực dữ liệu mà không phụ thuộc vào cách bạn tạo ra mô hình dữ liệu cũng như nguồn dữ liệu. Điều này cho phép bạn có thể lặp lại các phép kiểm tra ở nhiều chỗ khác nhau, và làm cho mã lệnh sáng sủa và dễ bảo trì hơn rất nhiều.

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

Khi định nghĩa các lớp mô hình dữ liệu dùng LINQ to SQL designer trong VS 2008, 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 sẽ khớp với các kiểu dữ liệu tương ứng trong CSDL. Điều này có nghĩa là bạn sẽ gặp lỗi biên dịch nếu cố gắng gán một giá trị kiểu boolean và 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 LINQ to SQL designer cũng cho phép NULL. Các cột không cho phép NULL sẽ tự động đưa ra các exception nếu bạn cố gắng lưu một đối tượng có thuộc tính đó mang giá trị NULL. LINQ to SQL sẽ đảm bảo các cột định danh/đuyn nhất không bị trùng lặp trong CSDL.

Bạn có thể dùng LINQ to SQL designer để 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 bạn không cần làm bất kỳ điều gì để cho phép chúng. LINQ to SQL cũng tự động xử lý các chuỗi escape, do vậy bạn không cần lo lắng về lỗi SQL injection.

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 bạn sẽ gặp phải những yêu cầu kiểm tra phức tạp hơn nhiều.

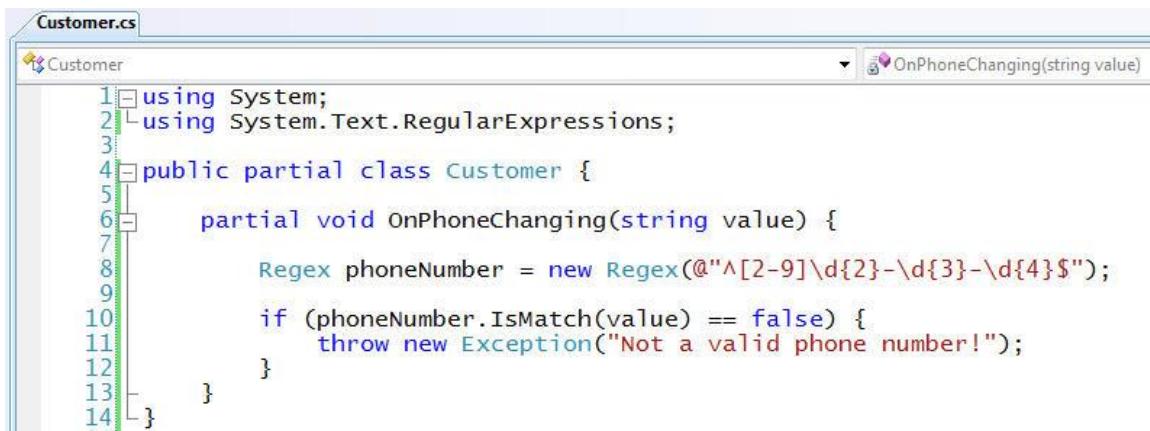
Hãy xem một ví dụ trong CSDL Northwind, khi tôi định nghĩa thuộc tính Phone thuộc lớp Customer có kiểu dữ liệu là nvarchar. Các nhà phát triển dùng LINQ to SQL có thể viết code giống như dưới đây để cập nhật nó với một số phone hợp lệ:

```
NorthwindDataContext northwind = new NorthwindDataContext();
Customer myCustomer = northwind.Customers.Single(c => c.CompanyName == "B's Beverages");
myCustomer.Phone = "425-703-8072";
northwind.SubmitChanges();
```

Vẫn đe là đoạn code trên được coi là hợp lệ đúng từ góc độ kiểu dữ liệu SQL, vì chuỗi trên vẫn là một chuỗi nvarchar mặc dù có thể nó không phải là một số phone hợp lệ:

```
NorthwindDataContext northwind = new NorthwindDataContext();
Customer myCustomer = northwind.Customers.Single(c => c.CompanyName == "B's Beverages");
myCustomer.Phone = "absdfdsfdsfsd";
northwind.SubmitChanges();
```

Để tránh việc thêm các số phone kiểu như trên vào CSDL, chúng ta có thể thêm một quy tắc kiểm tra tính hợp lệ vào lớp Customer. Thêm một quy tắc để kiểm tra thực sự đơn giản. Tất cả những gì chúng ta cần làm là thêm một partial class vào và định nghĩa phương thức như dưới đây:



```
Customer.cs
Customer
1 using System;
2 using System.Text.RegularExpressions;
3
4 public partial class Customer {
5
6     partial void OnPhoneChanging(string value) {
7
8         Regex phoneNumber = new Regex(@"^([2-9]\d{2}-\d{3}-)\d{4}$");
9
10        if (phoneNumber.IsMatch(value) == false) {
11            throw new Exception("Not a valid phone number!");
12        }
13    }
14}
```

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. Bạn 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 bạn có thể 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# có trong VS 2008 beta 2. Wes Dyer trong nhóm C# có một bài nói về cách các partial method làm việc tại đây.

Trong ví dụ về việc kiểm tra tính hợp lệ dữ liệu ở trên, tôi dùng phương thức OnPhoneChanging, đây là một phương thức sẽ được thực thi bất kỳ lúc nào người dùng gán lại giá trị cho thuộc tính Phone trên một đối tượng Customer. Tôi có thể dùng phương thức này để xác thực giá trị đầu vào theo bất kỳ cách gì tôi muốn (trong ví dụ này, tôi dùng một biểu thức chính quy). Nếu giá trị đã hợp lệ, tôi 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 tôi có thể phát ra một Exception bên trong phương thức kiểm tra, và phép gán khi đó sẽ không được thực hiện.

Hỗ trợ tùy biến việc 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 bạn muốn kiểm tra giá trị của từng thuộc tính riêng lẻ. Nhưng đôi khi, bạn sẽ cần phải kiểm tra dựa trên nhiều giá trị của các thuộc tính khác nhau.

Hãy xem ví dụ sau, tôi sẽ đặt giá trị cho 2 thuộc tính “OrderDate” và “RequiredDate”:

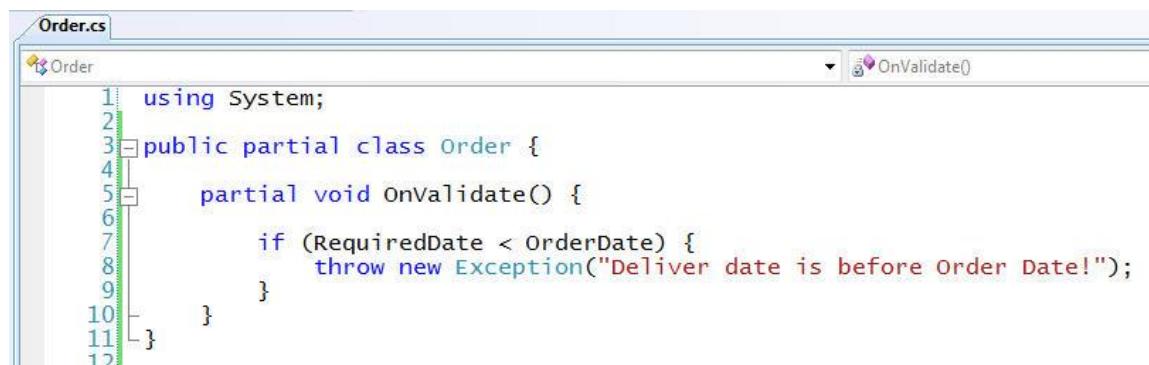
```
NorthwindDataContext northwind = new NorthwindDataContext();
Order myOrder = new Order();
myOrder.OrderDate = DateTime.Now;
myOrder.RequiredDate = DateTime.Now.AddDays(-1);

northwind.Orders.Add(myOrder);
northwind.SubmitChanges();
```

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

Đoạn lệnh trên là hợp lệ nếu chỉ đơn thuần xét từ góc độ ngôn ngữ – nhưng sẽ là không có ý nghĩa khi bạn lại muốn đặt ngày khách hàng yêu cầu trước ngày đặt hàng.

Tin vui là từ bản LINQ to SQL beta 2, chúng ta có thể thêm vào các quy tắc kiểm tra cho từng thực thể để tránh các lỗi kiểu như trên bằng cách thêm một lớp partial cho lớp “Order” 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.



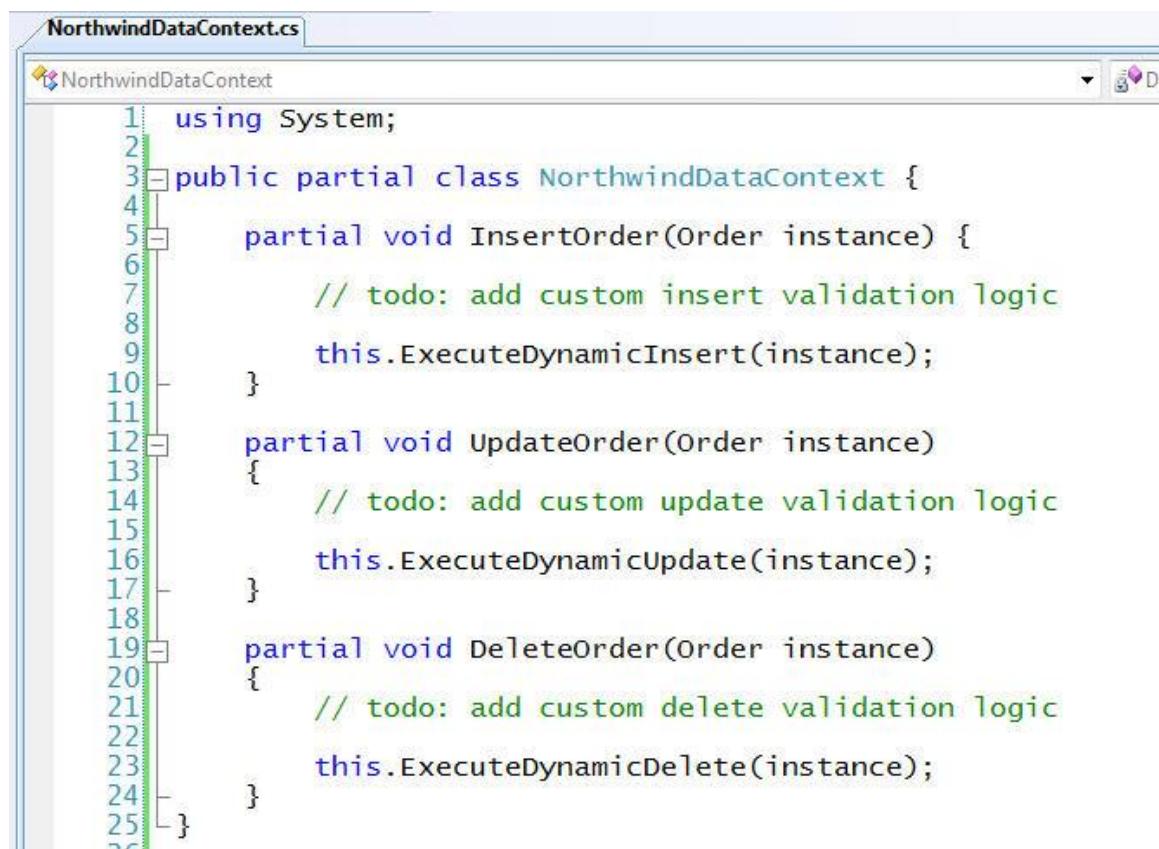
cập (chỉ đọc) vào các đối tượng liên quan, và có thể phát ra một exception nếu có tồn tại các giá trị không hợp lệ. Bất kỳ một exception nào được phát

ra từ phương thức OnValidate() sẽ làm cho việc cập nhật bị hủy bỏ, và hủy bỏ các thay đổi trong transaction.

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

Có nhiều lúc bạn 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 bạn 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 thao tác này sẽ được thực thi tự động khi bạn gọi SubmitChanges() trên lớp DataContext.

Bạn 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” của DataContext).



```
NorthwindDataContext.cs
NorthwindDataContext
1 using System;
2
3 public partial class NorthwindDataContext {
4
5     partial void InsertOrder(Order instance) {
6
7         // todo: add custom insert validation logic
8
9         this.ExecuteDynamicInsert(instance);
10    }
11
12    partial void UpdateOrder(Order instance)
13    {
14        // todo: add custom update validation logic
15
16        this.ExecuteDynamicUpdate(instance);
17    }
18
19    partial void DeleteOrder(Order instance)
20    {
21        // todo: add custom delete validation logic
22
23        this.ExecuteDynamicDelete(instance);
24    }
25}
```

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 tôi muốn tạo một Order mới và kết hợp nó với một Customer đã có:

```
NorthwindDataContext northwind = new NorthwindDataContext();
// Retrieve customer
Customer myCustomer = northwind.Customers.Single(c => c.CompanyName == "Microsoft");
// Create new Order
Order myOrder = new Order();
myOrder.OrderDate = DateTime.Now;
myOrder.ShipAddress = "One Microsoft Way";
myOrder.ShipPostalCode = "98052";
// Associate order with customer
myCustomer.Orders.Add(myOrder);
// Update changes
northwind.SubmitChanges();
```

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

Khi tôi gọi northwind.SubmitChanges() ở trên, LINQ to SQL sẽ xác định là nó cần lưu lại một đối tượng Order, và phương thức InsertOrder sẽ tự động được gọi.

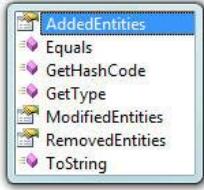
Nâng cao: Xem danh sách thay đổi cho Transaction

Đôi khi bạn muốn thêm các quy tắc kiểm tra mà không thể chỉ dựa trên từng thao tác thêm/xóa/sửa riêng lẻ, thay vào đó bạn 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 bạn 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 bạn có thể tạo một lớp thừa kế từ lớp DataContext và override phương thức SubmitChanges(). Khi đó bạn có thể lấy ChangeList() cho thao tác cập nhật và thực hiện các phép kiểm tra cần thiết trước khi thực thi:

```

public class MyNorthwindDataContext : NorthwindDataContext
{
    public override void SubmitChanges(System.Data.Linq.ConflictMode failureMode)
    {
        ChangeSet changes = this.GetChangeSet();
        changes.

        System.Collections.Generic.IList<object> ChangeSet.AddedEntities
        base.SubmitChanges(failureMode);
    }
}

```

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 là làm thế nào có thể 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 product bên trong một ứng dụng, và một người đặt lại giá trị cho ReorderLevel 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 triển 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 đổi bởi người đầu tiên sẽ bị thay đổi mà không biết. Và điều này thường được coi là một trải nghiệm kém cỏi (và không đúng) – có nghĩa người dùng sẽ 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 người phát triển có thể chọn giải pháp xử lý hoặc có thể yêu cầu người dùng chọn một thao tác nào họ muốn.

Tôi sẽ nói về cách dùng optimistic concurrency với LINQ to SQL trong các bài viết khác.

Dùng SPROCs hoặc tùy biến logic các câu SQL:

Một trong những câu hỏi mà các nhà phát triển (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 kiểm soát hoàn toàn các câu lệnh SQL được thực thi bên dưới ?”

Một tin tốt là LINQ to SQL có một mô hình cực kỳ mềm dẻo, nó cho phép các nhà phát triển có thể 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 bạn có thể bắt đầu bằng cách định nghĩa mô hình dữ liệu của riêng bạn và để LINQ to SQL tự thực hiện các thao tác thêm/sửa/xóa. Rồi sau đó bạn 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 bạn mà không phải thay đổi bất kỳ đoạn lệnh nào dùng mô hình dữ liệu đó, và cũng chẳng phải thay đổi bất kỳ quy tắc kiểm tra đã tạo trước đó. Điều này cung cấp khả năng tùy biến rất lớn cho bạn khi xây dựng ứng dụng.

Tôi cũng sẽ nói kỹ hơn về cách tùy biến mô hình dữ liệu dùng các thủ tục hay câu lệnh SQL trong một bài viết khác.

Sử dụng asp:LinqDataSource (LINQ to SQL phần 5)

Trong bài viết này, tôi sẽ khám phá control mới <asp:linqdatasource> có trong ASP.NET thuộc phiên bản .NET 3.5. Control này là một datasource control mới cho ASP.NET (giống ObjectDataSource và SQLDataSource có trong ASP.NET 2.0) cho phép bạn khai báo việc gắn kết dữ liệu vào mô hình dữ liệu của LINQ to SQL cực kỳ dễ dàng.</asp:linqdatasource>

Ứng dụng mẫu mà chúng ta sẽ xây dựng:

Chương trình web chỉnh sửa dữ liệu đơn giản mà tôi sẽ xây dựng qua các bước được mô tả trong bài này sẽ là một chương trình cho phép nhập/chỉnh sửa dữ liệu cho các sản phẩm trong một CSDL:

Products			
	ProductName	Supplier	UnitPrice
Edit Delete	Chai	Exotic Liquids	\$100.00
Update Cancel	Guaraná Fantástica	Refrescos Americanas LTDA	5.5000
Edit Delete	Sasquatch Ale	Exotic Liquids	\$15.00
Edit Delete	Steeleye Stout	New Orleans Cajun Delights	\$13.00
Edit Delete	Côte de Blaye	Grandma Kelly's Homestead	\$263.50
Edit Delete	Chartreuse verte	Tokyo Traders	
Edit Delete	Ipoh Coffee	Cooperativa de Quesos 'Las Cabras'	
Edit Delete	Laughing Lumberjack Lager	Mayum's	
Edit Delete	Outback Lager	Pavlova, Ltd.	
Edit Delete	Rhönbräu Klosterbier	Specialty Biscuits, Ltd.	\$18.00
		PB Knäckebröd AB	\$45.00
		Refrescos Americanas LTDA	
		Heli Süßwaren GmbH & Co. KG	
		Plutzer Lebensmittelgroßmärkte AG	
		Nord-Ost-Fisch Handelsgesellschaft mbH	
		Formaggi Fortini s.r.l.	
		Norske Meierier	
		Bigfoot Breweries	
		Svensk Sjöföda AB	
		Aux joyeux ecclésiastiques	
		New England Seafood Cannery	
		Leka Trading	

Chương trình sẽ hỗ trợ người dùng các tính năng sau:

Cho phép người dùng lọc sản phẩm theo phân loại.

Cho phép người dùng sắp xếp các sản phẩm bằng cách nhấp chuột lên tiêu đề cột (Name, Price, Units In Stock, ...).

Cho phép người dùng phân trang các sản phẩm (10 sản phẩm mỗi trang).

Cho phép người dùng chỉnh sửa và cập nhật các chi tiết sản phẩm ngay trên trang.

Cho phép người dùng xóa các sản phẩm trong danh sách.

Ứng dụng web này sẽ được xây dựng với một mô hình dữ liệu hướng đối tượng dùng LINQ to SQL.

Tất cả các quy tắc xử lý và kiểm tra dữ liệu sẽ được xây dựng trong lớp dữ liệu – mà không phải trong lớp giao diện. Điều này sẽ đảm bảo rằng: 1) một tập các quy tắc xử lý đồng nhất sẽ được dùng ở tất cả mọi chỗ trong ứng dụng, 2) chúng ta sẽ phải viết ít code mà không cần lặp lại, và 3) có thể dễ dàng chỉnh sửa/thay đổi các quy tắc xử lý sau này mà không cần cập nhật lại chúng ở nhiều chỗ khác nhau trong ứng dụng.

Chúng ta cũng sẽ tận dụng được ưu điểm của việc phân trang/sắp xếp bên trong LINQ to SQL để đảm bảo rằng các đặc tính đó không được thực hiện bên trong lớp giữa (middle-tier), mà sẽ được thực hiện trong CSDL (có nghĩa là chỉ có 10 sản phẩm được lấy ra trong CSDL tại một thời điểm, chúng ta sẽ không lấy hàng ngàn dòng rồi mới thực hiện phân trang hay sắp xếp trên web server).

<asp:linqdatasource> là gì và nó giúp gì cho chúng ta?</asp:linqdatasource>

Control <asp:linqdatasource> là một ASP.NET control hiện thực hóa mô hình DataSourceControl được giới thiệu trong ASP.NET 2.0. Nó tương tự như các control ObjectDataSource và SqlDataSource, bạn có thể dùng nó để khai báo việc gắn nối dữ liệu giữa một control ASP.NET với một nguồn dữ liệu. Điểm khác biệt là thay vì nó gắn nối trực tiếp vào CSDL (như SqlDataSource) hay vào một lớp (ObjectDataSource), <asp:linqdatasource> được thiết kế để gắn vào một mô hình dữ liệu LINQ.</asp:linqdatasource></asp:linqdatasource>

Một trong những ưu điểm của việc dùng <asp:linqdatasource> là nó tận dụng được tính mềm dẻo của các trình cung cấp LINQ (LINQ provider: như LINQ to SQL, LINQ to Object...). Bạn không cần định nghĩa các phương

thúc query/insert/update/delete cho nguồn dữ liệu để gọi, thay vào đó bạn có thể trả `<asp:linqdatasource>` đến mô hình dữ liệu của bạn, chỉ ra bản thực thể nào bạn muốn làm việc, rồi gắn nó vào một control Asp.NET và cho phép chúng làm việc với nhau.`</asp:linqdatasource></asp:linqdatasource>`

Ví dụ, để xây dựng một danh sách cơ bản các sản phẩm cho phép làm việc với các thực thể Product trong mô hình dữ liệu LINQ to SQL, tôi có thể khai báo một thẻ `<asp:linqdatasource>` trên trang và trả vào lớp datacontext của LINQ to SQL, và chỉ ra các thực thể (ví dụ: Products) trong mô hình LINQ to SQL mà tôi muốn gắn nó. Tôi có thể cho một GridView trả vào nó (bằng cách đặt thuộc tính DataSourceID) để cho phép xem các Product theo dạng lưới:`</asp:linqdatasource>`

```
<asp:GridView ID="GridView1" DataSourceID="SupplierDataSource"
    AllowPaging="true"
    AllowSorting="true"
    runat="server" />

<asp:LinqDataSource ID="SupplierDataSource"
    ContextTypeName="WebApplication12.NorthwindDataContext"
    TableName="Products"
    EnableUpdate="true" EnableDelete="true"
    runat="server" />
```

Không cần làm thêm bất kỳ điều gì, tôi đã có thể thực thi trang web và có một danh sách các Product với khả năng phân trang cũng như sắp xếp được tích hợp sẵn. Tôi cũng có thể thêm một nút edit/delete và cho phép người dùng chỉnh sửa dữ liệu. Tôi không cần thêm bất kỳ phương thức, ánh xạ các tham số, hay thậm chí viết bất kỳ câu lệnh nào cho `<asp:linqdatasource>` để xử lý các trường hợp hiển thị và cập nhật trên – nó có thể làm việc với mô hình LINQ to SQL mà chúng ta chỉ đến và thực hiện các thao tác tự động. Khi cập nhật, LINQ to SQL sẽ đảm bảo rằng các quy tắc xử lý và kiểm tra dữ liệu mà ta đã thêm vào mô hình LINQ to SQL (dưới dạng các phương thức partial) cũng sẽ được thực hiện trước khi dữ liệu được thực sự cập nhật vào CSDL.`</asp:linqdatasource>`

Quan trọng: Một trong những điểm hay của LINQ hay LINQ to SQL là nó không được thiết kế để chỉ làm việc với lớp giao diện, hay với một control cụ thể nào như LinqDataSource. Như bạn đã thấy trong các bài viết trước của cùng loạt bài này, viết code dùng LINQ to SQL cực kỳ sáng sủa. Bạn luôn có thể viết thêm các mã lệnh tùy biến để làm việc trực tiếp với mô hình dữ liệu LINQ to SQL nếu muốn, hay trong một ngữ cảnh nào đó mà `<asp:linqdatasource>` không phù hợp để dùng.`</asp:linqdatasource>`

Các phần dưới đây sẽ mô tả từng bước tạo nên ứng dụng web tôi đã nói ở trên bằng cách dùng LINQ to SQL và
`<asp:linqdatasource>.</asp:linqdatasource>`

Bước 1: Định nghĩa mô hình dữ liệu

Chúng ta sẽ bắt đầu việc tạo ra ứng dụng bằng cách định nghĩa mô hình dữ liệu để biểu diễn CSDL.

Tôi đã nói về cách tạo một mô hình dữ liệu LINQ to SQL dùng trình soạn thảo có trong VS 2008 trong bài 2. Dưới đây là ảnh chụp màn hình các lớp dữ liệu mà tôi có thể nhanh chóng tạo ra dùng LINQ to SQL designer để mô hình hóa CSDL “Northwind”:



Tôi sẽ duyệt lại mô hình này trong bước 5 khi tôi thêm một số quy tắc kiểm tra, nhưng trong giai đoạn đầu, tôi sẽ vẫn dùng nó (chưa chỉnh sửa) để tạo giao diện.

Bước 2: Tạo danh sách sản phẩm

Chúng ta sẽ bắt đầu phần giao diện bằng cách tạo một trang ASP.NET với một control <asp:gridview> và dùng CSS để định dạng:</asp:gridview>

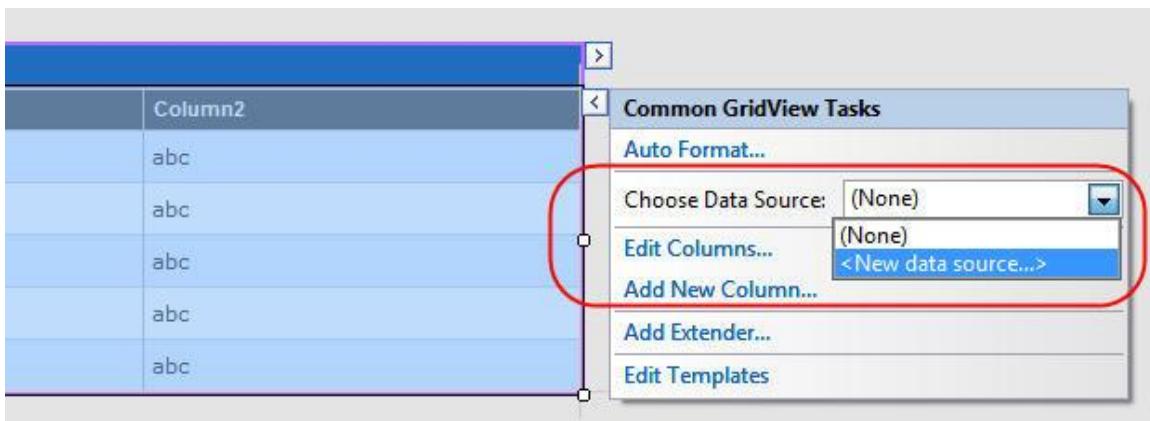


```
<%@ Page Language="vb" AutoEventWireup="false" MasterPageFile="~/Site.Master" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="server">
    <div class="productsheader">
        <h2>Products</h2>
    </div>
    <asp:GridView ID="GridView1" CssClass="gridview"
        AlternatingRowStyle-CssClass="even"
        runat="server" />
</asp:Content>
```

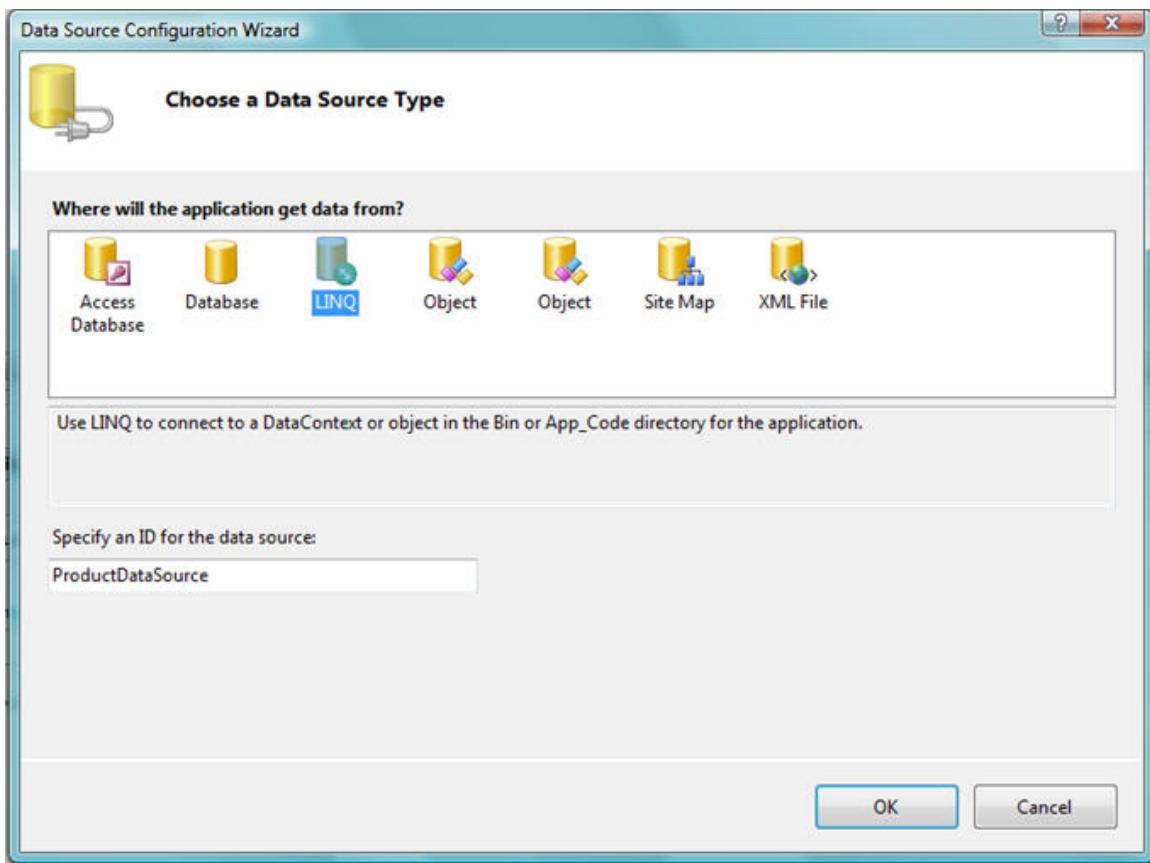
Chúng ta có thể viết code để gắn nối mô hình dữ liệu vào gridview này (giống như tôi đã làm trong phần 3), hoặc tôi có thể làm cách khác là dùng control mới <asp:linqdatasource> để gắn nối gridview này với mô hình dữ liệu.</asp:linqdatasource>

VS 2008 includes build-in designer support to make it easy to connect up our GridView (or any other ASP.NET server control) to LINQ data. To bind our grid above to the data model we created earlier, we can switch into design-view, select the GridView, and then select the “New Data Source...” option within the “Choose Data Source.” drop-down:

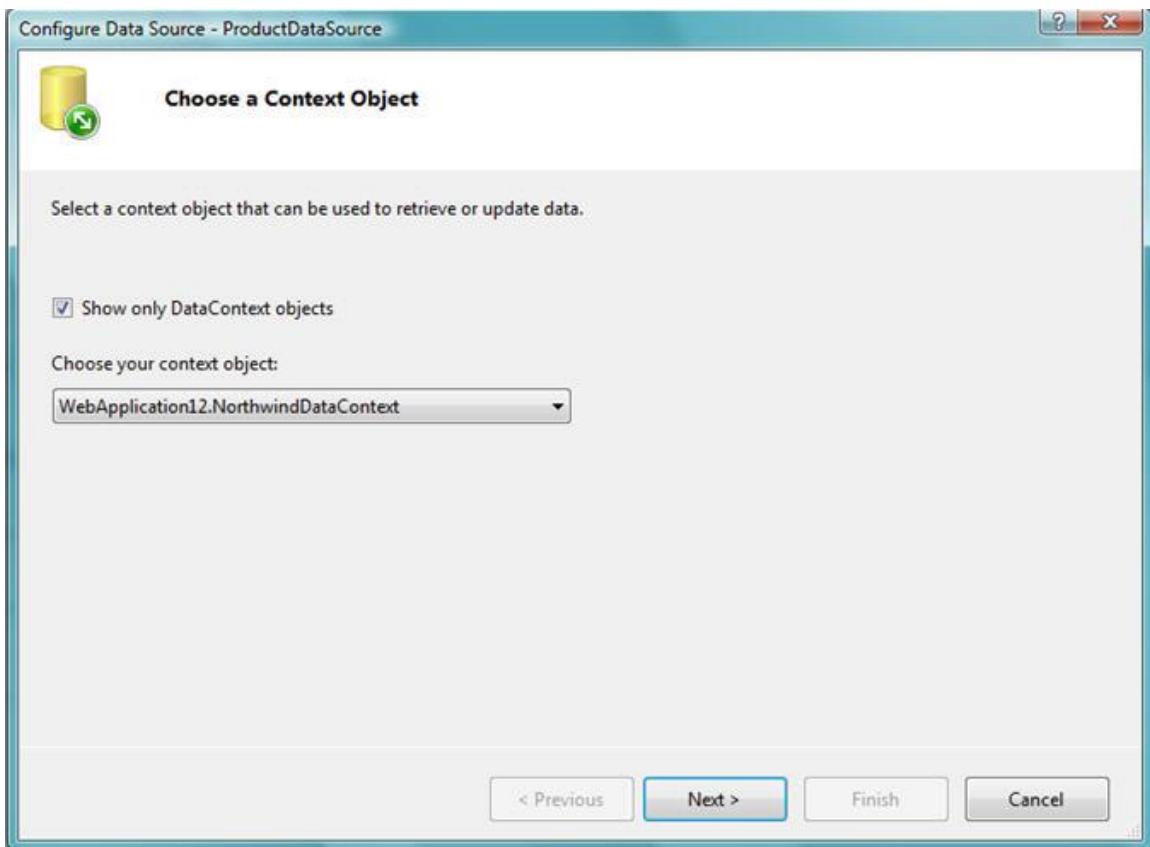
Trình thiết kế trong VS 2008 có sẵn khả năng hỗ trợ làm điều này một cách dễ dàng với GridView (hay bất kỳ control ASP.NET nào khác) vào dữ liệu LINQ. Để gắn nối, chúng ta có thể chuyển sang chế độ thiết kế, chọn GridView, và sau đó chọn “New Data Source...” bên trong danh sách “Choose Data Source.”:



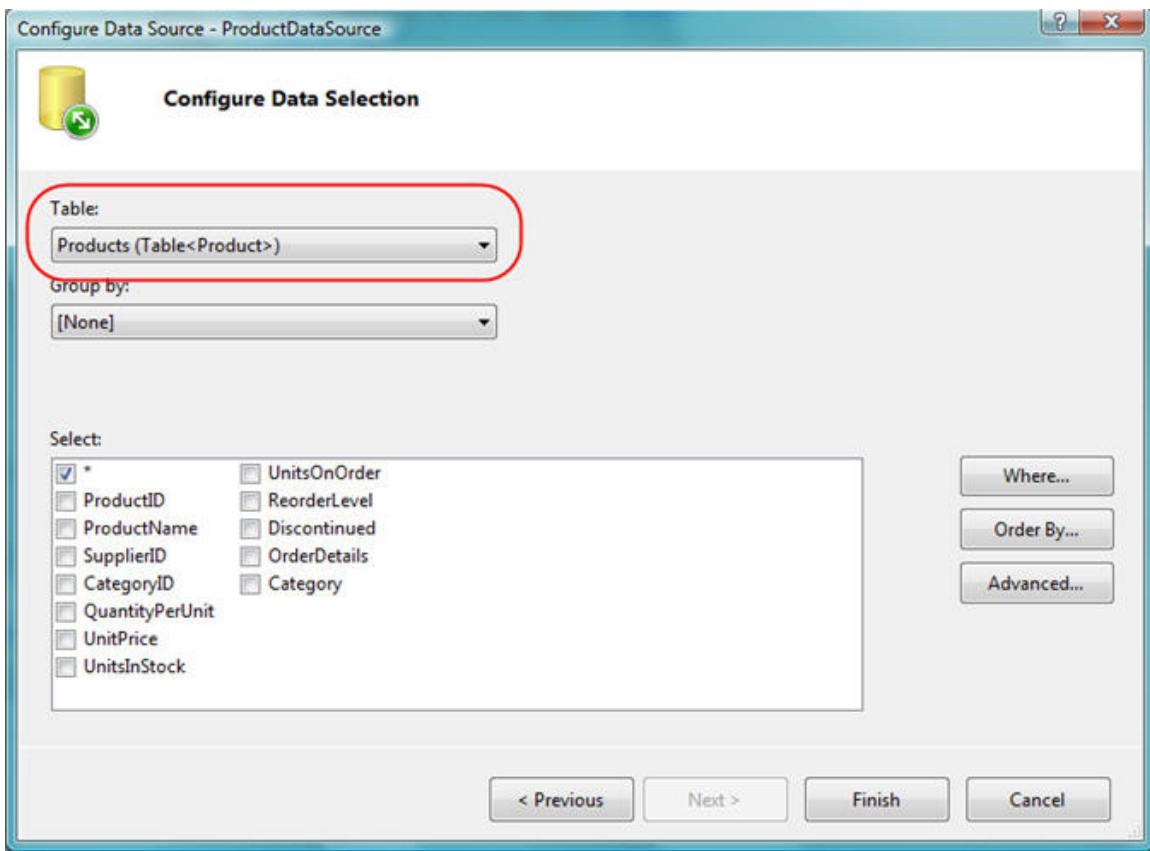
Một hộp thoại sẽ hiện lên, trong đó có danh sách các loại datasource, chọn LINQ trong hộp thoại này và đặt tên cho control <asp:linqdatasource> mà bạn muốn tạo:</asp:linqdatasource>



Trình thiết kế <asp:linqdatasource> sẽ hiển thị tiếp các lớp DataContext của LINQ to SQL mà ứng dụng của bạn có thể dùng được bao gồm cả trong các thư viện mà bạn đang tham chiếu tới):</asp:linqdatasource>



Chúng ta muốn chọn mô hình dữ liệu đã được tạo trước đây với trình thiết kế LINQ to SQL. Chúng ta cũng sẽ muốn chọn bảng dữ liệu bên trong mô hình dữ liệu mà chúng ta sẽ coi như thực thể chính để làm việc với `<asp:linqdatasource>`. Trong ví dụ này chúng ta sẽ chọn thực thể “Products”. Chúng ta cũng sẽ nhấn vào nút “Advanced” và cho phép việc cập nhật cũng như xóa dữ liệu: `</asp:linqdatasource>`

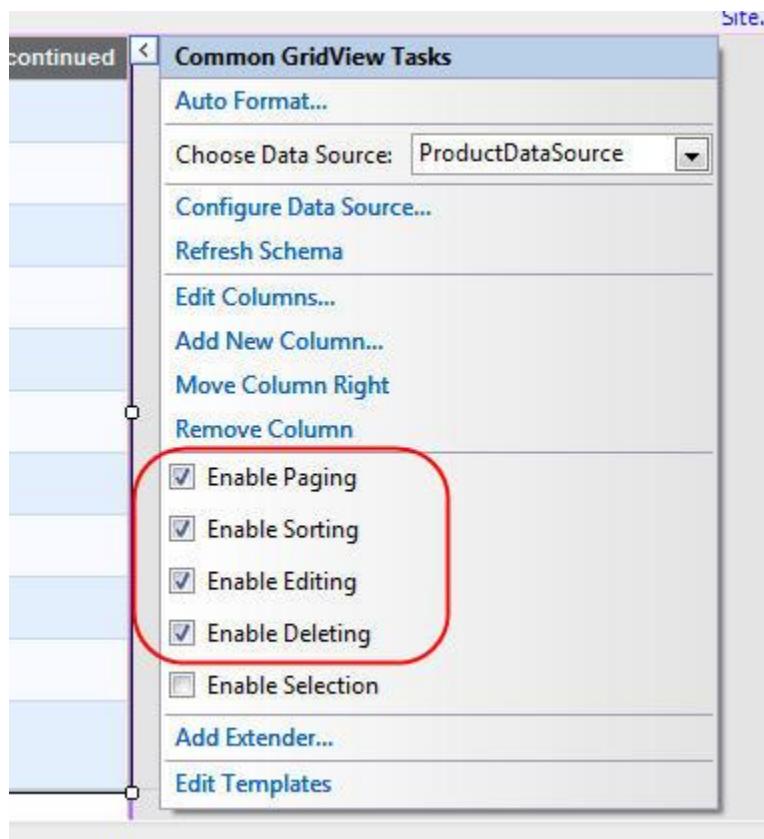


Khi nhấn vào nút “Finish” ở trên, VS 2008 sẽ khai báo một `<asp:linqdatasource>` trong trang .aspx, và cập nhật `<asp:gridview>` để trả đến nó (thông qua thuộc tính `DataSourceID`). Nó cũng sẽ tự động tạo ra các cột trong Grid dựa trên cấu trúc của thực thể `Products` mà chúng ta đã chọn: `</asp:gridview></asp:linqdatasource>`

ContentPlaceHolder1 (Custom)									
Products									
ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
0	abc	0	0	abc	0	0	0	<input type="checkbox"/>	<input type="checkbox"/>
1	abc	1	1	abc	0.1	1	1	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	abc	2	2	abc	0.2	2	2	<input type="checkbox"/>	<input type="checkbox"/>
3	abc	3	3	abc	0.3	3	3	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4	abc	4	4	abc	0.4	4	4	<input type="checkbox"/>	<input type="checkbox"/>

LinqDataSource - ProductDataSource

Chúng ta cũng có thể nhấp vào hình tam giác nhỏ để bật lên “smart task” của GridView và chỉ ra chúng ta muốn cho phép việc phân trang, sắp xếp, chỉnh sửa và xóa dữ liệu:



Chúng ta có thể nhấn F5 để thực thi, và có một trang hiển thị danh sách sản phẩm với đầy đủ khả năng phân trang cũng như sắp xếp các cột:

Products									
	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	Discontinued	
Edit	Delete	Chai	1	1 10 boxes x 20 bags	100.0000	39	0	<input type="checkbox"/>	
Edit	Delete	Chang	1	1 24 - 12 oz bottles	22.0000	16	41	<input type="checkbox"/>	
Edit	Delete	Aniseed Syrup	1	2 12 - 550 ml bottles	10.0000	13	70	<input type="checkbox"/>	
Edit	Delete	Chef Anton's Cajun Seasoning	2	2 48 - 6 oz jars	23.0000	53	0	<input type="checkbox"/>	
Edit	Delete	Chef Anton's Gumbo Mix	2	2 36 boxes	21.3500	0	0	<input checked="" type="checkbox"/>	
Edit	Delete	Grandma's Boysenberry Spread	3	2 12 - 8 oz jars	25.0000	120	0	<input type="checkbox"/>	
Edit	Delete	Uncle Bob's Organic Dried Pears	3	7 12 - 1 lb pkgs.	30.0000	15	0	<input type="checkbox"/>	
Edit	Delete	Northwoods Cranberry Sauce	3	2 12 - 12 oz jars	40.0000	6	0	<input type="checkbox"/>	
Edit	Delete	Mishi Kobe Niku	4	6 18 - 500 g pkgs.	97.0000	29	0	<input checked="" type="checkbox"/>	
Edit	Delete	Ikura	4	8 12 - 200 ml jars	31.0000	31	0	<input type="checkbox"/>	

1 2 3 4 5 6 7 8

Chúng ta cũng có thể nhấn và các nút “edit” hoặc “delete” để cập nhật lại dữ liệu:

Products				
	ProductName	SupplierID	CategoryID	QuantityPerUnit
Edit	Delete	Chai	1	1 10 boxes x 20 bags
Update	Cancel	Chang	1	1 24 - 12 oz bottles
Edit	Delete	Aniseed Syrup	1	2 12 - 550 ml bottles

Nếu nhìn vào mã nguồn của trang, chúng ta sẽ thấy các thẻ của trang chứa nội dung giống như dưới đây. Thẻ `<asp:linqdatasource>` chỉ đến lớp `DataContext` của LINQ to SQL mà ta đã tạo trước đây, cũng như bảng dữ liệu mà chúng ta muốn dùng. GridView sau đó chỉ đến `<asp:linqdatasource>`

(thông qua DataSourceID) và chỉ ra những cột nào sẽ được hiển thị, tiêu đề cột, cũng như cách sắp xếp sẽ được dùng khi tiêu đề cột được chọn.</asp:linqdatasource></asp:linqdatasource>

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <div class="productsheader">
        <h2>Products</h2>
    </div>

    <asp:GridView ID="GridView1" DataSourceID="ProductDataSource" CssClass="gridview" Alternati
        <columns>
            <asp:commandfield ShowDeleteButton="True" ShowEditButton="True"></asp:commandfield>
            <asp:boundfield DataField="ProductName" HeaderText="ProductName" SortExpression="Pr
            <asp:boundfield DataField="SupplierID" HeaderText="SupplierID" SortExpression="Supp
            <asp:boundfield DataField="CategoryID" HeaderText="CategoryID" SortExpression="Cate
            <asp:boundfield DataField="QuantityPerUnit" HeaderText="QuantityPerUnit" SortExpres
            <asp:boundfield DataField="UnitPrice" HeaderText="UnitPrice" SortExpression="UnitPr
            <asp:boundfield DataField="UnitsInStock" HeaderText="UnitsInStock" SortExpression="U
            <asp:boundfield DataField="UnitsOnOrder" HeaderText="UnitsOnOrder" SortExpression="U
            <asp:checkboxfield DataField="Discontinued" HeaderText="Discontinued" SortExpressio
        </columns>
    </asp:GridView>

    <asp:LinqDataSource ID="ProductDataSource"
        ContextTypeName="WebApplication12.NorthwindDataContext"
        TableName="Products"
        EnableDelete="True" EnableUpdate="True"
        runat="server" />

</asp:Content>
```

Giờ chúng ta đã có một trang web cơ bản để làm việc với mô hình dữ liệu LINQ to SQL, chúng ta có thể tiếp tục tùy biến giao diện và hành vi.

Bước 3: Bỏ các cột không cần thiết

GridView của chúng ta ở trên có rất nhiều cột được định nghĩa sẵn, 2 trong số đó (SupplierID và CategoryID) là các cột khóa ngoài, và việc hiển thị các cột này có vẻ như không phải là một ý tưởng hay.

Xóa bỏ các cột không cần thiết

Chúng ta có thể bắt đầu việc dọn dẹp giao diện bằng cách xóa đi một số cột không cần thiết. Tôi có thể làm điều này bằng cách sửa mã nguồn, hay trong chế độ thiết kế (nhấp chuột lên cột muốn xóa và chọn “Remove”). Ví dụ, bạn có thể bỏ cột “Quantity Per Unit” dưới đây và chạy lại ứng dụng của chúng ta để có một giao diện sáng sủa hơn:

Products								
	ProductName	SupplierID	CategoryID	UnitPrice	UnitsInStock	UnitsOnOrder	Discontinued	
Edit	Chai	1	1	100.0000	39	0	<input type="checkbox"/>	
Edit	Chang	1	1	22.0000	16	41	<input type="checkbox"/>	
Edit	Aniseed Syrup	1	2	10.0000	13	70	<input type="checkbox"/>	
Edit	Chef Anton's Cajun Seasoning	2	2	23.0000	53	0	<input type="checkbox"/>	
Edit	Chef Anton's Gumbo Mix	2	2	21.3500	0	0	<input checked="" type="checkbox"/>	
Edit	Grandma's Boysenberry Spread	3	2	25.0000	120	0	<input type="checkbox"/>	
Edit	Uncle Bob's Organic Dried Pears	3	7	30.0000	15	0	<input type="checkbox"/>	
Edit	Northwoods Cranberry Sauce	3	2	40.0000	6	0	<input type="checkbox"/>	
Edit	Mishi Kobe Niku	4	6	97.0000	29	0	<input checked="" type="checkbox"/>	
Edit	Ikura	4	8	31.0000	31	0	<input type="checkbox"/>	

1 2 3 4 5 6 7 8

Nếu bạn đã từng dùng control `<asp:objectdatasource>` trước đây và truyền các tham số cho các phương thức cập nhật, một trong những thứ khốn khổ bạn biết có lẽ là việc thay đổi tham số của các phương thức cập nhật trong TableAdapter khi các thông số được nhận từ lớp giao diện bị thay đổi. Ví dụ: nếu chúng ta xóa một cột trong bảng ở trên, chúng ta lại phải chỉnh sửa lại TableAdapter để nó hỗ trợ các phương thức cập nhật không cần tới tham số đã bị xóa đó.

Một điều hay là với `<asp:linqdatasource>` bạn không cần thực hiện các thay đổi kiểu như vậy. Chỉ đơn giản là thêm hoặc xóa một cột và chạy lại chương trình – không cần làm thêm bất cứ điều gì khác. Điều này làm cho việc thay đổi giao diện web dùng `<asp:linqdatasource>` dễ hơn nhiều.

Xóa các cột SupplierID và CategoryID

Hiện tại, chúng ta hiển thị các giá trị khóa ngoài đến các bảng Supplier và Category trong GridView:

Products								
	ProductName	SupplierID	CategoryID	UnitPrice	UnitsInStock	UnitsOnOrder	Discontinued	
Edit	Chai	1	1	100.0000	39	0	<input type="checkbox"/>	
Edit	Chang	1	1	22.0000	16	41	<input type="checkbox"/>	

Điều này tuy cần thiết đúng từ góc độ mô hình dữ liệu, nhưng lại không mang lại giá trị gì cho người dùng. Thú mà chúng ta làm là hiển thị CategoryName và SupplierName, và cung cấp một danh sách xổ xuống trong chế độ Edit cho phép người dùng dễ dàng chọn các giá trị cho SupplierID và CategoryID.

Tôi có thể thay đổi GridView để hiển thị Supplier Name và Category Name thay vì ID bằng việc thay thế <asp:boundfield> với một <asp:templatefield>. Trong TemplateField này, tôi có thể thêm bất kỳ nội dung nào tôi muốn để tùy biến lại cách hiển thị của cột dữ liệu.</asp:templatefield></asp:boundfield>

Trong đoạn mã dưới đây, tôi sẽ tận dụng các thuộc tính Supplier và Category trên mỗi Product, nhờ đó tôi có thể dễ dàng gắn nối các cột Supplier.CompanyName và Category.CategoryName và các cột tương ứng trong Grid.

```
<asp:GridView ID="GridView1" DataSourceID="ProductDataSource" CssClass="gridview" AlternatingRowColor="#D9E1F2" OnRowCommand="GridView1_RowCommand" OnRowDataBound="GridView1_RowDataBound" OnRowCreated="GridView1_RowCreated" OnRowDeleting="GridView1_RowDeleting" OnRowEditing="GridView1_RowEditing" OnRowUpdating="GridView1_RowUpdating">
    <columns>
        <asp:commandfield ShowDeleteButton="True" ShowEditButton="True"></asp:commandfield>
        <asp:boundfield DataField="ProductName" HeaderText="ProductName" SortExpression="ProductName" Type="String" ValidationGroup="ProductValidationGroup" ValidationType="Required" DataFormatString="<b>{0}</b>"></asp:boundfield>
        <asp:TemplateField HeaderText="Supplier" SortExpression="Supplier.CompanyName">
            <ItemTemplate>
                <%#Eval("Supplier.CompanyName")%>
            </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Category" SortExpression="Category.CategoryName">
            <ItemTemplate>
                <%#Eval("Category.CategoryName") %>
            </ItemTemplate>
        </asp:TemplateField>
        <asp:boundfield DataField="UnitPrice" HeaderText="UnitPrice" SortExpression="UnitPrice" Type="Number" DataFormatString="<b>{0}</b>"></asp:boundfield>
        <asp:boundfield DataField="UnitsInStock" HeaderText="UnitsInStock" SortExpression="UnitsInStock" Type="Number" DataFormatString="<b>{0}</b>"></asp:boundfield>
        <asp:boundfield DataField="UnitsOnOrder" HeaderText="UnitsOnOrder" SortExpression="UnitsOnOrder" Type="Number" DataFormatString="<b>{0}</b>"></asp:boundfield>
        <asp:checkboxfield DataField="Discontinued" HeaderText="Discontinued" SortExpression="Discontinued" Type="Boolean" DataFormatString="<b>{0}</b>"></asp:checkboxfield>
    </columns>
</asp:GridView>
```

Và bây giờ khi chạy ứng dụng, tôi sẽ có danh sách các Category và Supplier theo tên:

Products								
	ProductName	Supplier	Category	UnitPrice	UnitsInStock	UnitsOnOrder	Discontinued	
Edit	Côte de Blaye	Aux joyeux ecclésiastiques	Beverages	263.5000	17	0	<input type="checkbox"/>	
Edit	Chartreuse verte	Aux joyeux ecclésiastiques	Beverages	18.0000	69	0	<input type="checkbox"/>	
Edit	Sasquatch Ale	Bigfoot Breweries	Beverages	15.0000	111	0	<input type="checkbox"/>	
Edit	Steeleye Stout	Bigfoot Breweries	Beverages	13.0000	20	0	<input type="checkbox"/>	
Edit	Laughing Lumberjack Lager	Bigfoot Breweries	Beverages	14.0000	52	0	<input type="checkbox"/>	
Edit	Queso Cabrales	'Cooperativa de Quesos 'Las Cabras'	Dairy Products	22.0000	22	30	<input type="checkbox"/>	
Edit	Queso Manchego La Pastora	'Cooperativa de Quesos 'Las Cabras'	Dairy Products	38.0000	86	0	<input type="checkbox"/>	
Edit	Escargots de Bourgogne	Escargots Nouveaux	Seafood	13.2500	62	0	<input type="checkbox"/>	
Edit	Aniseed Syrup	Exotic Liquids	Condiments	10.0000	13	70	<input type="checkbox"/>	
Edit	Chang	Exotic Liquids	Beverages	22.0000	16	41	<input type="checkbox"/>	

1 2 3 4 5 6 7 8

Để tạo ra danh sách cho phép người dùng chọn các giá trị của các cột Supplier và Category trong chế độ Edit, đầu tiên tôi sẽ thêm hai control `<asp:linqdatasource>` nữa vào trang. Tôi sẽ cấu hình chúng để gắn nối với Categories và Suppliers bên trong mô hình dữ liệu LINQ to SQL mà ta đã tạo trước đây:

```

<asp:LinqDataSource ID="CategoryDataSource"
    ContextTypeName="WebApplication12.NorthwindDataContext"
    TableName="Categories"
    runat="server" />

<asp:LinqDataSource ID="SupplierDataSource"
    ContextTypeName="WebApplication12.NorthwindDataContext"
    TableName="Suppliers"
    runat="server" />

```

Tôi có thể quay trở lại các cột `<asp:templatefield>` mà chúng ta đã tạo và tùy biến giao diện Edit của chúng (bằng cách chỉ ra `EditItemTemplate`). Chúng ta cũng sẽ tùy biến mỗi cột để có một danh sách trong chế độ Edit, và các giá trị sẽ được lấy từ các datasource CategoryDataSource và SupplierDataSource ở trên, và các một liên hệ này sẽ là 2 chiều:

```

<asp:TemplateField HeaderText="Supplier" SortExpression="Supplier.CompanyName">
    <ItemTemplate>
        <%#Eval("Supplier.CompanyName")%>
    </ItemTemplate>
    <EditItemTemplate>
        <asp:DropDownList ID="DropDownList1"
            DataSourceID="SupplierDataSource"
            DataValueField="SupplierID"
            DataTextField="CompanyName"
            SelectedValue='<%#Bind("SupplierID") %>'
            runat="server" />
    </EditItemTemplate>
</asp:TemplateField>

```

Và giờ, khi người dùng nhấp chuột lên Edit trên GridView, chúng sẽ được hiển thị như một danh sách Supplier mà sản phẩm đang chọn kết hợp:

Products					
	ProductName	Supplier	Category	UnitP	UnitS
Edit Delete	Chai	Exotic Liquids	Beverages	100.00	20.00
Update Cancel	Chang	Exotic Liquids	Beverages	22.00	5.00
Edit Delete	Aniseed Syrup	New Orleans Cajun Delights	Condiments	10.00	1.20
Edit Delete	Chef Anton's Cajun Seasoning	Grandma Kelly's Homestead	Condiments	23.00	6.00
Edit Delete	Chef Anton's Gumbo Mix	Tokyo Traders	Condiments	21.35	5.00
		Cooperativa de Quesos 'Las Cabras'			
		Mayumi's Pavlova, Ltd.			

Và khi bạn bấm nút Save, sản phẩm sẽ được cập nhật một cách phù hợp (GridView sẽ dùng giá trị của dòng được chọn hiện tại trong DropDownList để đưa vào SupplierID).

Bước 4: Lọc danh sách sản phẩm

Thay vì hiển thị tất cả các sản phẩm trong CSDL, bạn có thể cập nhật phần giao diện để nó thêm một danh sách cho phép người dùng lọc lại các sản phẩm theo một phân loại nào đó.

Vì chúng ta đã thêm control `<asp:linqdatasource>` tham chiếu đến Categories vào trang web này trước đây, do vậy giờ những gì cần làm chỉ là tạo một dropdownlist trên đầu trang để gắn nối với nó. Ví dụ:`</asp:linqdatasource>`

```

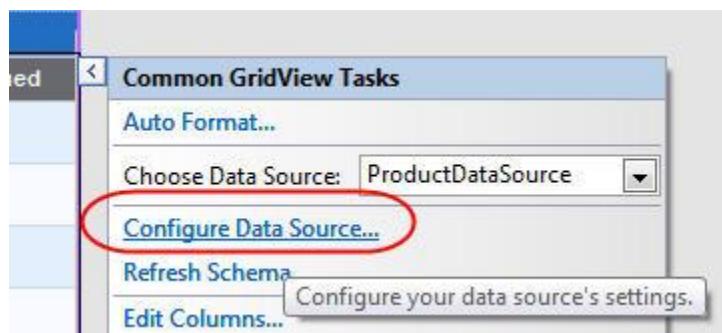
<div class="category">
    Pick Category:
    <asp:DropDownList ID="CategoryList"
        DataSourceID="CategoryDataSource"
        DataTextField="CategoryName"
        DataValueField="CategoryID"
        AutoPostBack="True"
        runat="server">
    </asp:DropDownList>
</div>

```

Khi tôi chạy trang web này, tôi sẽ có trên đầu trang một danh sách cho tất cả các mục phân loại:

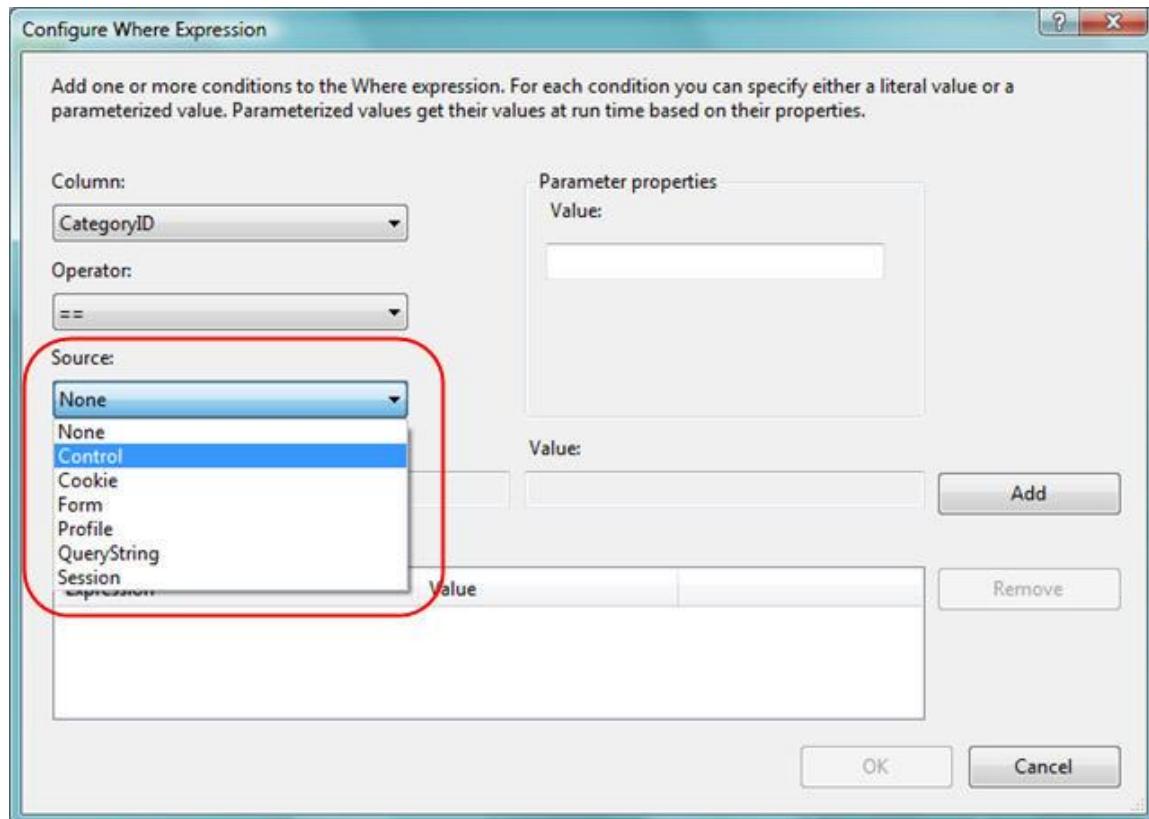


Bước cuối cùng là cấu hình GridView để nó chỉ hiển thị các sản phẩm trong phân loại được chọn, cách dễ nhất là chọn “Configure DataSource” trong smart task của GridView:

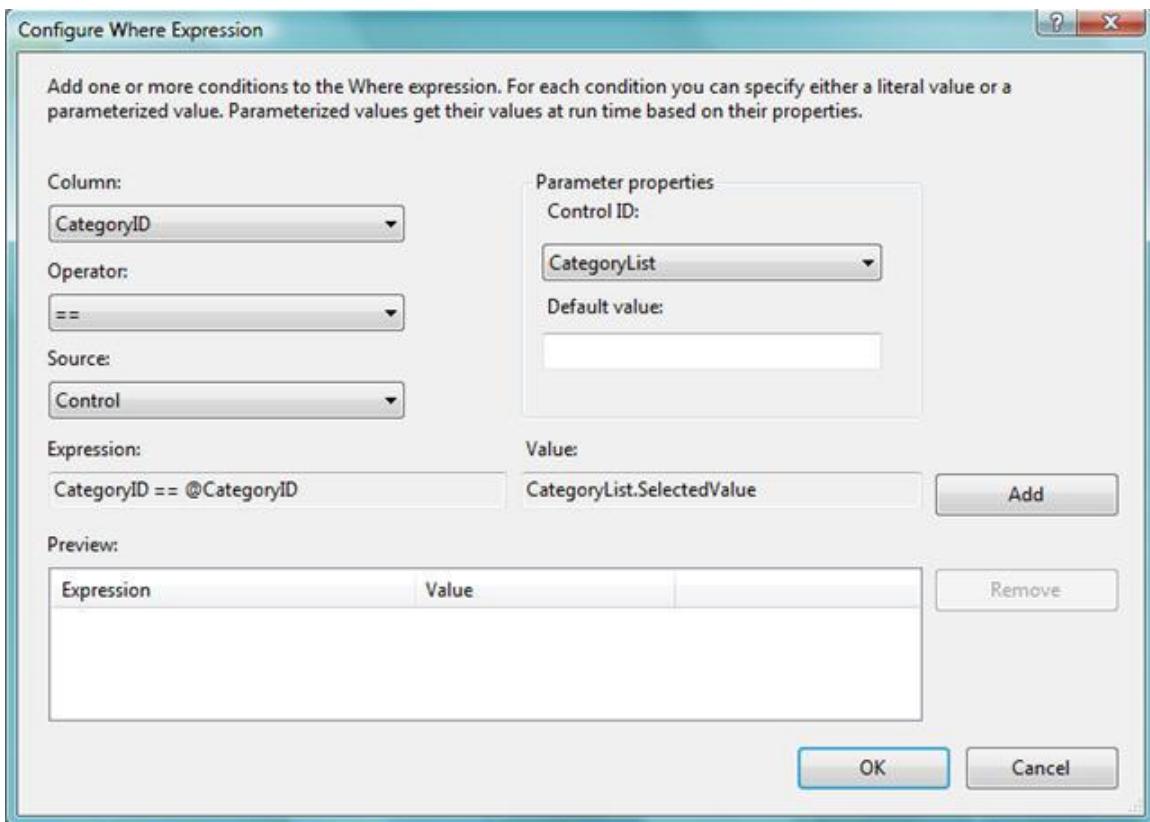


Nó sẽ đưa tôi quay trở lại cửa sổ thiết kế <asp:linqdatasource> mà tôi đã dùng trong phần đầu bài viết này. Tôi có thể chọn nút “Where” trong cửa sổ

này để thêm một bộ lọc vào control datasource. Tôi có thể tạo ra nhiều bộ lọc nếu cần, và kéo các giá trị để lọc từ một vài chỗ khác nhau (ví dụ: từ querystring (trên web), từ các giá trị trên form, từ các control khác trên trang...):</asp:linqdatasource>



Ở trên, tôi sẽ tạo bộ lọc các Products theo CategoryID, và sau đó lấy giá trị của CategoryID muốn lọc từ danh sách mà chúng ta đã tạo trên trang:



Sau khi bấm Finish, control <asp:linqdatasource> trên trang của chúng ta sẽ được cập nhật để sử dụng bộ lọc giống như sau:</asp:linqdatasource>

```
<asp:LinqDataSource ID="ProductDataSource" ContextTypeName="WebApplication12.NorthwindDataContext" Tal  
    <wherereparameters>  
        <asp:controlparameter ControlID="CategoryList" Name="CategoryID" PropertyName="SelectedValue" />  
    </wherereparameters>  
</asp:LinqDataSource>
```

Và bây giờ nếu thực thi trang web, người dùng sẽ có thể chọn một trong các phân loại có sẵn và sau đó phân trang, sắp xếp, chỉnh sửa hay xóa các sản phẩm trong phân loại đó:

Pick Category: Confections

Products				
	ProductName	Supplier	Category	UnitPrice
Edit Delete	Valkoinen suklaa	Karkki Oy	Confections	16.25
Update Cancel	Tarte au sucre	Forêts d'érables	Confections	49.30
Edit Delete	Scottish Longbreads	Specialty Biscuits, Ltd.	Confections	12.50

1 2

Control `<asp:linqdatasource>` sẽ tự động áp dụng các bộ lọc LINQ cần thiết khi làm việc với các lớp LINQ to SQL của chúng ta để đảm bảo rằng chỉ có các dữ liệu cần thiết được lấy về từ CSDL (ví dụ: trong Grid ở trên, chỉ có 3 dòng sản phẩm từ trang thứ hai trong nhóm các sản phẩm Confection được lấy về).</asp:linqdatasource>

Bạn có thể sử dụng sự kiện Selecting trên `<asp:linqdatasource>` nếu muốn tùy biến câu truy vấn LINQ trong đoạn code.</asp:linqdatasource>

Bước 5: Thêm các quy tắc kiểm tra logic

Như tôi đã nói đến trong phần 4 của loạt bài LINQ to SQL này, khi chúng ta định nghĩa mô hình dữ liệu LINQ to SQL, mặc nhiên chúng ta sẽ tự động có một tập hợp các ràng buộc trong các lớp mô hình dữ liệu, các ràng buộc này được sinh ra dựa trên định nghĩa trong CSDL. Điều này có nghĩa là nếu bạn thử nhập một giá trị null vào cho một cột mandatory, gán một string vào cho một cột số nguyên, hay đặt giá trị cho khóa ngoài cho một dòng không tồn tại, mô hình LINQ to SQL của chúng ta sẽ phát ra một lỗi và nhở vạy CSDL được toàn vẹn.

Việc kiểm tra theo cách này chỉ nhằm đảm bảo sự toàn vẹn ở mức cơ bản, dù vậy, nó vẫn đủ cho hầu hết các ứng dụng trong thực tế. Chúng ta cũng có thể mong muốn thêm vào các quy tắc logic ở một mức độ cao hơn, cho phép kiểm tra các quy tắc business vào trong các lớp mô hình dữ liệu. Xin cảm ơn LINQ to SQL đã cho phép làm điều này thật dễ dàng (để xem chi tiết, xin đọc lại phần 4).

Một ví dụ về các quy tắc logic

Lấy ví dụ, ngoài những quy tắc logic cơ bản, chúng ta còn muốn đảm bảo rằng người dùng sẽ không thể ngưng bán một loại sản phẩm nếu vẫn còn sản phẩm loại đó trong kho hàng.

Category	UnitPrice	UnitsIn Stock	UnitsOnOrder	Discontinued
Beverages	\$10.00	39	0	<input type="checkbox"/>
Beverages	\$5.50	20	44	<input type="checkbox"/>
Beverages	\$15.00	111	0	<input type="checkbox"/>
Beverages	13.0000	20	23	<input checked="" type="checkbox"/>
Beverages	\$263.50	17	0	<input type="checkbox"/>
Beverages	\$18.00	69	0	<input type="checkbox"/>

Nếu một người dùng nhấn nút Save dòng ở trên, chúng ta sẽ không cho phép việc thay đổi được lưu lại và phát ra một lỗi để báo cho người dùng.

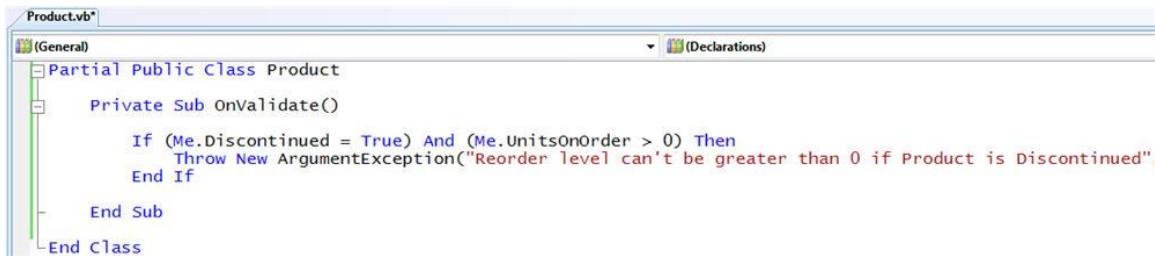
Thêm một quy tắc kiểm tra mô hình dữ liệu

Nếu kiểm tra các quy tắc này ở lớp giao diện thì sẽ là không phù hợp, vì khi đó quy tắc này sẽ chỉ được áp dụng cho chính nơi đó, và sẽ không tự động được áp dụng nếu chúng ta thêm một trang khác cũng cho phép cập nhật Product vào ứng dụng. Việc phân tán các quy tắc kiểm tra logic/business vào lớp giao diện sẽ làm cho việc bảo trì trở nên khó khăn khi ứng dụng trở nên lớn và phức tạp, vì các thay đổi/cập nhật đều cần áp dụng các thao tác cần thiết ở nhiều chỗ khác nhau.

Nơi được coi là phù hợp để đặt các quy tắc kiểm tra này là trong các lớp mô hình dữ liệu LINQ to SQL mà chúng ta đã định nghĩa trước đây. Như đã đề cập đến trong phần 4, tất cả các lớp được sinh ra bởi LINQ to SQL designer đều được định nghĩa như các lớp “partial” – nó cho phép chúng ta có thể dễ dàng thêm vào các phương thức/sự kiện/thuộc tính. Các lớp mô hình dữ liệu LINQ to SQL sẽ tự động gọi các phương thức kiểm tra mà chúng ta có thể viết ra để thực hiện việc kiểm tra theo mong muốn riêng.

Ví dụ, tôi có thể thêm một lớp partial vào ứng dụng để hiện thực phương thức OnValidate() mà LINQ to SQL sẽ gọi trước khi lưu một đối tượng Product vào CSDL. Bên trong phương thức này tôi có thể thêm quy tắc sau

để đảm bảo rằng các sản phẩm không thể có một ReOrder Level nếu sản phẩm đã ngưng bán:



```
Product.vb*
(General) (Declarations)
Partial Public Class Product
    Private Sub OnValidate()
        If (Me.Discontinued = True) And (Me.UnitsOnOrder > 0) Then
            Throw New ArgumentException("Reorder level can't be greater than 0 if Product is Discontinued");
        End If
    End Sub
End Class
```

Một khi đã thêm lớp ở trên vào dự án, quy tắc business ở trên sẽ được áp dụng bất kỳ lúc nào người dùng dùng đến mô hình dữ liệu và chỉnh sửa lại CSDL. Điều này được áp dụng cho cả việc thêm một sản phẩm mới, cũng như cập nhật lại một sản phẩm đã có.

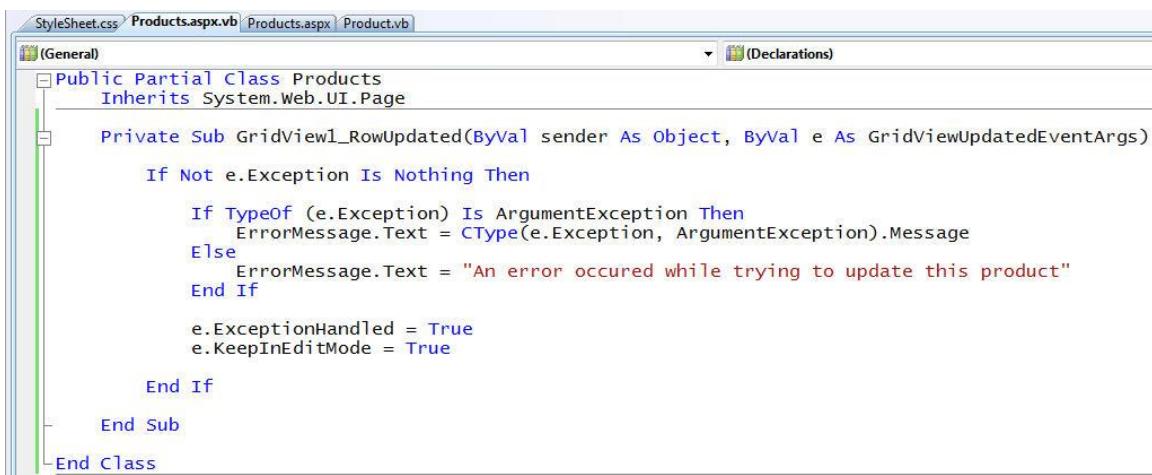
Vì `<asp:linqdatasource>` mà chúng ta đã định nghĩa ở trên làm việc với các lớp mô hình dữ liệu LINQ to SQL, do vậy các thao tác cập nhật/xóa/thêm đều phải qua được phép kiểm tra ở trên trước khi được áp dụng vào CSDL. Chúng ta không cần làm thêm bất kỳ điều gì ở lớp UI để phép kiểm tra này được thực hiện – nó sẽ tự động được dùng bất kỳ nơi nào cũng như bất kỳ lúc nào mô hình LINQ to SQL được dùng.

Thêm phần kiểm soát lỗi vào giao diện

Mặc nhiên nếu người dùng nhập vào một giá trị không hợp lệ cho `UnitsOnOrder/Discontinued` vào `GridView`, các lớp LINQ to SQL của chúng ta sẽ sinh ra một exception. Đến lượt `<asp:linqdatasource>` sẽ bắt lỗi này và cung cấp một sự kiện mà người sử dụng có thể dùng để xử lý lỗi đó. Nếu không có trình xử lý lỗi nào được cung cấp, khi đó `GridView` (hoặc một control khác) gắn nối vào `<asp:linqdatasource>` sẽ bắt lỗi này và cung cấp một event để người dùng có thể xử lý nó. Nếu lại tiếp tục không có ai xử lý lỗi, khi đó nó sẽ được chuyển đến cho `Page`, và chuyển đến hàm xử lý `Application_Error()` trong file `Global.asax` nếu vẫn không có trình xử lý lỗi. Các nhà phát triển có thể chọn bất kỳ chỗ nào trong chuỗi xử lý này để cung cấp một cách tương tác hợp lý nhất đến người dùng cuối.

Đối với ứng dụng của chúng ta, nơi hợp lý nhất đến xử lý các lỗi cập nhật dữ liệu là bắt sự kiện `RowUpdated` trên `GridView`. Sự kiện này sẽ được phát ra mỗi khi một lệnh cập nhật được thực hiện trên datasource, và chúng ta có thể

truy cập thông tin chi tiết của exception nếu việc cập nhật không thành công, sau đó hiển thị thông báo thích hợp cho người dùng.



```
StyleSheet.css Products.aspx.vb Products.aspx Product.vb
(General) (Declarations)

Public Partial Class Products
    Inherits System.Web.UI.Page

    Private Sub GridView1_RowUpdated(ByVal sender As Object, ByVal e As GridViewUpdatedEventArgs)
        If Not e.Exception Is Nothing Then
            If TypeOf (e.Exception) Is ArgumentException Then
                ErrorMessage.Text = CType(e.Exception, ArgumentException).Message
            Else
                ErrorMessage.Text = "An error occurred while trying to update this product"
            End If
            e.ExceptionHandled = True
            e.KeepInEditMode = True
        End If
    End Sub
End Class
```

Để ý rằng ở trên tôi không hề thêm bất kỳ hàm kiểm tra nào vào lớp giao diện. Thay vì vậy, tôi sẽ lấy về chuỗi thông báo lỗi của exception đã phát ra từ phần business logic và hiển thị nó cho người dùng.

Chú ý là tôi cũng đã chỉ ra ở trên là tôi muốn GridView vẫn ở trong chế độ Edit khi lỗi xảy ra – bằng cách đó người dùng sẽ không bị mất đi những thay đổi mà họ đã tạo ra, và có thể chỉnh sửa các giá trị họ đã nhập vào và click nút “update” một lần nữa để lưu lại. Chúng ta cũng có thể thêm một control <asp:literal> với ID “ErrorMessage” bất kỳ chỗ nào mà ta muốn thông báo lỗi hiện ra:</asp:literal>

```
<span id="errorMsg">
    <asp:literal ID="ErrorMessage" runat="server" />
</span>
```

Và bấy giờ chúng ta sẽ thử cập nhật Product với các giá trị kết hợp không hợp lệ, chúng ta sẽ thấy một thông báo lỗi, nhờ đó người dùng sẽ biết cách sửa lại cho phù hợp:

	ProductName	Supplier	Category	UnitPrice
Edit Delete	Chai	Exotic Liquids	Beverages	\$10.00
Update Cancel	Guaraná Fantástica	Refrescos Americanas LTDA	Beverages	5.5000
Edit Delete	Sasquatch Ale	Bigfoot Breweries	Beverages	\$15.00

Một trong những ưu điểm khi làm theo cách trên là tôi có thể thêm hay thay đổi các quy tắc trong mô hình dữ liệu mà không cần chỉnh sửa lại code trong lớp giao diện để có thể hiển thị thông báo phù hợp. Các quy tắc xác thực, và thông báo lỗi tương ứng, có thể được viết ở một chỗ trong lớp mô hình dữ liệu và sẽ được áp dụng phù hợp bất kỳ khi nào bạn dùng nó.

Tổng kết

Control `<asp:linqdatasource>` cung cấp một cách dễ dàng để gắn nối bất kỳ control ASP.NET vào một mô hình dữ liệu LINQ to SQL. Nó cho phép các control dùng hiển thị giao diện có thể vừa lấy dữ liệu về từ LINQ to SQL, cũng như áp dụng các thay đổi thêm/xóa/sửa vào mô hình dữ liệu.

Trong ứng dụng ở trên, chúng ta đã dùng LINQ to SQL designer để tạo ra một mô hình dữ liệu rõ ràng và hướng đối tượng. Chúng ta sau đó thêm ba control ASP.NET vào trang (GridView, DropDownList, ErrorMessage Literal), và thêm ba control `<asp:linqdatasource>` để gắn nối dữ liệu cho Product, Category, và Supplier.

Chúng ta sau đó viết thêm 5 dòng để kiểm tra dữ liệu trong lớp mô hình dữ liệu, và 11 dòng trong lớp giao diện để xử lý lỗi.

Kết quả cuối cùng là một ứng dụng web đơn giản với giao diện được tùy biến cho phép người dùng lọc dữ liệu động theo phân loại, sắp xếp và phân trang một cách hiệu quả trên danh sách sản phẩm, chỉnh sửa trực tiếp thông tin sản phẩm và cho phép lưu lại các thay đổi, và xóa các sản phẩm từ hệ thống.

Trong bài viết tiếp theo của loạt bài này, chúng ta sẽ khám phá thêm LINQ to SQL bao gồm kiểm soát truy xuất đồng thời, lazy loading, thừa kế các ánh xạ bảng, cũng như cách dùng các thủ tục SQL để tùy biến.

Trong bài viết hôm nay, tôi sẽ cho thấy cách chúng ta có thể dùng các stored procedure (SPROCs) và các hàm do người dùng định nghĩa (UDFs) với mô hình dữ liệu LINQ to SQL. Bài viết này sẽ tập trung chủ yếu vào cách dùng SPROCs để truy vấn và lấy dữ liệu về từ CSDL. Trong bài viết kế tiếp, tôi sẽ hiển thị cách bạn có thể dùng các SPROCs để cập nhật, thêm, xóa dữ liệu từ CSDL.

Dùng SPROC hay không SPROC? Đó là một vấn đề....

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, kiến trúc sư phần mềm và các DBA. Rất nhiều người thông minh hơn tôi nhiều đã viết về chủ đề này, vì vậy tôi sẽ không nói thêm về vấn đề này ở đây nữa.

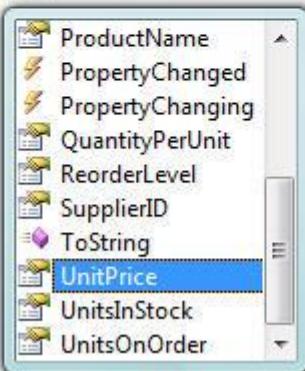
LINQ to SQL đi cùng với .NET 3.5 rất mềm dẻo, và có thể được dùng để tạo các lớp mô hình dữ liệu, trong đó các đối tượng không phụ thuộc vào cấu trúc CSDL phía dưới, và có thể xử lý các phép kiểm tra logic cũng như xác thực tính hợp lệ của dữ liệu mà không phụ thuộc vào việc dữ liệu sẽ được lưu nạp dùng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

Trong bài Truy vấn Cơ sở dữ liệu (phần 3), tôi đã thảo luận cách bạn có thể viết các biểu thức truy vấn LINQ cho một mô hình dữ liệu LINQ to SQL dùng đoạn mã như sau:

```

NorthwindDataContext northwind = new NorthwindDataContext();
var products = from p in northwind.Products
               where p.Category.CategoryName == "Beverages"
               orderby p.UnitPrice descending
               select p;
foreach (Product product in products)
{
    product.
}

```



Khi bạn viết các biểu thức LINQ kiểu như vậy, LINQ to SQL sẽ thực thi các câu lệnh SQL động để bạn có thể lấy về các đối tượng khớp với câu truy vấn của bạn.

Như bạn đã được học trong bài viết này, bạn cũng có thể 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:

```

NorthwindDataContext northwind = new NorthwindDataContext();
var products = northwind.GetProductsByCategoryName("Beverages");
foreach (Product product in products)
{
    product.
}

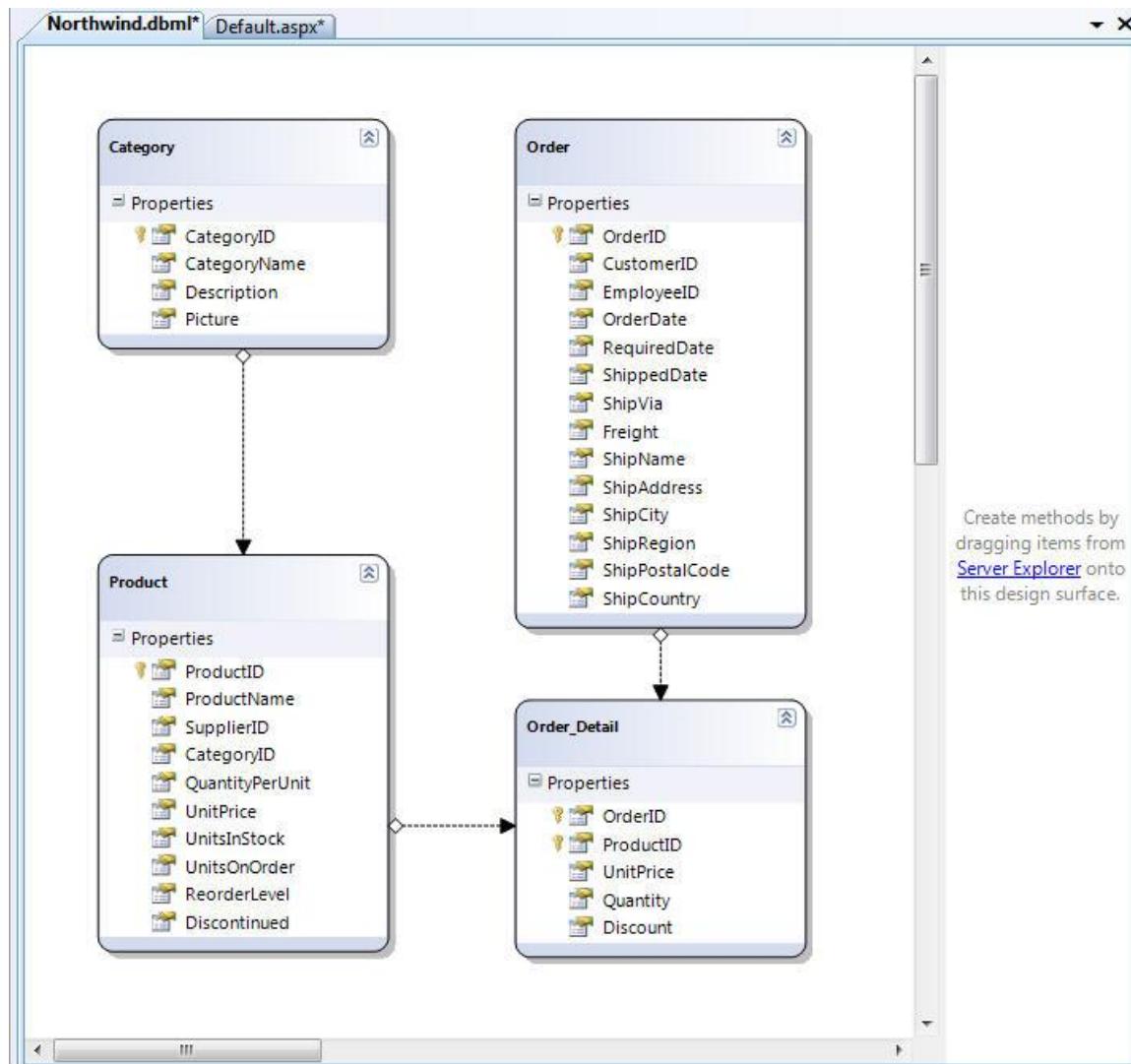
```



Khả năng này cho phép bạn dùng cả các câu SQL động và các SPROCs với một mô hình dữ liệu rõ ràng, mạnh mẽ cũng như cung cấp sự mềm dẻo khi làm việc với các dự án.

Các bước ánh xạ và gọi SPROC dùng LINQ to SQL

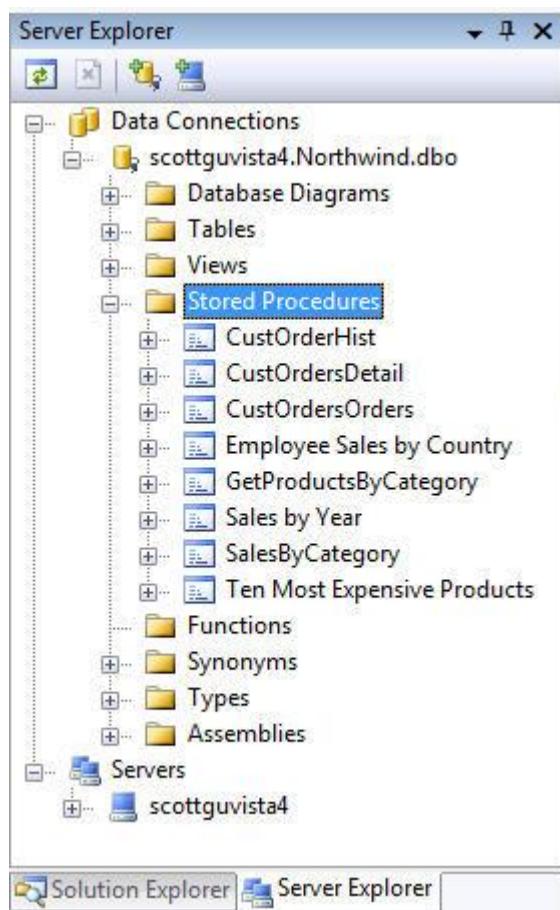
Trong phần 2, tôi đã nói về cách dùng LINQ to SQL designer để tạo ra một mô hình dữ liệu LINQ to SQL như dưới đây:



Ở cửa sổ trên có chứa 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.

Cách ánh xạ một SPROC vào một DataContext của LINQ

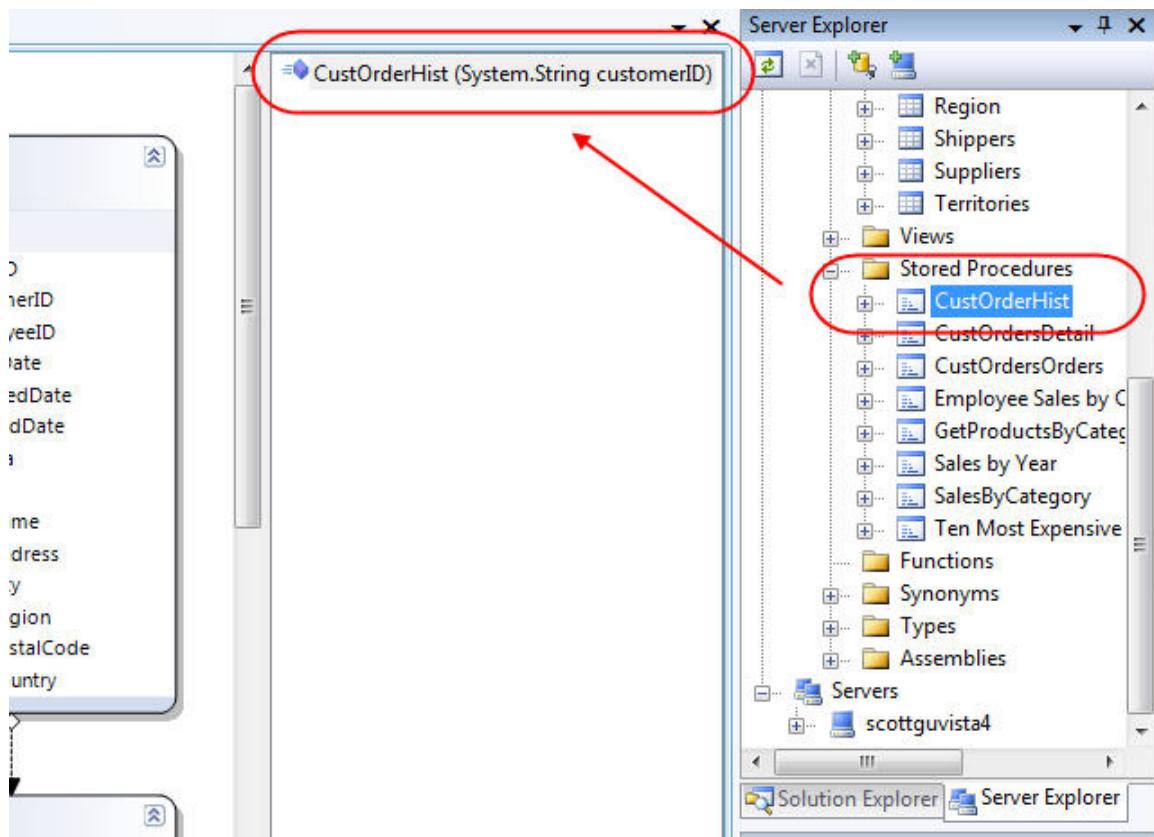
Để ánh xạ một SPROC vào lớp DataContext, trước tiên hãy mở cửa sổ Server Explorer trong VS 2008 và mở danh sách các SPROC trong CSDL:



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

```
ALTER PROCEDURE CustorderHist @CustomerID nchar(5)
AS
SELECT ProductName, Total=SUM(Quantity)
FROM Products P, [Order Details] OD, Orders O, Customers C
WHERE C.CustomerID = @CustomerID
AND C.CustomerID = O.CustomerID AND O.OrderID = OD.OrderID AND OD.ProductID = P.ProductID
GROUP BY ProductName
```

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

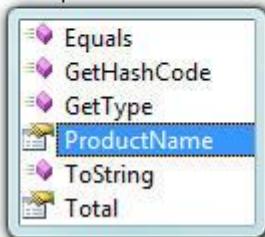


Mặc nhiên tên của phương thức được tạo trong lớp `DataContext` sẽ chính là tên của SPROC, và kiểu trả về của phương thức sẽ là một kiểu được tạo tự động với cách đặt tên theo dạng “[SprocName]Result”. Ví dụ: SPROC ở trên sẽ trả về một dãy các đối tượng có kiểu “`CustOrderHistResult`”. Chúng ta có thể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

Cách gọi SPROC mới được tạo

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

```
NorthwindDataContext northwind = new NorthwindDataContext();
var productHistory = northwind.CustOrderHist("ALFKI");
foreach (CustOrderHistResult product in productHistory)
{
    product.|
```



Thêm nữa, thay vì lặp qua tập kết quả như ở trên, tôi cũng có thể gắn nó vào cho một control để hiển thị ra màn hình, ví dụ như tôi có thể dùng <asp:gridview>:</asp:gridview>

```
Dim northwind As New NorthwindDataContext
GridView1.DataSource = northwind.CustOrderHist("ALFKI")
GridView1.DataBind()
```

Khi đó danh sách các sản phẩm được mua bởi khách hàng sẽ được hiển thị như sau:

The screenshot shows a Microsoft Internet Explorer window with the title "Untitled Page - Windows Internet Explorer". The address bar displays "http://localhost:542". The main content area contains a table with two columns: "ProductName" and "Total". The table lists ten products with their respective total counts. The table has a blue header row and white rows for the data. The data rows are as follows:

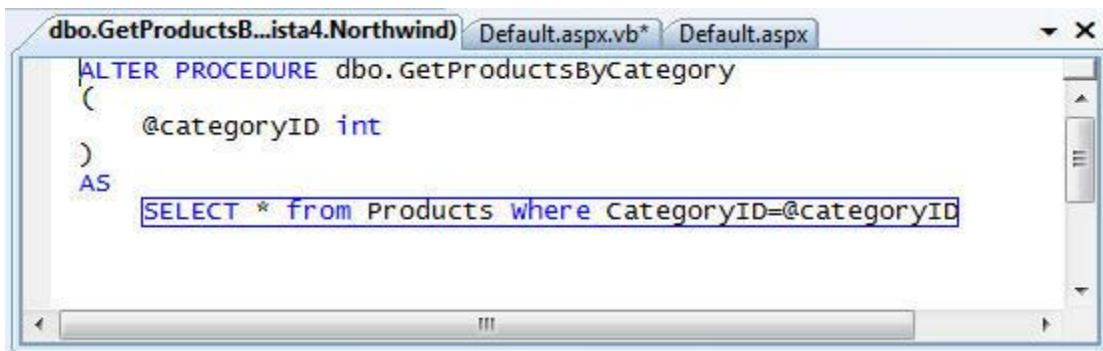
ProductName	Total
Aniseed Syrup	6
Chartreuse verte	21
Escargots de Bourgogne	40
Flotemysost	20
Grandma's Boysenberry Spread	16
Lakkalikööri	15
Original Frankfurter grüne Soße	2
Raclette Courdavault	15
Rössle Sauerkraut	17
Spegesild	2
Vegie-spread	20

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

Trong thủ tục CustOrderHist ở trên, thủ tục trả về một danh sách dữ liệu bao gồm 2 cột: ProductName chứa tên và TotalNumber chứa số sản phẩm đã được đặt hàng trong quá khứ. LINQ to SQL designer sẽ tự động tạo ra một lớp có tên CustOrderHistResult 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 Product hay Order.

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

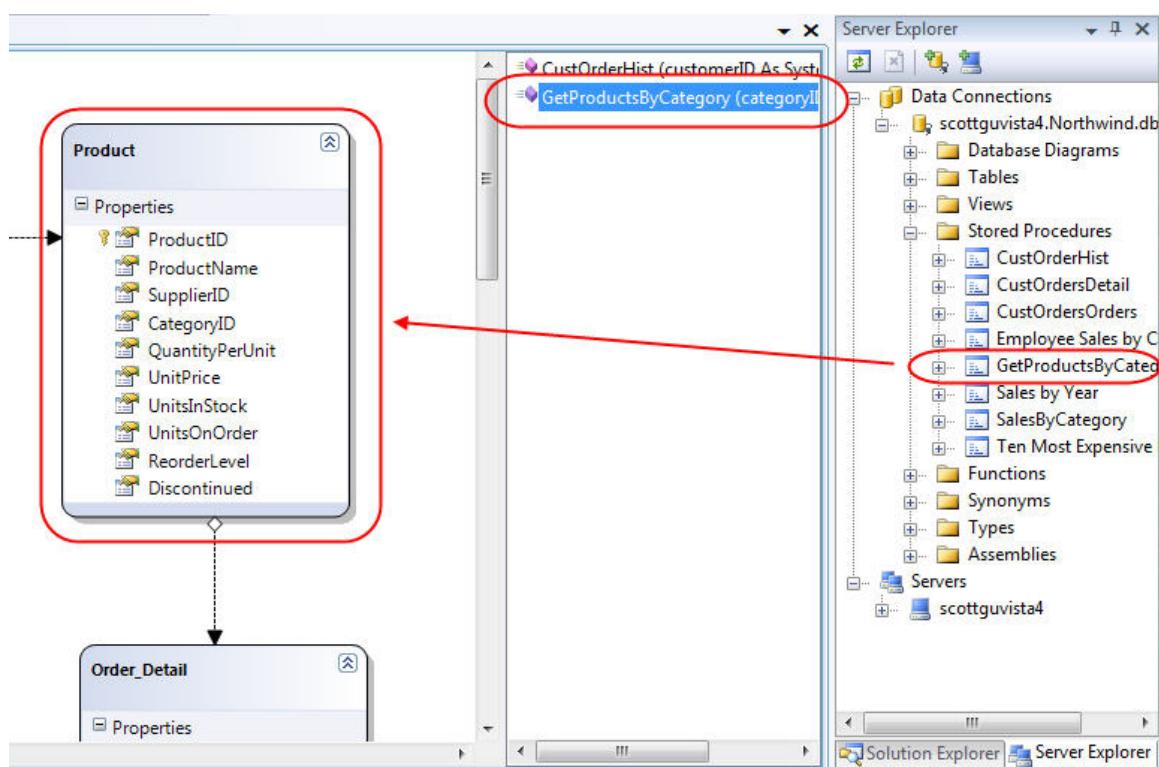


```

ALTER PROCEDURE dbo.GetProductsByCategory
(
    @categoryID int
)
AS
    SELECT * FROM Products WHERE CategoryID = @categoryID

```

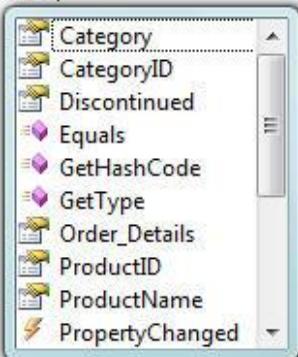
Cũng như trước đây, ta có thể tạo một phương thức GetProductsByCategory ở bên trong lớp DataContext mà nó sẽ gọi thủ tục này bằng cách kéo nó vào cửa sổ LINQ to SQL designer. Thay vì thả nó vào một vị trí bất kỳ, chúng ta sẽ thả nó lên trên lớp Product mà ta đã tạo ra sẵn trên sửa sổ này:



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

```

NorthwindDataContext northwind = new NorthwindDataContext();
List<Product> products = northwind.GetProductsByCategory(1).ToList();
foreach (Product product in products)
{
    product.|
```



The screenshot shows an IntelliSense dropdown menu for a 'Product' object. The visible properties are: Category, CategoryID, Discontinued, Equals, GetHashCode, GetType, Order_Details, ProductID, ProductName, and PropertyChanged.

Một ưu điểm của việc sử dụng lớp Product như kiểu 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 được 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, bạn có thể viết đoạn code giống như dưới đây (dùng một SPROC) và thay đổi giá của các sản phẩm bên trong một Category nào đó thành 90% giá trị cũ:

```

NorthwindDataContext northwind = new NorthwindDataContext();

// Retrieve all products from a category and reset their price
// to be 90% of what it currently is

List<Product> products = northwind.GetProductsByCategory(1).ToList();
foreach (Product product in products)
{
    product.UnitPrice = product.UnitPrice * (Decimal) .9;
}
northwind.SubmitChanges();
```

Khi gọi SubmitChanges, nó sẽ cập nhật lại giá của tất cả các sản phẩm. Để hiểu thêm về cách quản lý các thay đổi và cách phương thức

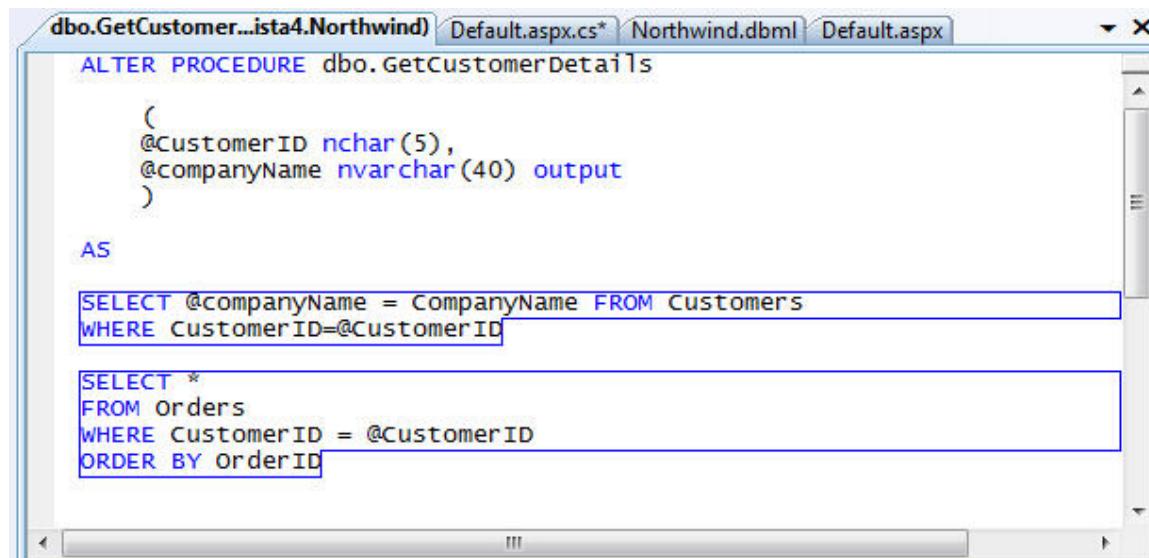
SubmitChanges() làm việc, cũng như các thêm các phương thức xác thực logic dữ liệu, xin mời đọc lại bài 4 trong cùng loạt bài này.

Trong bài viết tiếp theo của loạt bài về LINQ to SQL, tôi sẽ hướng dẫn các bạn cách thay thế các câu lệnh SQL động cho việc INSERT/UPDATE/DELETE bằng các thủ tục SPROC. Và khi thay thế như vậy, bạn hoàn toàn không phải thay đổi gì trên các đoạn lệnh trên – việc thay đổi này hoàn toàn xảy ra trên mô hình dữ liệu và hoàn toàn trong suốt với các chương trình dùng nó.

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).</nullable>

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



```
ALTER PROCEDURE dbo.GetCustomerDetails
(
    @CustomerID nchar(5),
    @CompanyName nvarchar(40) output
)
AS
SELECT @CompanyName = CompanyName FROM Customers
WHERE CustomerID=@CustomerID
SELECT *
FROM Orders
WHERE CustomerID = @CustomerID
ORDER BY OrderID
```

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

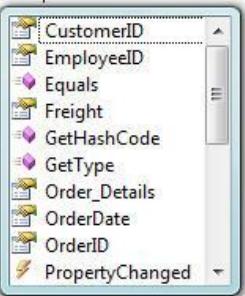
```

NorthwindDataContext northwind = new NorthwindDataContext();

// Lookup customer name and order history

string companyName = "";
List<Order> orders = northwind.GetCustomerDetails("ALFKI", ref companyName).ToList();

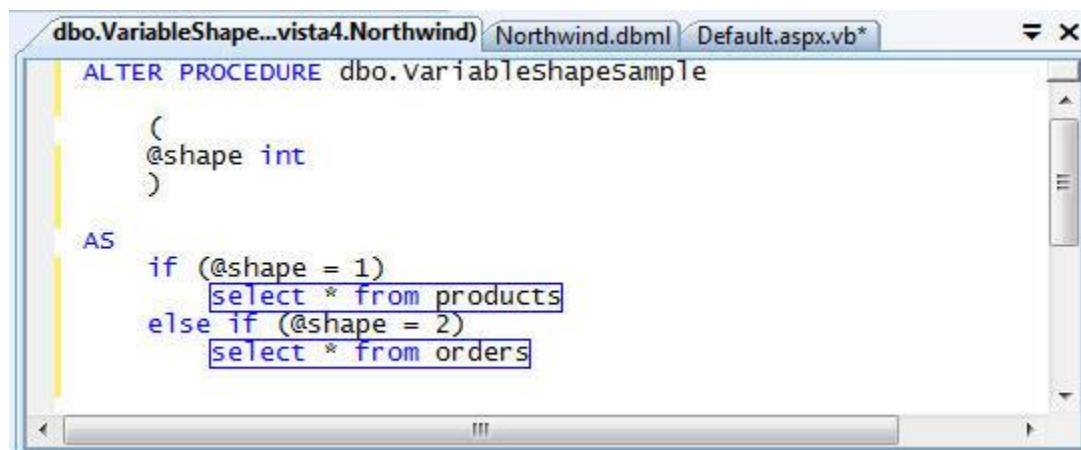
foreach (Order order in orders)
{
    order.|
```



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

Xử lý các thủ tục trả về nhiều kiểu kết quả khác nhau

Khi một thủ tục trả về nhiều kiểu kết quả khác nhau, kiểu 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 sản phẩm hay lệnh đặt hàng tùy thuộc vào tham số đầu vào:



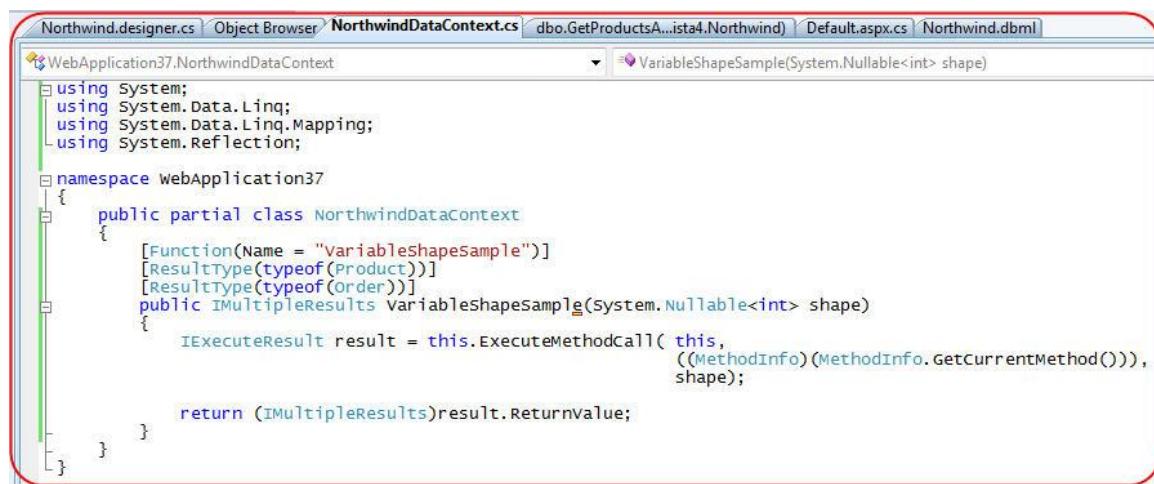
```

dbo.VariableShape...vista4.Northwind] Northwind.dbml Default.aspx.vb*
ALTER PROCEDURE dbo.variableShapeSample
(
    @shape int
)

AS
    if (@shape = 1)
        select * from products
    else if (@shape = 2)
        select * from orders
```

LINQ to SQL hỗ trợ việc tạo các phương thức trợ giúp cho phép trả về Product hay Order bằng cách thêm một lớp partial NorthwindDataContext vào dự án và định nghĩa một phương thức trong lớp này (trong ví dụ này

chúng ta gọi là VariablesShapeSample) để gọi thủ tục và trả về một đối tượng có kiểu IMultipleResult như trong ví dụ sau:



The screenshot shows the Visual Studio Object Browser with the following details:

- Project: Northwind
- Namespace: WebApplication37
- Class: NorthwindDataContext
- Method: VariableShapeSample (highlighted in blue)
- Description: VariableShapeSample(System.Nullable<int> shape)
- Implementation:

```
using System;
using System.Data.Linq;
using System.Data.Linq.Mapping;
using System.Reflection;

namespace WebApplication37
{
    public partial class NorthwindDataContext
    {
        [Function(Name = "VariableShapeSample")]
        [ResultType(typeof(Product))]
        [ResultType(typeof(Order))]
        public IMultipleResults VariableShapeSample(System.Nullable<int> shape)
        {
            IExecuteResult result = this.ExecuteMethodCall(this,
                ((MethodInfo)(MethodInfo.GetCurrentMethod())),
                shape);

            return (IMultipleResults)result.ReturnValue;
        }
    }
}
```

Một khi đã thêm phương thức này vào dự án, bạn có thể gọi và chuyển về kiểu thích hợp là Product hoặc Order:

```
NorthwindDataContext northwind = new NorthwindDataContext();

// call SPROC and retrieve Products
//

IMultipleResults result = northwind.VariableShapeSample(1);

var products = result.GetResult<Product>();

foreach (Product product in products)
{
    Response.Write(product.ProductName + "<br>");
}

// call SPROC and retrieve Orders
//

IMultipleResults result2 = northwind.VariableShapeSample(2);

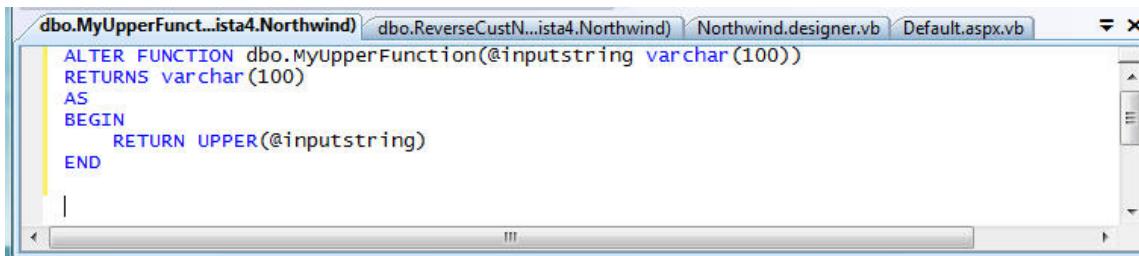
var orders = result2.GetResult<Order>();

foreach (Order order in orders)
{
    Response.Write("OrderID: " + order.OrderID + "<br>");
}
```

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 được thêm vào lớp DataContext như một phương thức, bạn có thể dùng các hàm UDF này trong câu trong các câu lệnh LINQ.

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



```
dbo.MyUpperFunc...ista4.Northwind  dbo.ReverseCustN...ista4.Northwind  Northwind.designer.vb  Default.aspx.vb
ALTER FUNCTION dbo.MyUpperFunction(@inputstring varchar(100))
RETURNS varchar(100)
AS
BEGIN
    RETURN UPPER(@inputstring)
END
```

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

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

```
NorthwindDataContext northwind = new NorthwindDataContext();
var products = from p in northwind.Products
               where northwind.MyUpperFunction(p.Category.CategoryName) == "BEVERAGES"
               select new
               {
                   p.ProductID,
                   p.ProductName,
                   p.UnitPrice
               };
```

Nếu bạn dùng LINQ to SQL Debug Visualizer mà tôi đã viết tại đây, bạn có thể thấy các LINQ to SQL chuyển đổi câu truy vấn ở trên thành câu lệnh SQL để thực thi hàm UDF khi chạy:

The screenshot shows the SQL Server Query Visualizer window. The title bar says "SQL Server Query Visualizer". The main area displays a generated SQL query:

```
Table(Product).Where(p => (value(WebApplication37._Default+>c_DisplayClass1).northwind.MyUpperFunction(p.Category.CategoryName) = "BEVERAGES")).Select(p => new <>f_AnonymousType0'3(ProductID = p.ProductID, ProductName = p.ProductName, UnitPrice = p.UnitPrice))
```

Below the query, the generated SQL code is shown:

```
SELECT [t0].[ProductID], [t0].[ProductName], [t0].[UnitPrice]
FROM [dbo].[Products] AS [t0]
LEFT OUTER JOIN [dbo].[Categories] AS [t1] ON [t1].[CategoryID] = [t0].[CategoryID]
WHERE [dbo].[MyUpperFunction]([t1].[CategoryName]) = 'BEVERAGES'
```

At the bottom left is a "Original query" button, and at the bottom right is an "Execute" button.

Tổng kết

LINQ to SQL supports the ability to call Stored Procedures and UDFs within the database and nicely integrate them into our data model. In this blog post I demonstrated how you can use SPROCs to easily retrieve data and populate our data model classes. In my next blog post in this series I'll cover how you can also use SPROCs to override the update/insert/delete logic when you SubmitChanges() on your DataContext to persist back to the database.

LINQ to SQL hỗ trợ khả năng gọi các thủ tục và hàm trong CSDL và có khả năng tích hợp dễ dàng vào trong mô hình dữ liệu. Trong bài viết này tôi đã trình diễn cách dùng các thủ tục SPROC để dễ dàng truy xuất cũng như cập nhật các lớp mô hình dữ liệu. Trong bài kế tiếp tôi sẽ biểu diễn cách dùng SPROC để thực hiện việc cập nhật/thêm/xóa khi gọi SubmitChanges để cập nhật lại dữ liệu vào CSDL.

Cập nhật dữ liệu dùng Stored Procedure (LINQ to SQL phần 7)

Vài tuần trước tôi bắt đầu viết loạt bài về LINQ to SQL. LINQ to SQL là một bộ khung (framework) có sẵn cho O/RM (object relational mapping) trong .NET 3.5, nó cho phép bạn dễ dàng mô hình hóa các CSDL quan hệ dùng các lớp .NET. Bạn có thể dùng các biểu thức LINQ để truy vấn CSDL, cũng như có thể cập nhật/thêm/xóa dữ liệu từ đó.

Dưới đây là 6 phần đầu tiên của loạt bài này:

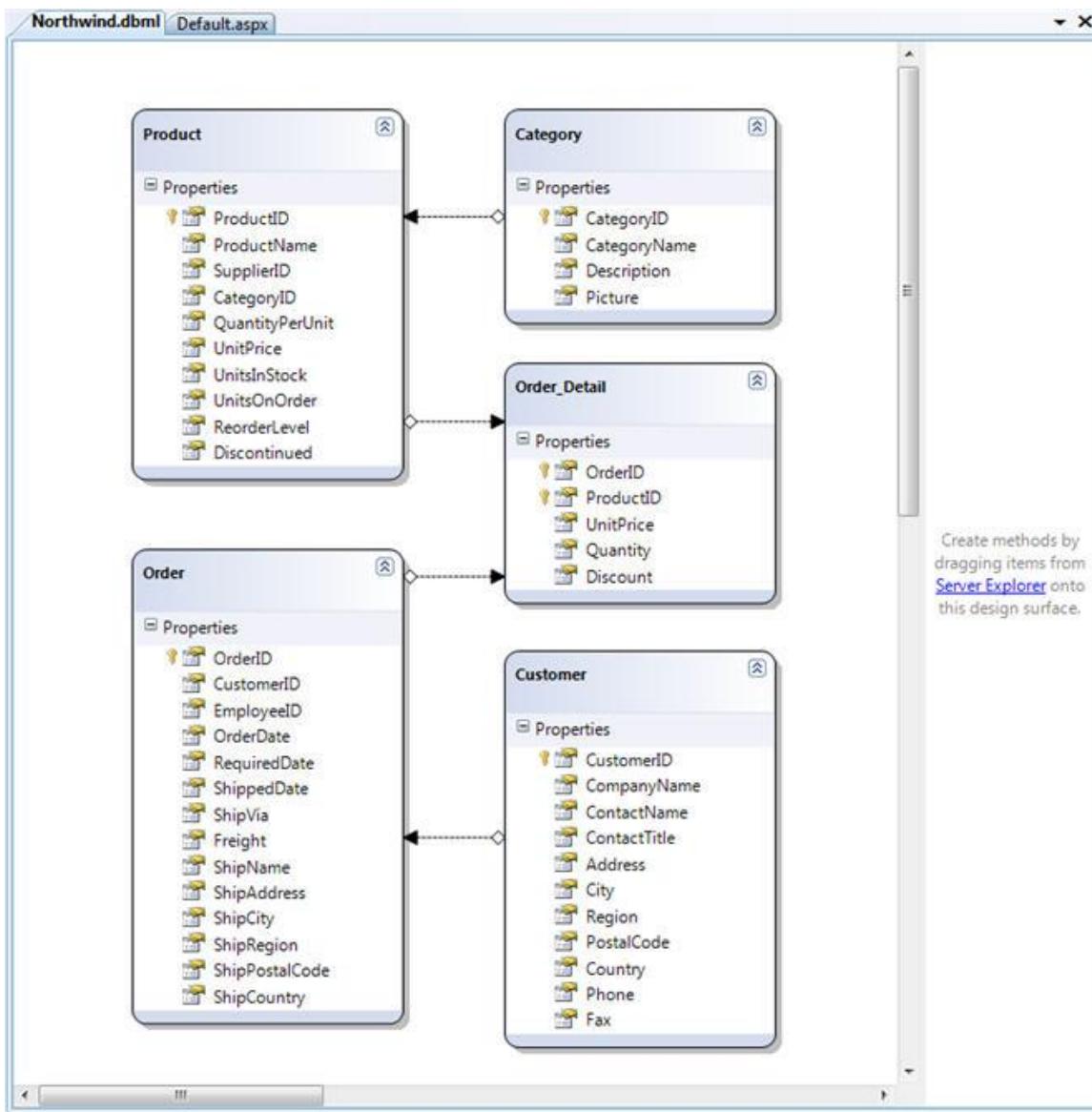
- Sử dụng LINQ to SQL (phần 1)
- Định nghĩa các lớp mô hình dữ liệu (phần 2)
- Truy vấn Cơ sở dữ liệu (phần 3)
- Cập nhật cơ sở dữ liệu (LINQ to SQL phần 4)
- Sử dụng asp:LinqDataSource (phần 5)
- Lấy dữ liệu dùng Stored Procedure (LINQ to SQL phần 6)
- Cập nhật dữ liệu dùng Stored Procedure (LINQ to SQL phần 7)
- Thực thi các biểu thức SQL tùy biến (LINQ to SQL phần 8)

Trong phần 6 tôi đã nói tới cách chúng ta có thể dùng các Stored Procedure (SPROC) và các hàm do người dùng định nghĩa (UDF) để truy vấn và lấy dữ liệu về dùng mô hình dữ liệu LINQ to SQL. Trong viết này, tôi sẽ nói về cách dùng các thủ tục này để cập nhật, thêm hoặc xóa dữ liệu.

Để có thể minh họa cho điều này, chúng ta hãy bắt đầu từ đầu và xây dựng một lớp truy xuất dữ liệu cho CSDL mẫu Northwind:

Bước 1: Tạo lớp truy xuất dữ liệu (chưa dùng đến các thủ tục)

Trong phần 2, tôi có nói về cách dùng LINQ to SQL designer có trong VS 2008 để tạo một mô hình lớp giống như dưới đây:



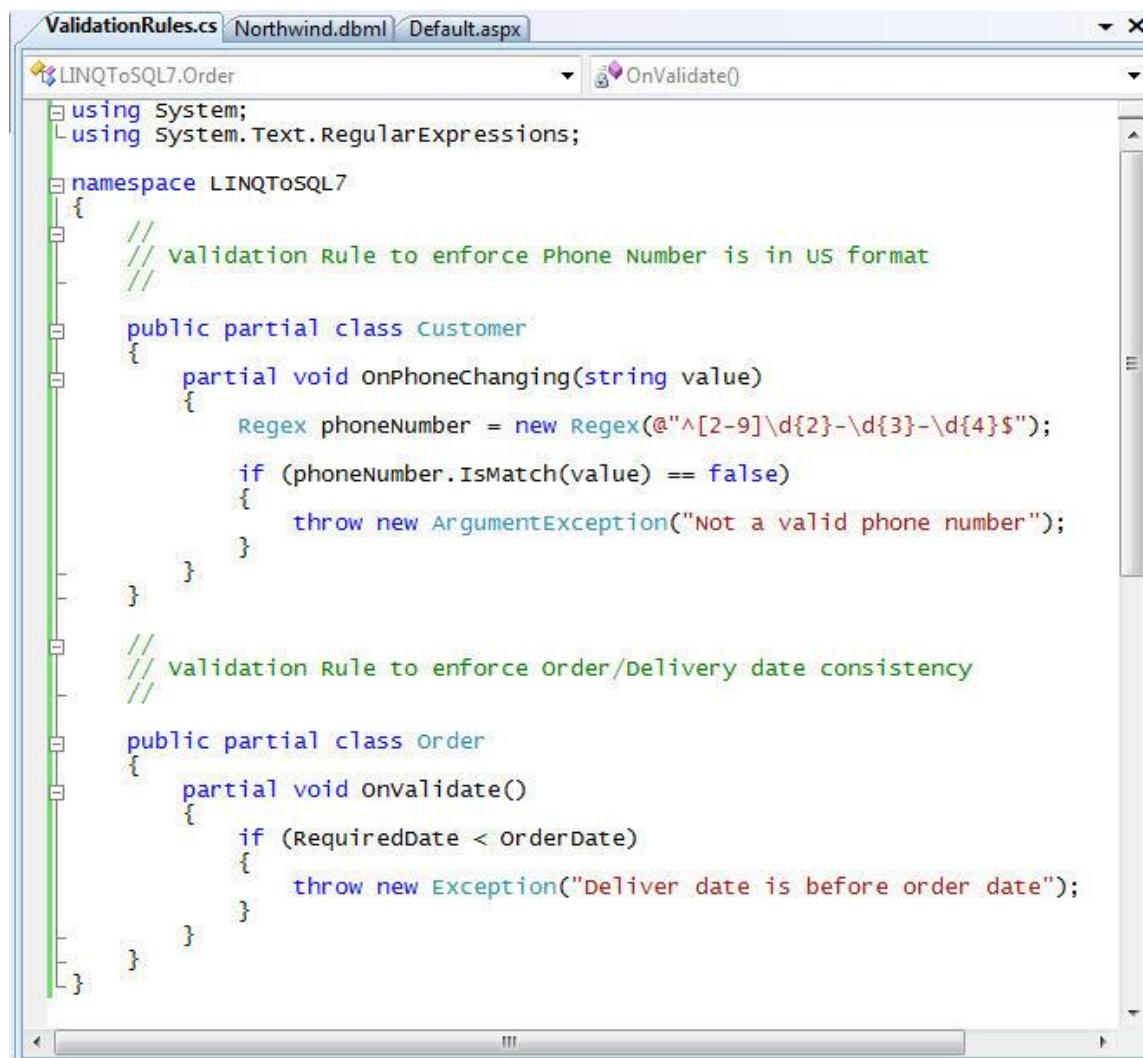
Thêm các quy tắc kiểm tra dữ liệu vào các lớp mô hình dữ liệu

Sau khi định nghĩa các lớp trong mô hình dữ liệu và các quan hệ giữa chúng, chúng ta sẽ tiếp tục thêm vào các quy tắc kiểm tra tính hợp lệ của dữ liệu.

Chúng ta có thể làm điều này bằng cách thêm các lớp partial vào trong dự án và thêm các quy tắc kiểm tra vào các lớp mô hình dữ liệu (tôi đã nói đến vấn đề này khá kỹ trong bài 4).

Ví dụ, bạn có thể thêm một quy tắc để đảm bảo rằng số điện thoại của khách hàng được nhập đúng định dạng, và chúng ta không cho phép thêm một đơn hàng (Order) nếu trường OrderDate lớn hơn RequiredDate. 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.



```
ValidationRules.cs Northwind.dbml Default.aspx LINQToSQL7.Order OnValidate()

using System;
using System.Text.RegularExpressions;

namespace LINQTOSQL7
{
    // validation Rule to enforce Phone Number is in us format

    public partial class Customer
    {
        partial void OnPhoneChanging(string value)
        {
            Regex phoneNumber = new Regex(@"^([2-9]\d{2}-\d{3}-\d{4}$");
            if (phoneNumber.IsMatch(value) == false)
            {
                throw new ArgumentException("Not a valid phone number");
            }
        }

        // validation Rule to enforce order/delivery date consistency

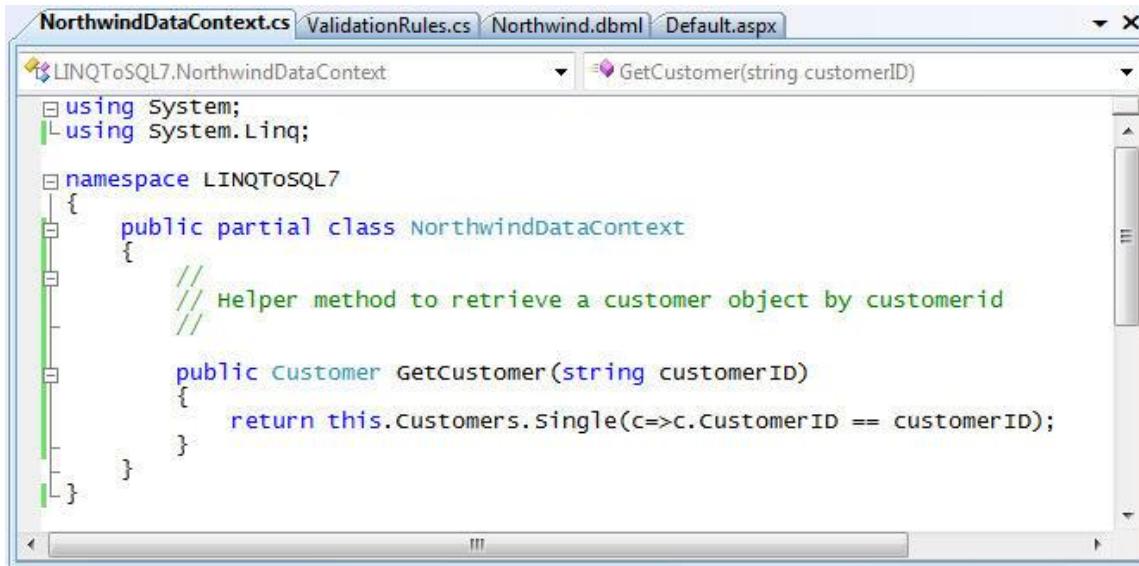
        public partial class Order
        {
            partial void Onvalidate()
            {
                if (RequiredDate < orderDate)
                {
                    throw new Exception("Deliver date is before order date");
                }
            }
        }
    }
}
```

Thêm phương thức GetCustomer() vào lớp DataContext

Hiện tại chúng ta 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 (tôi đã có nói về điều này trong bài 3). Thêm nữa tôi cũng có thể ánh xạ các SPROC vào lớp DataContext và dùng chúng để đưa dữ liệu vào CSDL (bài 6).

Khi xây dựng các lớp dữ liệu LINQ to SQL, bạn sẽ thường có nhu cầu đưa các câu lệnh LINQ thường dùng vào các phương thức tiện ích trong lớp

DataContext. Bạn có thể làm được điều này bằng cách thêm một lớp partial vào project. Ví dụ, banks có thể thêm một phương thức có tên “GetCustomer()” cho phép chúng ta tìm kiếm và lấy về các đối tượng Customer từ CSDL dựa trên giá trị của CustomerID:



The screenshot shows a code editor window with the tab 'NorthwindDataContext.cs' selected. The code defines a partial class 'NorthwindDataContext' within the namespace 'LINQToSQL7'. It contains a single public method 'GetCustomer(string customerID)' which returns a 'Customer' object by its ID. The code is annotated with comments explaining the purpose of the helper method.

```
using System;
using System.Linq;

namespace LINQToSQL7
{
    public partial class NorthwindDataContext
    {
        // Helper method to retrieve a customer object by customerid
        public Customer GetCustomer(string customerID)
        {
            return this.Customers.Single(c=>c.CustomerID == customerID);
        }
    }
}
```

Bước 2: 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 thể thực hiện truy vấn, cập nhật, thêm và xóa dữ liệu.

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

```
NorthwindDataContext northwind = new NorthwindDataContext();
// Retrieve Customer
Customer myCustomer = northwind.GetCustomer("ALFKI");
// Update the Contact Name and Phone Number
myCustomer.ContactName = "scott Guthrie";
myCustomer.Phone = "425-555-1212";
// Create New Order for the Customer
Order order = new Order();
order.OrderDate = DateTime.Now;
order.RequiredDate = DateTime.Now.AddDays(3);
order.ShipCity = "Redmond";
order.ShipPostalCode = "98052";
// Associate order with Customer
myCustomer.orders.Add(order);
// save all updates to the database
northwind.SubmitChanges();
```

LINQ to SQL theo dõi các thay đổi 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 Customer ở trên, và thêm một bản ghi mới vào bảng Orders.

Chờ một giây – Tôi nghĩ bài viết này định nói về việc dùng SPROC cơ mà ???

Nếu vẫn đang đọc bài này, bạn có lẽ sẽ cảm thấy khó hiểu vì không thấy nói gì về SPROC. Tại sao tôi hướng dẫn bạn cách viết lệnh để làm việc với các đối tượng trong mô hình dữ liệu, rồi cho phép các câu lệnh SQL động được thực thi? Sao tôi vẫn chưa cho các bạn thấy cách để gọi các SPROC để thực hiện việc chèm/sửa/xóa dữ liệu ?

Lý do là vì mô hình lập trình của LINQ to SQL để làm việc với các đối tượng mô hình dữ liệu bằng SPROC cũng hoàn toàn tương tự với việc sử dụng các câu lệnh 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 chuyển sang dùng SPROC). Đoạn lệnh ở trên để lấy

về một Customer, rồi cập nhật và thêm một Order sẽ hoàn toàn giống nhau, không phụ thuộc vào việc chúng ta dùng các câu SQL động hay các SPROC để thực hiện việc truy cập vào CSDL.

Mô hình lập trình này rất mạnh mẽ theo cả hai nghĩa: nó không bắt bạn phải học hai cách dùng khác nhau, và bạn cũng không cần phải quyết định ngay từ đầu là dùng SPROC hay không. Ban đầu, bạn 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 bạn quyết định. Các đoạn lệnh và các đoạn test bạn đã viết trước đây sẽ vẫn được sử dụng tiếp, không phụ thuộc vào việc dùng SQL hay SPROC.

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

1) Dùng LINQ to SQL designer để cấu hình các SPROC để thực thi khi gấp thao tác thêm/xóa/sửa dữ liệu trên các lớp mô hình dữ liệu.

hoặc:

2) Thêm một lớp partial NorthwindDataContext vào dự án, 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 để lưu đối tượng đó vào CSDL.

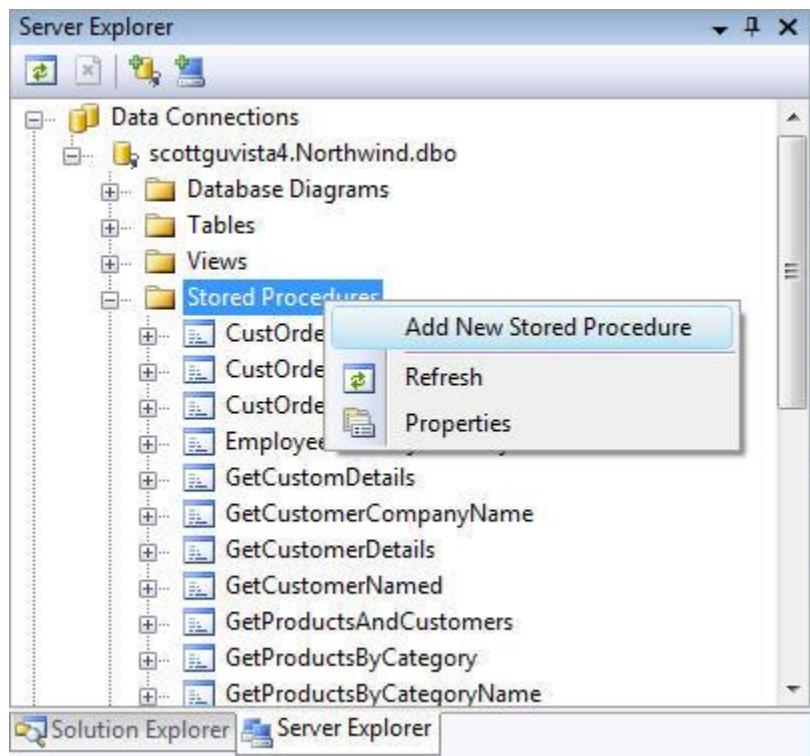
Khi dùng cách 1) (dùng LINQ to SQL designer) để cấu hình các SPROC để gọi, thì thực ra nó cũng sẽ tạo ra cách lệnh tương tự như chúng ta dùng trong

cách 2). Nói chúng tôi khuyên các bạn dùng LINQ to SQL designer để cấu hình các SPROC trong 90% trường hợp – và chỉ trong các trường hợp nào bạn cần tùy biến lại cách gọi ở một mức độ cao, bạn mới nên viết các lệnh một cách trực tiếp.

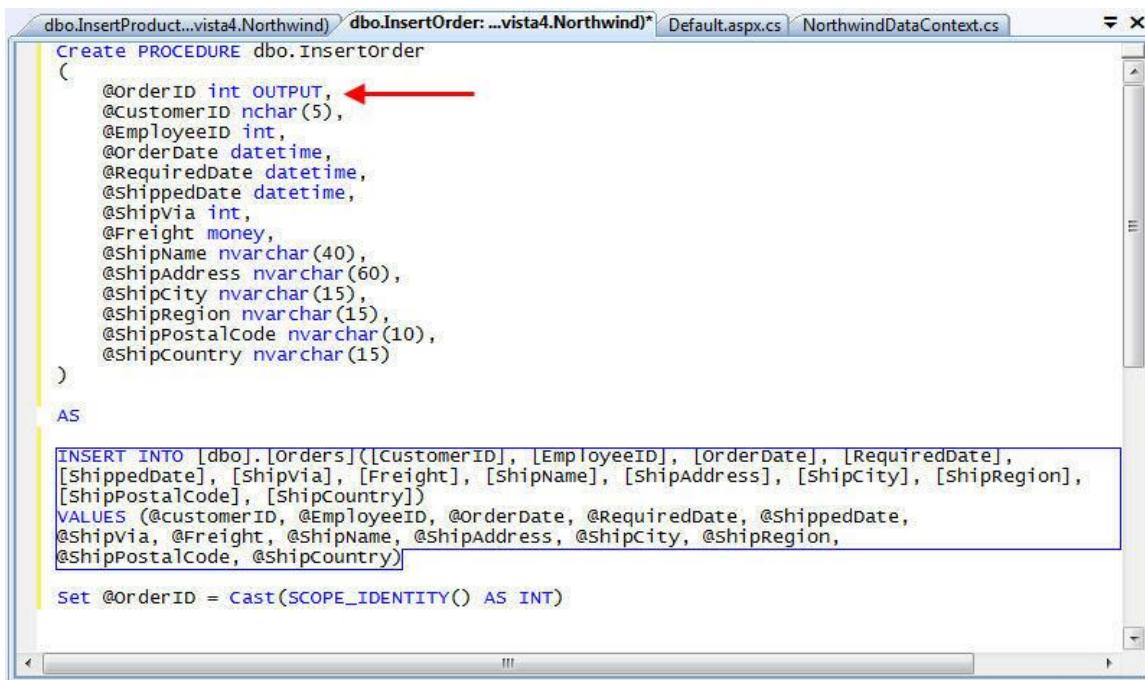
Bước 3: Thêm một Order bằng cách dùng SPROC

Chúng ta sẽ bắt đầu chuyển mô hình dữ liệu sang dùng SPROC, bắt đầu từ đối tượng Object.

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



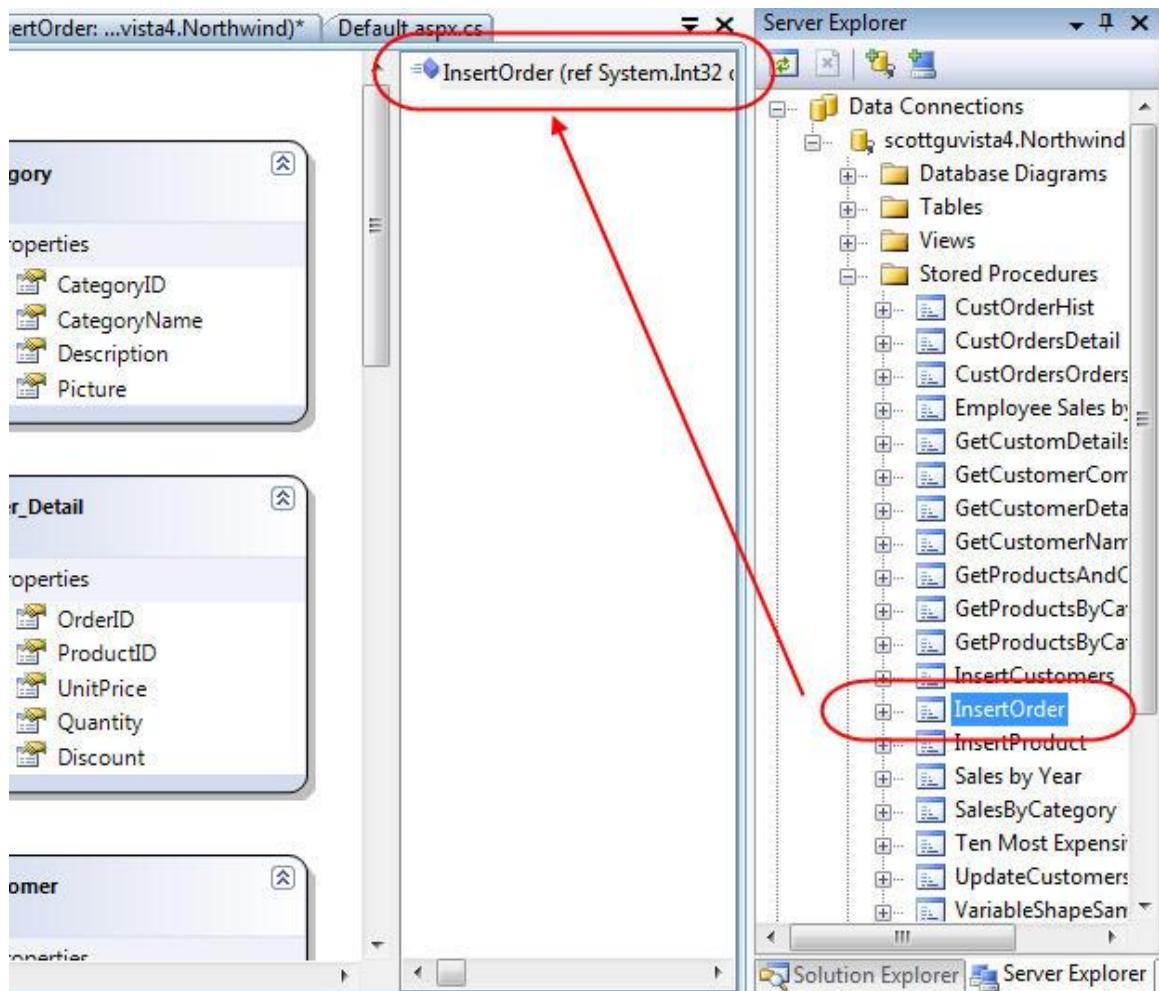
Sau đó ta tạo thêm một thủ tục có tên “InsertOrder” có nhiệm vụ chèn thêm một bản ghi mới vào bảng Orders:



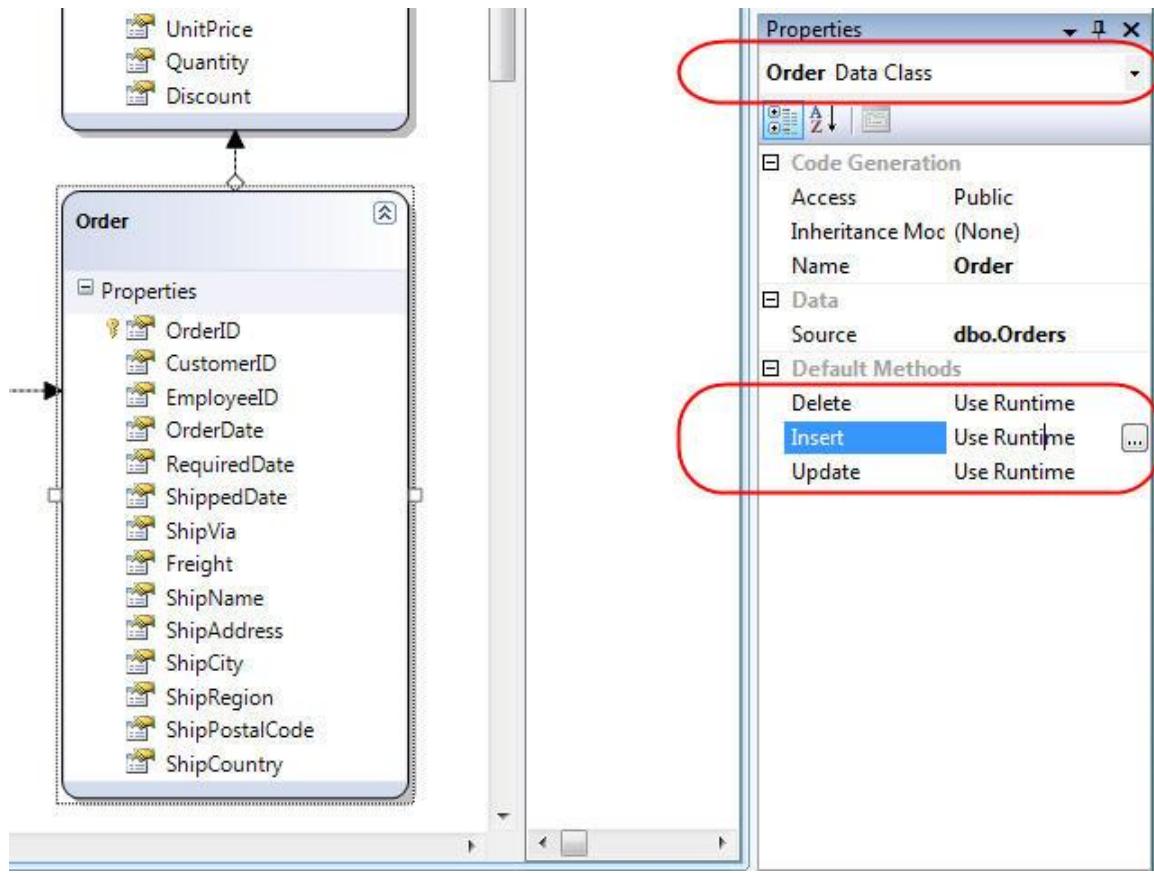
```
CREATE PROCEDURE dbo.InsertOrder
(
    @OrderID int OUTPUT, ←
    @CustomerID nchar(5),
    @EmployeeID int,
    @OrderDate datetime,
    @RequiredDate datetime,
    @ShippedDate datetime,
    @ShipVia int,
    @Freight money,
    @ShipName nvarchar(40),
    @ShipAddress nvarchar(60),
    @ShipCity nvarchar(15),
    @ShipRegion nvarchar(15),
    @ShipPostalCode nvarchar(10),
    @ShipCountry nvarchar(15)
)
AS
    INSERT INTO [dbo].[Orders]([CustomerID], [EmployeeID], [OrderDate], [RequiredDate],
    [ShippedDate], [ShipVia], [Freight], [ShipName], [ShipAddress], [ShipCity], [ShipRegion],
    [ShipPostalCode], [ShipCountry])
    VALUES (@CustomerID, @EmployeeID, @OrderDate, @RequiredDate, @ShippedDate,
    @ShipVia, @Freight, @ShipName, @ShipAddress, @ShipCity, @ShipRegion,
    @ShipPostalCode, @ShipCountry)
    set @OrderID = cast(SCOPE_IDENTITY() AS INT)
```

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

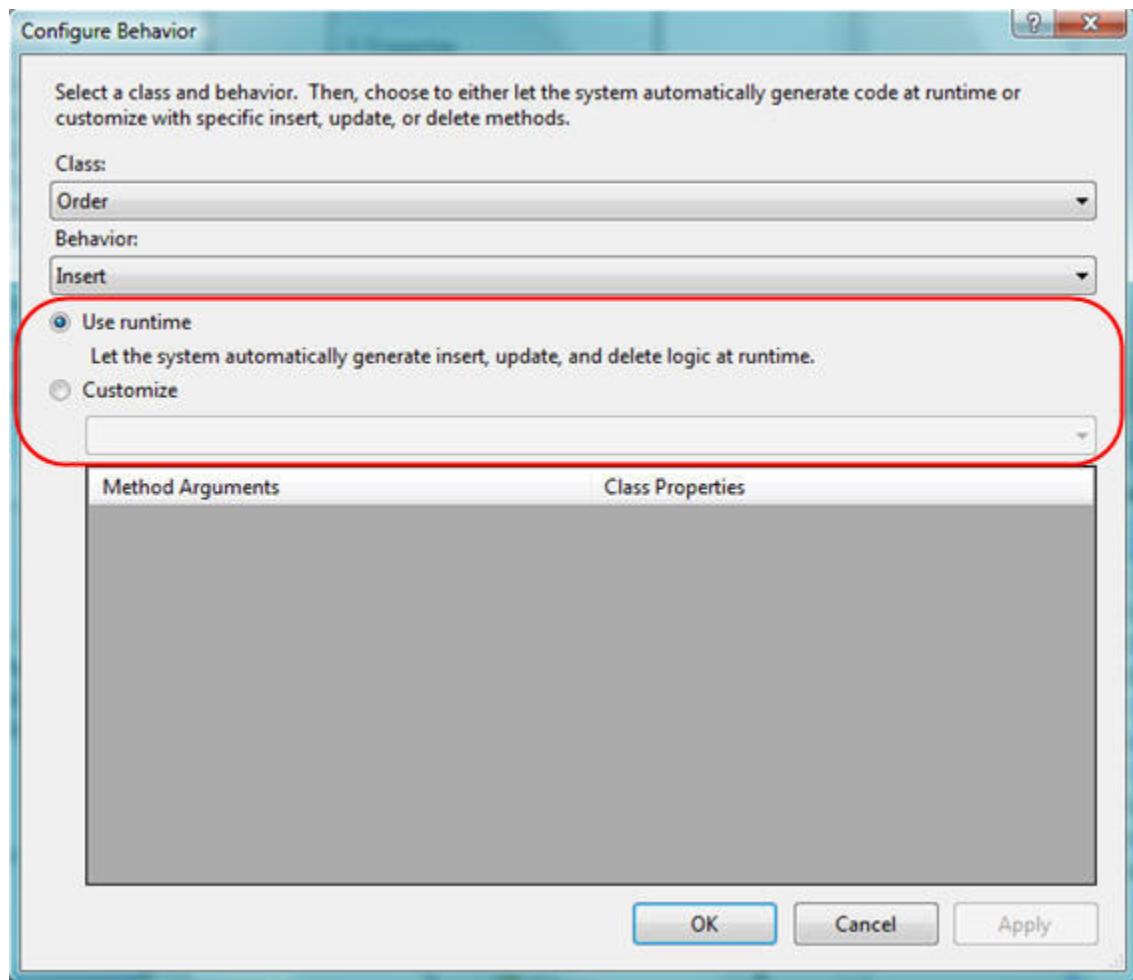
Sau khi tạo ra SPROC, chúng ta sẽ mở LINQ to SQL designer của lớp truy cập dữ liệu. Như tôi đã nói trong bài 6, chúng ta có thể kéo/thả các SPROC từ Server Explorer lên trên màn hình chính của trình thiết kế. Chúng ta cũng sẽ làm điều tương tự với thủ tục InsertOrder vừa được tạo:



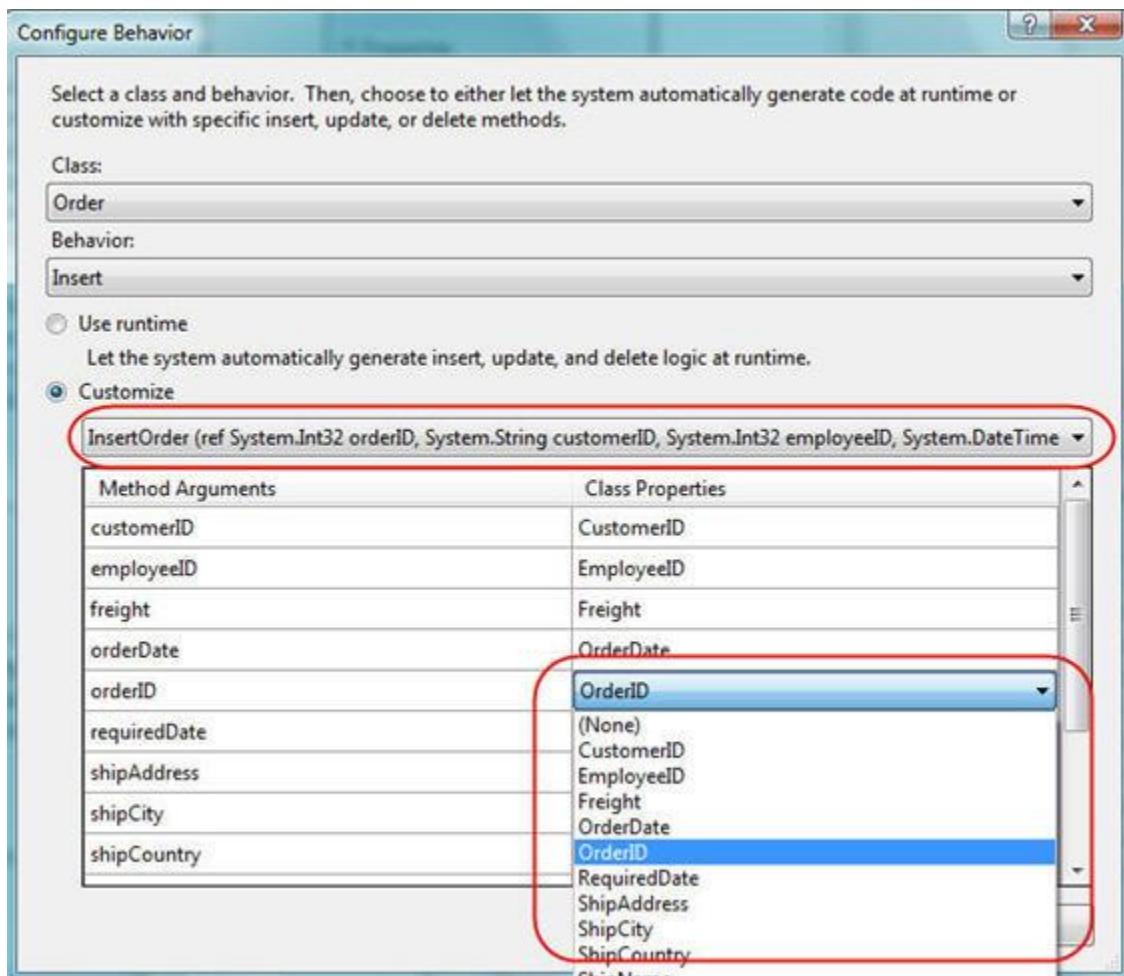
Bước cuối cùng là cấu hình lại lớp truy cập dữ liệu dùng thủ tục SPROC khi chèn các đối tượng Order mới vào trong CSDL. Chúng ta có thể là điều này bằng cách chọn lớp Order trong cửa sổ LINQ to SQL designer, và sau đó chuyển đến bảng thuộc tính và nhấn nút 3 chấm (...) ở mục Insert để chọn thao tác tương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



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



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



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

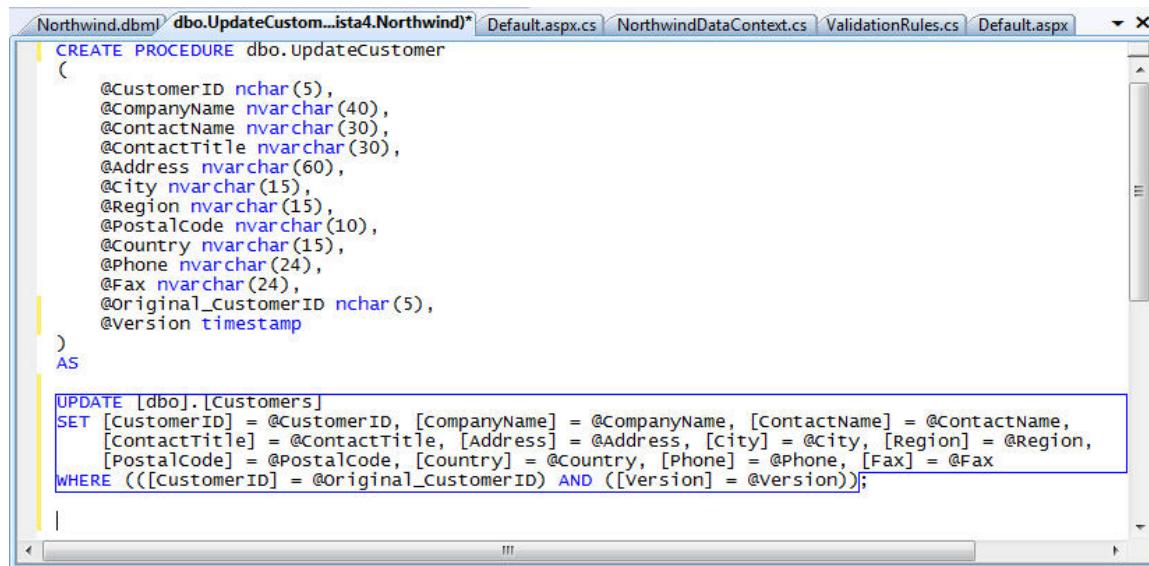
Nhấn vào nút Ok là xong. Giờ đây bất cứ khi nào một đối tượng Order được thêm vào DataContext và phương thức SubmitChanges() được gọi, thủ tục InsertOrder sẽ được thực thi thay cho câu lệnh SQL động.

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

Bước 4: Thực hiện cập nhật dùng SPROC

Giờ chúng ta sẽ sửa lại đối tượng Customer để cho phép cập nhật bằng cách dùng SPROC.

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

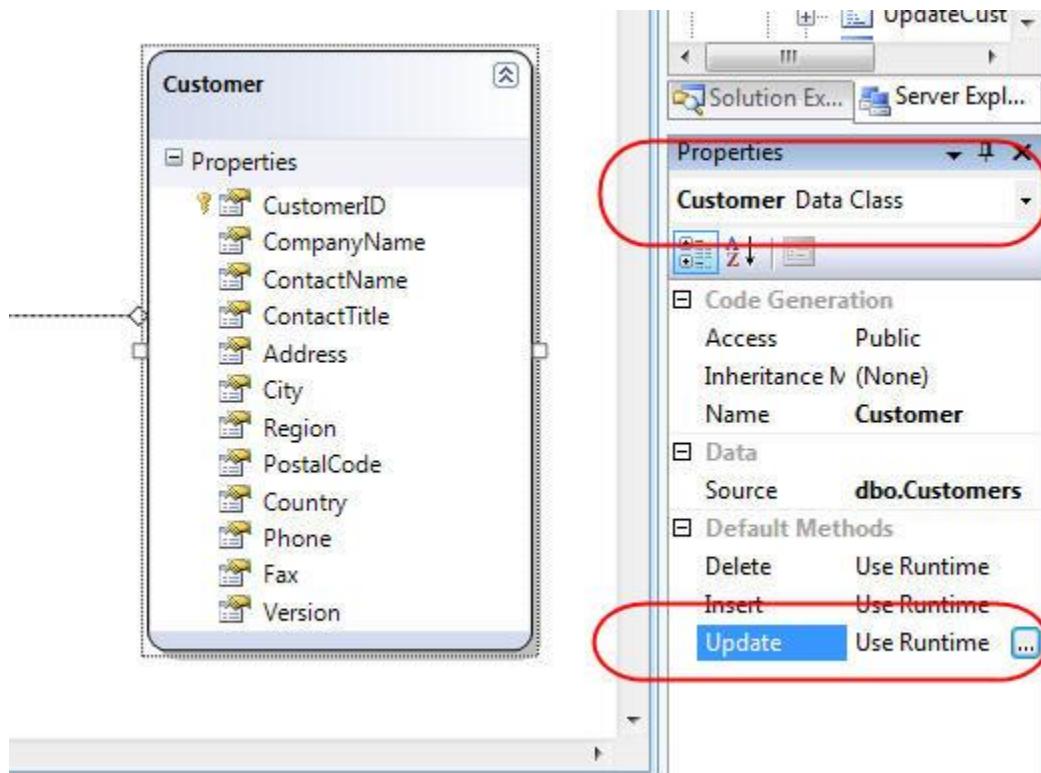


```
CREATE PROCEDURE dbo.UpdateCustomer
(
    @CustomerID nchar(5),
    @CompanyName nvarchar(40),
    @ContactName nvarchar(30),
    @ContactTitle nvarchar(30),
    @Address nvarchar(60),
    @City nvarchar(15),
    @Region nvarchar(15),
    @PostalCode nvarchar(10),
    @Country nvarchar(15),
    @Phone nvarchar(24),
    @Fax nvarchar(24),
    @Original_CustomerID nchar(5),
    @Version timestamp
)
AS
UPDATE [dbo].[Customers]
SET [CustomerID] = @CustomerID, [CompanyName] = @CompanyName, [ContactName] = @ContactName,
    [ContactTitle] = @ContactTitle, [Address] = @Address, [City] = @City, [Region] = @Region,
    [PostalCode] = @PostalCode, [Country] = @Country, [Phone] = @Phone, [Fax] = @Fax
WHERE (([CustomerID] = @Original_CustomerID) AND ([version] = @Version));
```

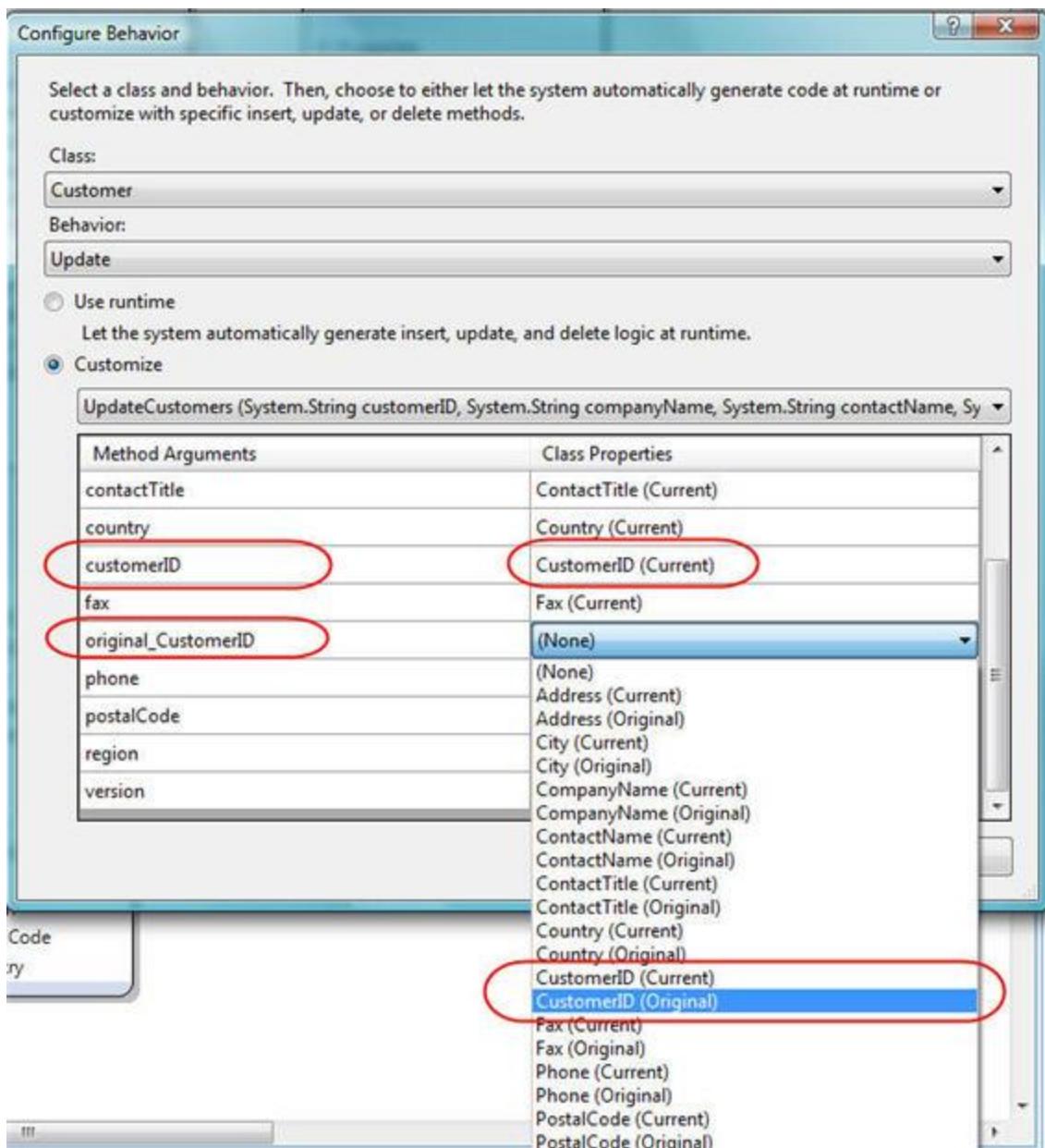
Chú ý ở trên, ngoài việc truyền giá trị cho tham số CustomerID, tôi cũng truyền một tham số khác có tên @Original_CustomerID. Cột CustomerID trong bảng Customers không phải là một cột tự tăng, và nó có thể được chỉnh sửa như một phần của thao tác cập nhật. Do vậy chúng ta sẽ phải truyền cả giá trị của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

Bạn sẽ thấy ở trên tôi đã truyền một tham số có tên @Version (có kiểu timestamp) vào cho SPROC. Đây là một cột tôi đã thêm vào bảng Customers để có thể xử lý việc tranh chấp khi các thao tác cập nhật được diễn ra đồng thời (optimistic concurrency). Tôi sẽ nói chi tiết hơn về việc xử lý tranh chấp này trong bài viết sau của loạt bài LINQ to SQL, nhưng tôi cũng nói luôn là LINQ to SQL hỗ trợ đầy đủ optimistic concurrency, và cho phép bạn có thể chọn dùng version timestamp hay bằng cách cung cấp cả giá trị cũ/mới cho SPROC để có thể xác định được các thay đổi được tạo ra bởi người khác kể từ lần cuối bạn đọc dữ liệu. Trong ví dụ này tôi dùng timestamp vì nó giúp viết lệnh rõ ràng hơn.

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



Chúng ta sẽ chọn ô “Customize” và chọn để dùng UpdateCustomer:



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

Khi nhấn “Ok” trên ửa sổ này, bạn đã hoàn thành việc ánh xạ các tham số vào các thuộc tính. Từ giờ trở đi, mỗi khi cập nhật lại giá trị cho đối tượng

Customer và gọi SubmitChanges(), thủ tục UpdateCustomer sẽ được gọi thay cho câu lệnh SQL động.

Quan trọng: Dù rằng hiện tại bạn đã dùng SPROC để cập nhật, phương thức “OnPhoneChanging()” mà chúng ta đã tạo trước đó (trong bước 1 của bài này) để xác thực số điện thoại vẫn được thực thi trước khi bất kỳ thay đổi nào được lưu lại hay “UpdateCustomer” được gọi. Chúng ta có một cách rõ ràng, sáng sửa để hiện thực hóa cá quy tắc xử lý cũng như xác thực dữ liệu, và có thể dùng chúng mà không phụ thuộc và việc chúng ta đang dùng câu lệnh SQL động hay SPROC.

Bước 5: Dùng lớp DAL lần nữa

Một khi đã cập nhật lớp truy cập dữ liệu (DAL) để dùng SPROC thay vì câu lệnh SQL động, bạn 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:

```
Dim northwind As New NorthwindDataContext  
    ' Retrieve Customer  
  
    Dim myCustomer As Customer = northwind.GetCustomer("ALFKI")  
        ' Update the Contact Name and Phone Number  
  
        myCustomer.ContactName = "Scott Guthrie"  
        myCustomer.Phone = "425-555-1212"  
        ' Create New Order for the Customer  
  
        Dim order As New Order  
        order.OrderDate = Now()  
        order.RequiredDate = Now().AddDays(3)  
        order.ShipCity = "Redmond"  
        order.ShipPostalCode = "98052"  
        ' Associate Order with Customer  
  
        myCustomer.Orders.Add(order)  
        ' Save all updates to the database  
        northwind.SubmitChanges()
```

Giờ đây việc cập nhật đối tượng Customer, và việc thêm 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 hết như trước đây, và các câu

lệnh chúng ta đã dùng để sử dụng các lớp mô hình dữ liệu cũng hoàn toàn tương tự.

Một số ưu điểm của việc dùng SPROC

Sau đây là một vài ý nhỏ có thể có ích cho bạn trong việc dùng SPROC:

Dùng các tham số dạng output:

Trong phần 3 ở trên, tôi đã biểu diễn cách chúng ta có thể trả về giá trị OrderID mới được tạo (đây là một cột tự tăng trong CSDL) bằng cách dùng một tham số dạng output. Bạn sẽ không bị giới hạn trong việc trả về chỉ các cột tự tăng – mà thật sự bạn có thể trả về các giá trị cho bất kỳ tham số nào của SPROC. Bạn 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 để lấy các giá trị đã được tạo ra.

Sẽ thế nào nếu một SPROC phát ra một lỗi?

Nếu một SPROC phát ra một lỗi khi thực hiện việc Insert/Update/Delete, LINQ 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 bạn sẽ luôn trong trạng thái đúng đắn.

Tôi có thể viết code thay vì dùng ORM designer để gọi SPROC?

Như đã nói trong phần đầu bài viết này, bạn có thể dùng LINQ to SQL designer để ánh xạ các thao tác thêm/sửa/xóa vào các SPROC, hoặc bạn 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 NorthwindDataContext dùng UpdateCustomer để gọi một thủ tục:

```
partial void UpdateCustomer(Customer customer)
{
    Customer original = ((Customer)(customers.GetOriginalEntityState(customer)));
    this.UpdateCustomers(customer.CustomerID, customer.CompanyName, customer.ContactName, customer.ContactTitle,
}

[Function(Name="dbo.updateCustomers")]
public int UpdateCustomers([Parameter](Name="CustomerID", DbType="NChar(5)") string customerID, [Parameter](Name=
{
    IExecuteResult result = this.ExecuteMethodCall(this, ((MethodInfo)(MethodInfo.GetCurrentMethod())), customer
    return ((int)(result.ReturnValue));
}
```

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

Tổng kết

LINQ to SQL là một trình ánh xạ đối tượng (ORM) cực kỳ mềm dẻo. Nó cho phép bạn viết các đoạn code theo kiểu hướng đối tượng một cách rõ ràng, sang sửa dễ lấy, cập nhật hay thêm dữ liệu.

Hơn hết, nó cho phép bạn thiết kế các lớp mô hình dữ liệu một cách dễ dàng, không phụ thuộc vào cách nó được lưu hay nạp lại từ CSDL. Bạn có thể dùng trình ORM xây dựng sẵn để lấy về hay cập nhật dữ liệu một cách hiệu quả bằng cách dùng các câu SQL động. Hoặc bạn cũng có thể cấu hình lớp dữ liệu để dùng SPROC. Điều hay là các đoạn lệnh của bạn để dùng lớp dữ liệu này, cũng như các thủ tục để kiểm tra logic đều không phụ thuộc vào cách lưu/nạp dữ liệu thực sự được dùng.

Trong bài tiếp theo của loạt bài này, tôi sẽ nói về một số khái niệm còn lại trong LINQ to SQL, bao gồm: Single Table Inheritance, Deferred/Eager Loading, Optimistic Concurrency, và xử lý trong các ngữ cảnh Multi-Tier.

Thực thi các biểu thức SQL tùy biến (LINQ to SQL phần 8)

Vài tuần trước tôi bắt đầu viết loạt bài về LINQ to SQL. LINQ to SQL là một bộ khung (framework) có sẵn cho O/RM (object relational mapping) trong .NET 3.5, nó cho phép bạn dễ dàng mô hình hóa các CSDL quan hệ dùng các lớp .NET. Bạn có thể dùng các biểu thức LINQ để truy vấn CSDL, cũng như có thể cập nhật/thêm/xóa dữ liệu từ đó.

Dưới đây là 7 phần đầu tiên của loạt bài này:

Sử dụng LINQ to SQL (phần 1)

Định nghĩa các lớp mô hình dữ liệu (phần 2)

Truy vấn Cơ sở dữ liệu (phần 3)

Cập nhật cơ sở dữ liệu (LINQ to SQL phần 4)

Sử dụng asp:LinqDataSource (phần 5)

Lấy dữ liệu dùng Stored Procedure (LINQ to SQL phần 6)

Cập nhật dữ liệu dùng Stored Procedure (LINQ to SQL phần 7)

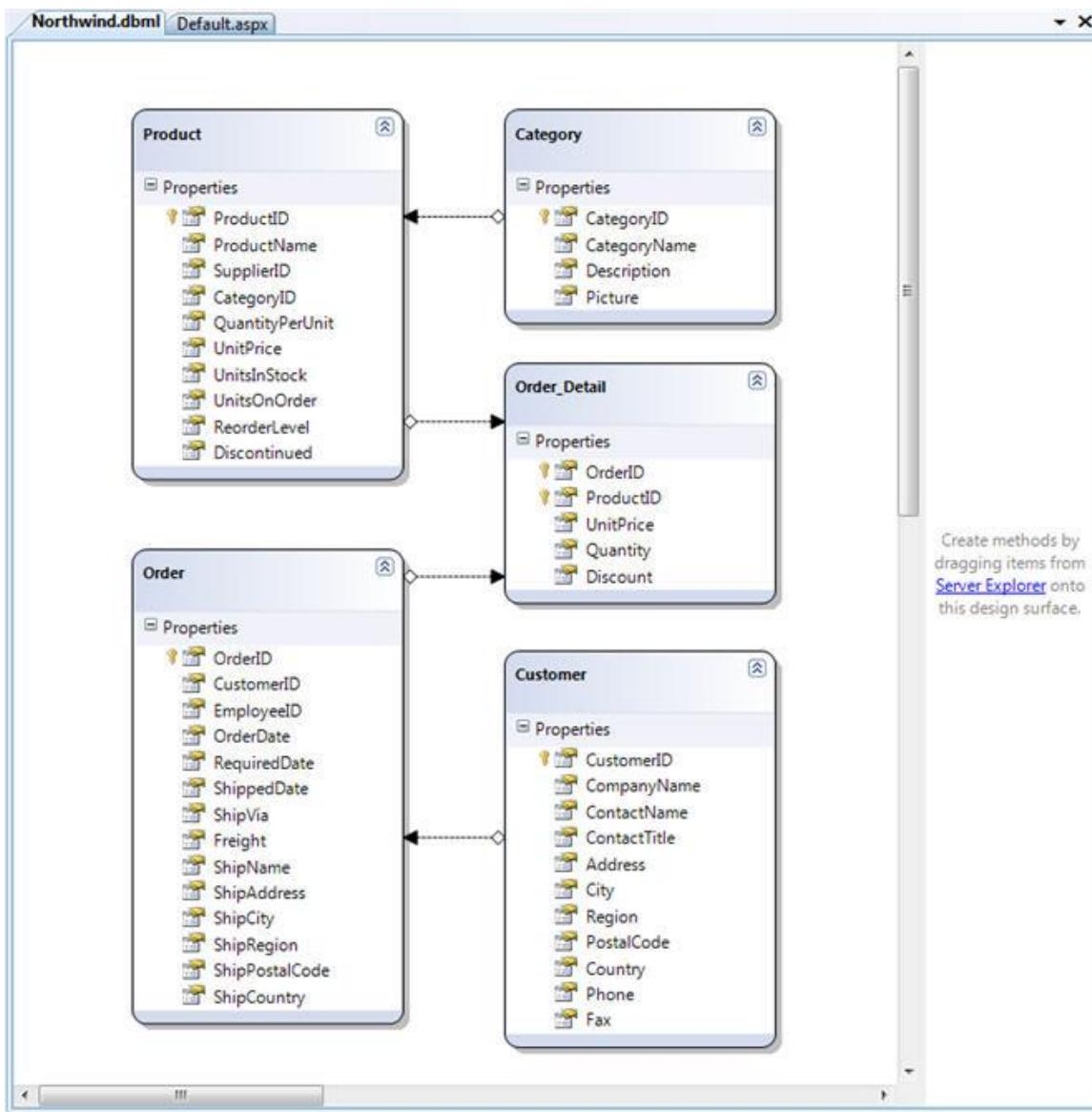
Thực thi các biểu thức SQL tùy biến (LINQ to SQL phần 8)

Trong hai bài cuối (bài 6 và bài 7), tôi đã biểu diễn cách bạn có thể dùng các thủ tục trong CSDL (SPROC) để thực hiện truy vấn, cập nhật, thêm hoặc xóa dữ liệu dùng mô hình dữ liệu LINQ to SQL.

Có một vài bạn đã hỏi tôi khi viết các bài này là “Liệu tôi có thể kiểm soát hoàn toàn các câu SQL được dùng bởi LINQ to SQL mà không cần phải viết các SPROC?”. Trong bài viết này tôi sẽ nói về điều này – và thảo luận cách bạn có thể viết các câu SQL tùy biến để truy vấn, cũng như để thêm, sửa hay xóa dữ liệu.

Dùng các biểu thức truy vấn LINQ với LINQ to SQL

Trong bài viết này, chúng ta sẽ dùng mô hình mô hình dữ liệu được tạo với CSDL Northwind (xin hãy đọc phần 2 để học cách dùng VS 2008 để tạo ra mô hình này):



Trong phần 3, tôi đã cho các bạn thấy cách dùng ngôn ngữ LINQ mới được đưa vào VB và C# 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, bạn có thể thêm một phương thức trợ giúp “GetProductsByCategory” vào lớp `DataContext` trong mô hình dữ liệu của chúng ta mà nó sẽ dùng một câu truy vấn LINQ để trả về các đối tượng `Product` từ CSDL:

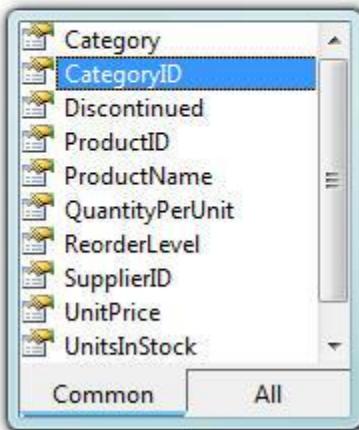
The screenshot shows a code editor window with the tab 'NorthwindDataContext.cs*' selected. The code is written in C# and defines a partial class 'NorthwindDataContext' with a method 'GetProductsByCategory'. The method uses LINQ to query the 'Products' collection where the 'CategoryID' matches the input parameter.

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

namespace LINQToSQL8
{
    public partial class NorthwindDataContext
    {
        public IEnumerable<Product> GetProductsByCategory(int categoryID)
        {
            return from p in this.Products
                   where p.CategoryID == categoryID
                   select p;
        }
    }
}
```

Một khi bạn đã định nghĩa phương thức LINQ như trên, bạn có thể viết lệnh giống như dưới đây để dùng nó lấy về các sản phẩm, và duyệt qua tập kết quả trả về:

```
Dim northwind As New NorthwindDataContext
Dim products = northwind.GetProductsByCategory(1)
For Each product In products
    product.
Next
```



Khi biểu thức LINQ bên trong phương thức “GetProductsByCategory” được thực thi, trình quản lý LINQ to SQL sẽ tự động thực thi câu SQL động để lấy về dữ liệu Product và tạo ra danh sách các đối tượng Product. Bạn có thể dùng trình debug để xem cách biểu thức LINQ này thực thi.

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 Product. Thay vì vậy, LINQ to SQL sẽ tự động dịch biểu thức LINQ thành câu lệnh SQL chúng ta 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? Một cách để làm điều này là dùng một SPROC giống như tôi đã trình bày trong bài 6 và bài 7. Một cách khác là dùng phương thức “ExecuteQuery” trong lớp DataContext để thực thi một câu SQL do chúng ta cung cấp.

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, bạn có thể thực thi bất kỳ câu lệnh SQL bạn muốn với CSDL (kể cả các câu lệnh JOIN nhiều bảng).

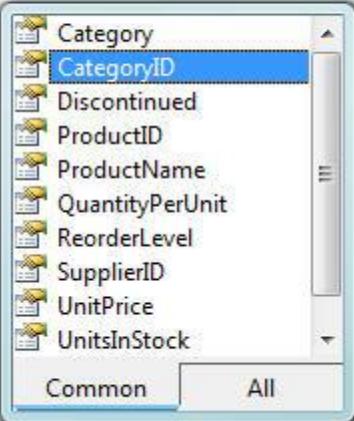
Điều làm cho ExecuteQuery thực sự hữu dụng là nó cho phép bạn chỉ ra cách nó trả về dữ liệu. Bạn có thể làm được điều này bằng cách truyền một đối tượng có kiểu mong muốn như một tham số của phương thức, hay dùng kiểu generic.

Ví dụ, bạn có thể thay đổi phương thức GetProductsByCategory() được tạo ra trước đây – phiên bản dùng một biểu thức LINQ – để dùng phương thức ExecuteQuery thực thi một câu SQL với CSDL và trả về một tập đối tượng Product như kết quả:

```
public partial class NorthwindDataContext
{
    public IEnumerable<Product> GetProductsByCategory(int categoryId)
    {
        return ExecuteQuery<Product>(@"select * from products where categoryid={0}", categoryId);
    }
}
```

Chúng ta có thể gọi GetProductsByCategory() dùng cùng cách như trước đây:

```
Dim northwind As New NorthwindDataContext  
  
Dim products = northwind.GetProductsByCategory(1)  
  
For Each product In products  
    product.  
Next
```



Nhưng không như trước đây, trong trường hợp này câu SQL tùy biến sẽ được gọi thay cho câu SQL động được tạo bởi biểu thức LINQ.

Tùy biến các biểu thức SQL và theo vết (tracking) các thao tác cập nhật:

Mặc nhiên, khi bạn lấy về một mô hình dữ liệu dùng LINQ to SQL, nó sẽ lưu lại các thay đổi mà bạn làm. Nếu gọi phương thức “SubmitChanges()” trên lớp DataContext, nó sẽ lưu lại các thay đổi vào CSDL. Tôi đã nói chi tiết về vấn đề này trong phần 4 của loạt bài này.

Một trong những tính năng nổi trội của ExecuteQuery là nó có thể kết hợp hoàn toàn vào quá trình theo vết và cập nhật lại mô hình dữ liệu. Ví dụ, bạn có thể viết đoạn lệnh dưới đây để lấy về tất cả các sản phẩm từ một chủng loại nào đó và giảm giá toàn bộ 10%:

```
Dim northwind As New NorthwindDataContext  
  
Dim products = northwind.GetProductsByCategory(1)  
  
For Each product In products  
    product.UnitPrice = product.UnitPrice * 0.9  
Next  
  
northwind.SubmitChanges()
```

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

Tùy biến các biểu thức SQL với các lớp của bạn

Phương thức ExecuteQuery() cho phép bạn chỉ ra bất kỳ lớp nào như kiểu trả về của câu truy vấn. Lớp này không nhất thiết phải được tạo ra bởi trình LINQ to SQL designer, hay phải thừa kế từ bất kỳ class/interface nào.

Ví dụ, bạn có thể định nghĩa một lớp ProductSummary mới chứa các thuộc tính là tập con của Product như dưới đây (chú ý là chúng ta dùng đặc tính Automatic Properties mới có trong C#):

```
public class ProductSummary
{
    public int ProductID { get; set; }
    public string ProductName { get; set; }
    public Decimal? UnitPrice { get; set; }
}
```

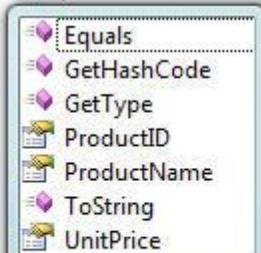
Chúng ta có thể sau đó tạo ra một phương thức tên là GetProductSummariesByCategory() trong lớp NorthwindDataContext, nó sẽ trả về các kết quả dựa trên kiểu ProductSummary. Để ý là câu SQL dưới đây chỉ yêu cầu các thuộc tính của Product nó cần – ExecuteQuery sẽ tự biết cách đưa các giá trị đó vào các đối tượng ProductSummary mà nó sẽ trả về.

```
public partial class NorthwindDataContext
{
    public IEnumerable<ProductSummary> GetProductSummariesByCategory(int categoryId)
    {
        return ExecuteQuery<ProductSummary>(@"select ProductId, ProductName, UnitPrice
                                         from products where categoryid={0}", categoryId);
    }
}
```

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

```

NorthwindDataContext db = new NorthwindDataContext();
var products = db.GetProductSummariesByCategory(1);
foreach(ProductSummary p in products)
{
    p.|
```



Tùy biến các câu SQL cho Inserts/Updates Deletes

Thêm vào việc dùng các biểu thức SQL tùy biến để truy vấn, bạn cũng có thể dùng chúng để thực hiện các thao tác như thêm/xóa/sửa.

Chúng ta có thể 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 Product, bạn có thể định nghĩa một phương thức partial DeleteProduct như sau:

```

public partial class NorthwindDataContext
{
    partial void DeleteProduct(Product instance)
    {
        ExecuteCommand("DELETE FROM Products WHERE ProductId={0}", instance.ProductID);
    }
}
```

Và bây giờ, nếu bạn viết đoạn code dưới đây để xóa một Product nào đó khỏi CSDL, LINQ to SQL sẽ gọi phương thức DeleteProduct – 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:

```

NorthwindDataContext db = new NorthwindDataContext();
// Retrieve a specific product
Product product = db.Products.Single(p=>p.ProductName == "My Product");
// Mark it to be deleted
db.Products.Remove(product);
// update database
db.SubmitChanges();
```

Tổng kết

Trình quản lý LINQ to SQL tự động tạo ra và thực thi các câu SQL động để thực hiện các câu truy vấn, cập nhật, thêm và xóa dữ liệu trong CSDL.

Đối với một số trường hợp, khi bạn muốn kiểm soát hoàn toàn câu lệnh SQL được thực thi, bạn có thể dùng các thủ tục SPROC, hay cũng có thể viết các câu SQL của riêng bạn. Điều này cung cấp khả năng tùy biến mạnh mẽ khi xây dựng các lớp truy cập dữ liệu.

Trong bài viết tiếp theo, tôi sẽ nói đến một số khái niệm còn lại như: Single Table Inheritance, Deferred/Eager Loading, Optimistic Concurrency, và sử dụng trong mô hình Multi-Tier.

Dùng biểu thức LINQ tùy biến với <asp:LinqDataSource> (LINQ to SQL phần 9)

Vài tuần trước tôi bắt đầu viết loạt bài về LINQ to SQL. LINQ to SQL là một bộ khung (framework) có sẵn cho O/RM (object relational mapping) trong .NET 3.5, nó cho phép bạn dễ dàng mô hình hóa các CSDL quan hệ dùng các lớp .NET. Bạn có thể dùng các biểu thức LINQ để truy vấn CSDL, cũng như có thể cập nhật/thêm/xóa dữ liệu từ đó.

Dưới đây là 7 phần đầu tiên của loạt bài này:

Sử dụng LINQ to SQL (phần 1)

Định nghĩa các lớp mô hình dữ liệu (phần 2)

Truy vấn Cơ sở dữ liệu (phần 3)

Cập nhật cơ sở dữ liệu (LINQ to SQL phần 4)

Sử dụng asp:LinqDataSource (phần 5)

Lấy dữ liệu dùng Stored Procedure (LINQ to SQL phần 6)

Cập nhật dữ liệu dùng Stored Procedure (LINQ to SQL phần 7)

Thực thi các biểu thức SQL tùy biến (LINQ to SQL phần 8)

Trong phần 5 của loạt bài này tôi đã giới thiệu control <asp:LinqDataSource> mới trong .NET 3.5 và nói về cách dùng nó để gắn nối các control ASP.NET dễ dàng vào các mô hình dữ liệu LINQ to SQL. Tôi cũng đã trình bày một chút về cách dùng chúng trong một bài viết sau đó khi nói về control <asp:ListView>.

Trong cả hai bài viết trên, các câu truy vấn được thực hiện đều tương đối dễ hiểu (mệnh đề Where làm việc chỉ với một bảng dữ liệu). Trong bài viết hôm nay tôi sẽ biểu diễn cách tận dụng khả năng xây dựng các câu truy vấn nhanh chóng với LINQ dùng LinqDataSource, và cách bạn có thể dùng bất kỳ biểu thức LINQ nào để thực hiện truy vấn với nó.

Tóm tắt: dùng <asp:LinqDataSource> với một mệnh đề where được khai báo

Trong 2 bài viết đó tôi đã biểu diễn cách bạn có thể dùng các bộ lọc có sẵn của LinqDataSource để khai báo nhanh một bộ lọc trên một mô hình dữ liệu LINQ to SQL.

Ví dụ, cho là bạn đã tạo ra một mô hình dữ liệu LINQ to SQL của CSDL Northwind (cách dùng đã được nói đến trong phần 2 của loạt bài này), chúng ta có thể khai báo một control <asp:LinqDataSource> trên trang với một mệnh đề <where> mà nó chỉ trả về các sản phẩm thuộc một chủng loại nào đó (được chỉ ra qua tham số “categoryid” của chuỗi query string):

```
<asp:LinqDataSource ID="ProductDataSource" runat="server" ContextTypeName="NorthwindDataContext"
    TableName="Products" Where="CategoryID == @CategoryID">
    <WhereParameters>
        <asp:QueryStringParameter DefaultValue="1" Name="CategoryID" QueryStringField="categoryid" Type="Int32" />
    </WhereParameters>
</asp:LinqDataSource>
```

Chúng ta có thể trỏ một control <asp:gridview> đến datasource đã tạo và cho phép phân trang, chỉnh sửa và sắp xếp:

```
<asp:GridView ID="Gridview1" DataSourceID="ProductDataSource" DataKeyNames="ProductID" CssClass="gridview" AlternatingRowColor="#D9E1F2" OnRowCommand="Gridview1_RowCommand" OnRowDataBound="Gridview1_RowDataBound">
    <columns>
        <asp:boundfield DataField="ProductID" HeaderText="ProductID" InsertVisible="False" ReadOnly="True" SortExpression="ProductID" />
        <asp:boundfield DataField="ProductName" HeaderText="ProductName" SortExpression="ProductName" />
        <asp:boundfield DataField="UnitPrice" HeaderText="UnitPrice" SortExpression="UnitPrice" />
        <asp:boundfield DataField="UnitsInStock" HeaderText="UnitsInStock" SortExpression="UnitsInStock" />
        <asp:checkboxfield DataField="Discontinued" HeaderText="Discontinued" SortExpression="Discontinued" />
    </columns>
</asp:GridView>

<asp:LinqDataSource ID="ProductDataSource" runat="server" ContextTypeName="NorthwindDataContext" TableName="products" where="CategoryID == @CategoryID" EnableUpdate="true">
    <where parameters>
        <asp:querystringparameter DefaultValue="1" Name="CategoryID" QueryStringField="categoryid" Type="Int32" />
    </where parameters>
</asp:LinqDataSource>
```

Khi chạy trang trên, chúng ta sẽ có một GridView với khả năng tự động sắp xếp, phân trang cũng như chỉnh sửa dữ liệu dữ trên dữ liệu có trong mô hình dữ liệu của chúng ta:

	ProductID	ProductName	UnitPrice	UnitsInStock	Discontinued
Edit	1	Chai	66.0000	55	<input type="checkbox"/>
Update Cancel	24	Guaraná Fantástica	3.2476	20	<input checked="" type="checkbox"/>
Edit	34	Sasquatch Ale	8.8573	111	<input type="checkbox"/>
Edit	35	Steeleye Stout	7.6763	20	<input type="checkbox"/>
Edit	38	Côte de Blaye	155.5940	17	<input type="checkbox"/>
Edit	39	Chartreuse verte	10.6288	69	<input type="checkbox"/>
Edit	43	Ipoh Coffee	1.7714	13	<input type="checkbox"/>
Edit	67	Laughing Lumberjack Lager	8.2668	52	<input type="checkbox"/>
Edit	70	Outback Lager	8.8573	14	<input type="checkbox"/>
Edit	75	Rhönbräu Klosterbier	4.5762	121	<input type="checkbox"/>

hợp. Nhưng sẽ thế nào nếu bạn muốn câu lệnh lọc phức tạp hơn? Ví dụ, sẽ thế nào nếu chúng ta chỉ muốn hiển thị các sản phẩm được tạo bởi các nhà cung cấp dựa trên một tập động các quốc gia?

Dùng các sự kiện Selecting với <asp:LinqDataSource>

Để xử lý các trường hợp trên, bạn có thể tạo các hàm xử lý cho các sự kiện “Selecting” thuộc control <asp:LinqDataSource>. Bên trong các hàm xử lý sự kiện này, bạn có thể viết bất kỳ đoạn lệnh nào bạn muốn để lấy về tập kết quả. Bạn có thể làm được điều này với một câu truy vấn LINQ, gọi một thủ tục SPROC hay dùng một biểu thức SQL tùy biến. Một khi đã lấy dữ liệu về, những gì cần làm là gán nó cho thuộc tính “Result” của đối tượng LinqDataSourceSelectEventArgs. Control <asp:LinqDataSource> khi đó sẽ dùng tập kết quả do bạn trả về để làm việc.

Ví dụ, dưới đây là một câu truy vấn LINQ to SQL để lấy về chỉ các sản phẩm từ các nhà cung cấp thuộc các nước được chọn:

```
protected void ProductDataSource_Selecting(object sender, LinqDataSourceSelectEventArgs e)
{
    NorthwindDataContext northwind = new NorthwindDataContext();
    // List of countries to filter by
    string[] countries = new string[] { "UK", "France", "Germany" };
    // Retrieve those products whose supplier is in one of the countries
    var products = from p in northwind.Products
                   where countries.Contains(p.supplier.country)
                   select p;
    e.Result = products;
}
```

Ghi chú: Bạn không cần viết câu truy vấn ngay bên trong hàm xử lý sự kiện. Một cách tiếp cận sáng sủa hơn là đưa các câu lệnh truy vấn vào trong các hàm trợ giúp, và sau đó gọi lại chúng từ các hàm xử lý sự kiện. Tôi đã dùng cách tiếp cận này trong phần đầu của bài 8 (dùng hàm trợ giúp GetProductsByCategory).

Bây giờ, bằng cách dùng hàm xử lý Selecting, mỗi khi chạy bạn sẽ chỉ thấy các sản phẩm được cung cấp bởi các nhà cung cấp đến từ các quốc gia mà chúng ta đã cho trước.

Products from UK, France, Germany

	<u>ProductID</u>	<u>ProductName</u>	<u>UnitPrice</u>	<u>UnitsInStock</u>	<u>Discontinued</u>
Edit	19	Teatime Chocolate Biscuits	9.7000	25	<input type="checkbox"/>
Edit	28	Rössle Sauerkraut	45.6000	26	<input checked="" type="checkbox"/>
Edit	77	Original Frankfurter grüne Soße	13.0000	32	<input type="checkbox"/>
Edit	20	Sir Rodney's Marmalade	81.0000	40	<input type="checkbox"/>
Edit	27	Schoggi Schokolade	43.9000	49	<input type="checkbox"/>
Edit	58	Escargots de Bourgogne	13.2500	62	<input type="checkbox"/>
Edit	39	Chartreuse verte	10.6288	69	<input type="checkbox"/>
Edit	25	NuNuCa Nuß-Nougat-Creme	14.0000	76	<input type="checkbox"/>
Edit	59	Raclette Courdavault	55.0000	79	<input type="checkbox"/>
Edit	75	Rhönbräu Klosterbier	4.5762	121	<input type="checkbox"/>

[1](#) [2](#)

Một trong những điều thật sự thú vị là các chức năng phân trang và sắp xếp vẫn làm việc với GridView của chúng ta – dù rằng chúng ta đã chuyển sang dùng sự kiện Selecting. Và quan trọng là việc phân trang cũng như sắp xếp này được thực hiện bên trong CSDL – có nghĩa là chúng ta chỉ lấy về 10 sản phẩm từ CSDL mà chúng ta cần để hiển thị cho trang hiện tại trên GridView, điều này giúp việc thực thi hiệu quả hơn rất nhiều.

Bạn có lẽ sẽ tự hỏi – làm sao nó có thể hỗ trợ việc sắp xếp và phân trang hiệu quả như vậy ngay cả khi ta dùng sự kiện Selecting? Lý do là vì câu truy vấn LINQ sẽ không được thực thi tới chừng nào bạn còn chưa lấy kết quả trả về của nó (deferred execution model). Ưu điểm của mô hình này là nó cho phép bạn dễ dàng soạn câu truy vấn trước khi thực thi nó, cũng như dễ dàng đưa thêm các tính năng “add-on” và. Bạn có thể tìm hiểu kỹ hơn trong phần 3 của loạt bài này.

Trong hàm xử lý sự kiện “Selecting” ở trên chúng ta khai báo câu truy vấn LINQ chúng ta muốn thực thi và sau đó gán nó vào thuộc tính e.Query. Mặc dù vậy, câu lệnh LINQ không được thực thi vì chúng ta không lấy kết quả của nó (bằng cách dùng những hàm như ToArray() hay ToList()). LINQDataSource sau đó sẽ có thể thêm mệnh đề order by, và nối thêm các hàm mở rộng Take() và Skip(), nhờ vậy mà tập kết quả sẽ được phân trang và sắp xếp. Chỉ khi đó LINQDataSource mới thực hiện câu lệnh LINQ và lấy dữ liệu về, và LINQ to SQL sẽ đảm bảo rằng việc sắp xếp và phân trang này được thực hiện bên trong CSDL – và chỉ có đúng 10 dòng được trả về.

Chú ý dưới đây chúng ta vẫn dùng GridView để chỉnh sửa và xóa dữ liệu, ngay cả khi dùng sự kiện “Selecting” của LinqDataSource:

Products from UK, France, Germany						
	ProductID	ProductName	UnitPrice	UnitsInStock	Discontinued	
Edit	19	Teatime Chocolate Biscuits	9.7000	25	<input type="checkbox"/>	
Edit	28	Rössle Sauerkraut	45.6000	26	<input checked="" type="checkbox"/>	
Edit	77	Original Frankfurter grüne Soße	13.0000	32	<input type="checkbox"/>	
Update Cancel	20	Sir Rodney's Marmalade	81.0000	40	<input type="checkbox"/>	
Edit	27	Schoggi Schokolade	43.9000	49	<input type="checkbox"/>	
Edit	58	Escargots de Bourgogne	13.2500	62	<input type="checkbox"/>	
Edit	39	Chartreuse verte	10.6288	69	<input type="checkbox"/>	
Edit	25	NuNuCa Nuß-Nougat-Creme	14.0000	76	<input type="checkbox"/>	
Edit	59	Raclette Courdavault	55.0000	79	<input type="checkbox"/>	
Edit	75	Rhönbräu Klosterbier	4.5762	121	<input type="checkbox"/>	
1	2					

Khả năng hỗ trợ việc xóa/sửa dữ liệu sẽ còn làm việc chừng nào thuộc tính Query của sự kiện Selecting còn được gán một tập các thực thể cùng loại (ví dụ: một dãy các đối tượng kiểu Product, Supplier, Category, Order...). LINQDataSource khi đó sẽ có thể tự động xử lý các trường hợp UI control thực hiện việc cập nhật đổi với nó.

Để học thêm về cách cập nhật trong LINQ to SQL, xin đọc lại bài 4 của loạt bài này. Và sau đó đọc tiếp bài 5 để xem cách cập nhật với LINQDataSource.

Thực hiện các phép chiếu khi truy vấn với sự kiện Selecting

Một trong những điểm mạnh của LINQ là khả năng trả về các “đạng” dữ liệu tùy biến, hay còn gọi là phép chiếu dữ liệu. Đó là khả năng mà bạn chỉ trả về một tập con các giá trị của thực thể (một số cột nào đó mà thôi), hay trả về các giá trị được tính toán tự động bằng các biểu thức do bạn định nghĩa. Bạn có thể tìm hiểu thêm cách LINQ thực hiện các phép chiếu này trong phần 3 của loạt bài này.

Ví dụ, bạn có thể sửa lại sự kiện hàm xử lý sự kiện Selecting để đưa thông tin vào cho GridView một tập tùy biến các giá trị của Product. Trong grid này, ta sẽ chỉ hiển thị ProductID, ProductName, Product UnitPrice, số lệnh đặt hàng trên sản phẩm này

(Number of Orders), và doanh thu của sản phẩm (Revenue). Chúng ta có thể tính toán động 2 giá trị cuối dùng một biểu thức LINQ như dưới đây:

```
protected void ProductDataSource_Selecting(object sender, LinqDataSourceSelectEventArgs e)
{
    NorthwindDataContext northwind = new NorthwindDataContext();

    var products = from p in northwind.Products
                   where p.Category.CategoryName.StartsWith("C")
                   select new {
                       p.ProductID,
                       p.ProductName,
                       p.UnitPrice,
                       NumOrders = p.OrderDetails.Count,
                       Revenue = p.OrderDetails.Sum(o=>o.Quantity * o.UnitPrice)
                   };

    e.Result = products;
}
```

Ghi chú: hàm Sum được dùng để tính toán Revenue ở trên là một ví dụ về “Phương thức mở rộng” (Extension Method). Tham số được truyền cho hàm này là một ví dụ về biểu thức Lambda. Kiểu trả về được tạo bởi biểu thức LINQ là một kiểu vô danh (anonymous type) và nó được hình thành từ biểu thức truy vấn. Extension Methods, Lambda Expressions, và Anonymous Types là các đặc tính mới của VB và C# trong VS 2008.

Kết quả của biểu thức LINQ trên khi gắn nối vào GridView sẽ tương tự như sau:

Products

ProductID	ProductName	UnitPrice	NumOrders	Revenue
2	Chang	23.0000	44	18559.200000
3	Aniseed Syrup	10.0000	12	3080.000000
4	Chef Anton's Cajun Seasoning	23.0000	20	9424.800000
5	Chef Anton's Gumbo Mix	21.3500	10	5801.150000
6	Grandma's Boysenberry Spread	25.0000	12	7345.000000
8	Northwoods Cranberry Sauce	40.0000	13	13760.000000
15	Genen Shouyu	15.5000	6	1813.500000
16	Pavlova	17.4500	43	18748.050000
19	Teatime Chocolate Biscuits	9.7000	37	6159.500000
20	Sir Rodney's Marmalade	81.0000	16	23635.800000

rằng chúng ta đã chuyển sang dùng câu lệnh LINQ tùy biến.

Dù vậy, vẫn có một tính năng sẽ không làm việc khi dùng phép chiếu dữ liệu, đó là việc hỗ trợ cập nhật dữ liệu ngay trong GridView. Đó là vì LINQDataSource không biết cách

nào để cập nhật dữ liệu một cách an toàn. Nếu chúng ta muốn thêm khả năng cập nhật vào cho GridView để hỗ trợ các kiểu trả về tùy biến như vậy, chúng ta hoặc sẽ phải chuyển sang dùng một control ObjectDataSource (ta phải cung cấp thêm phương thức Update để xử lý việc cập nhật), hoặc phải cung cấp thêm một trang để người dùng cập nhật – và hiển thị một DetailsView hay FormView gắn nối và thực thể Product để chỉnh sửa.

Tổng kết

Bạn có thể dễ dàng thực hiện các thao tác truy vấn thường dùng với mô hình dữ liệu LINQ to SQL dùng khả năng khai báo các bộ lọc của LINQDataSource.

Để thực hiện biểu thức lọc phức tạp hơn, bạn có thể tận dụng ưu điểm của sự kiện Selecting có trong LINQDataSource. Điều này cho phép bạn thực hiện bất kỳ logic nào bạn muốn để lấy về các dòng dữ liệu phù hợp. Bạn có thể nhiều cách để lấy dữ liệu này, chẳng hạn dùng Query Expressions, gọi Stored Procedures, hay thực hiện một câu truy vấn tùy biến.