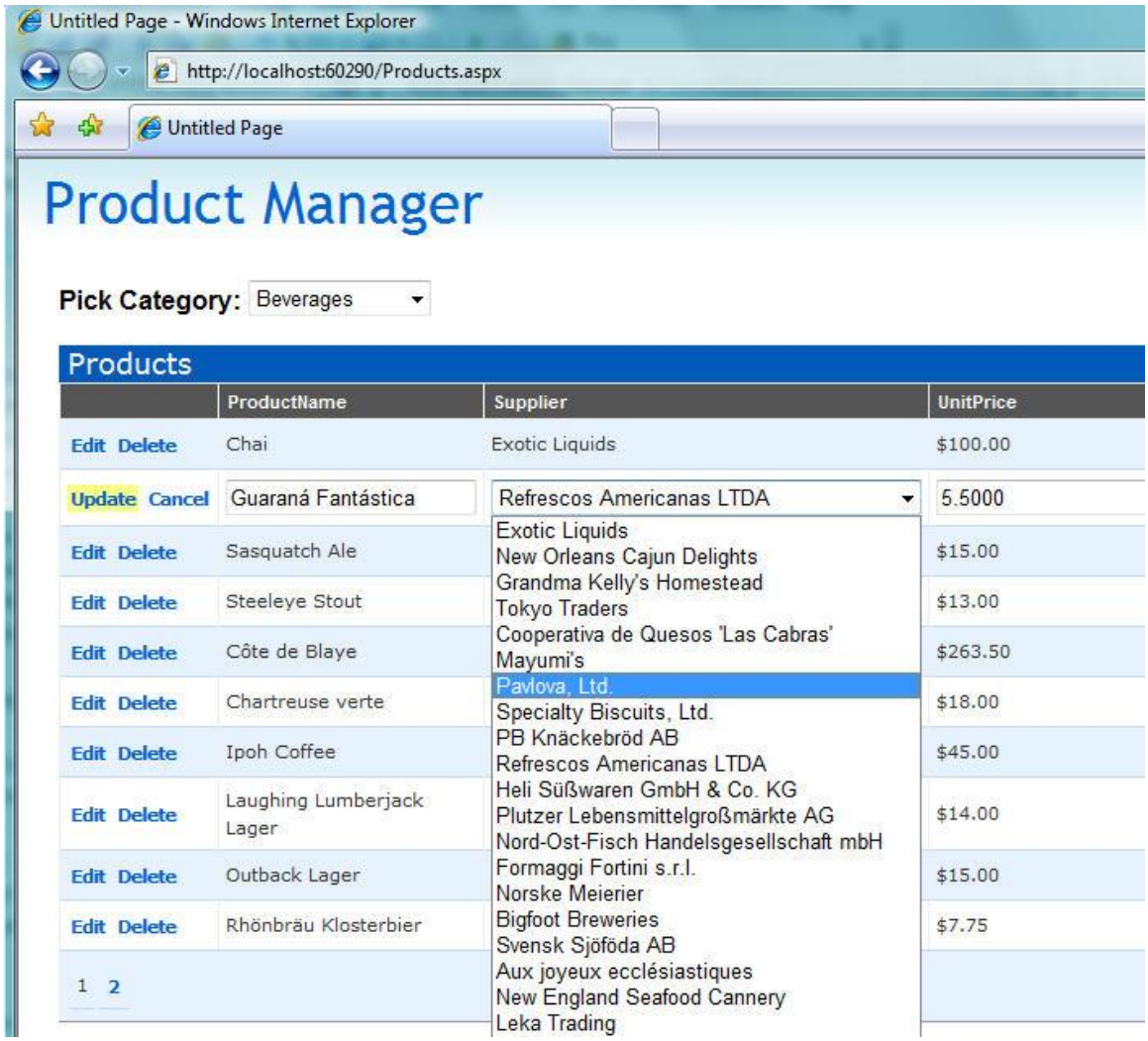
Sử dụng asp:LinqDataSource (LINQ to SQL phần 5)

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:



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 nố 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

thức query/insert/update/delete cho nguồn dữ liệu để gọi, thay vào đó bạn có thể trỏ <asp:linqdatasource> đến mô hình dữ liệu của bạn, chỉ ra bản thực thể nào bạn muốn làm việc, rồi gắn nối 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 trỏ 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 nối. Tôi có thể cho một GridView trỏ 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 cấu 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>

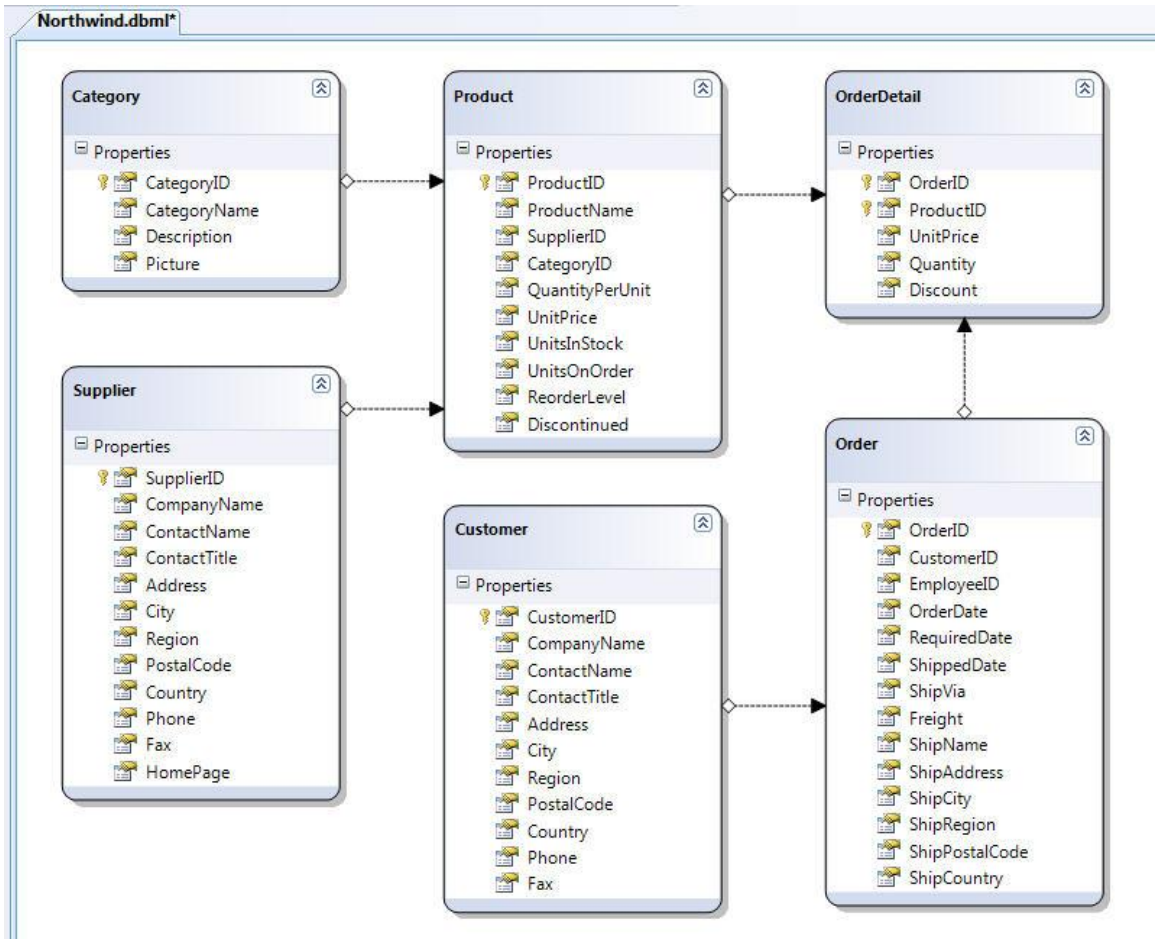
Quan trọng: Một trong những điểm hay của LINQ hay LINQ to SQL là nó không được thiết kế để chỉ làm việc với lớp giao diện, hay với một control cụ thể nào như LinqDataSource. Như bạn đã thấy trong các bài viết trước của cùng loạt bài này, viết code dùng LINQ to SQL cực kỳ sáng sủa. Bạn luôn có thể viết 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.

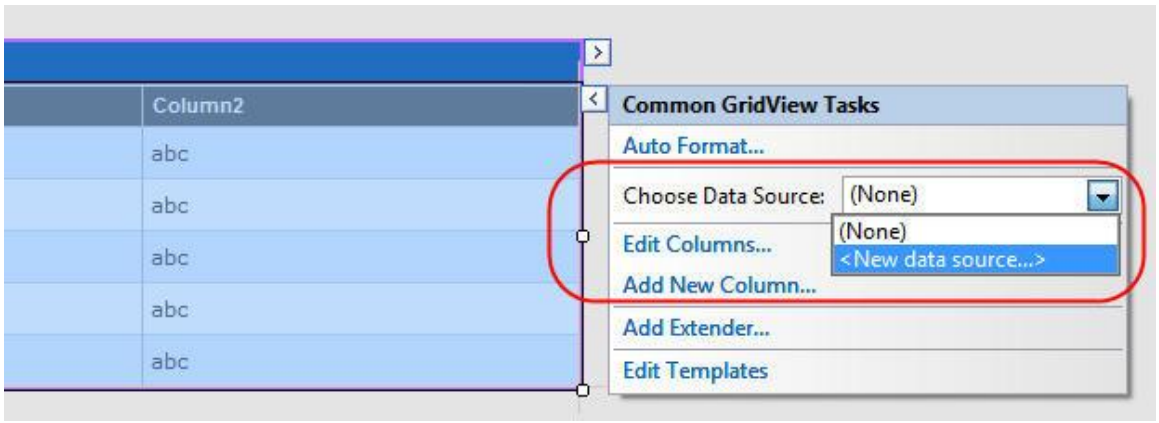
Bước 2: Tạo danh sách sản phẩm

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>

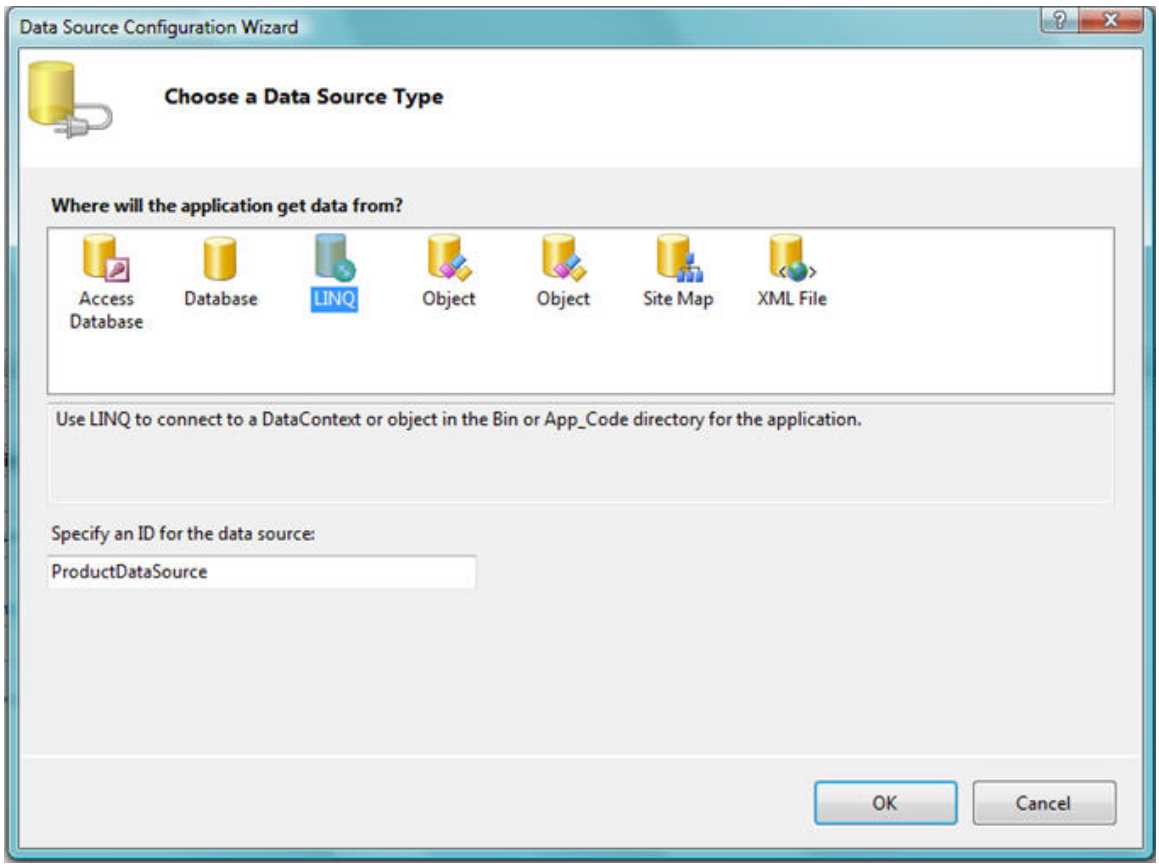
Chúng ta có thể viết code để gắn nối mô hình dữ liệu vào gridview này (giống như tôi đã làm trong phần 3), hoặc tôi có thể làm cách khác là dùng control mới <asp:linqdatasource> để gắn nối gidview này với mô hình dữ liệu.</asp:linqdatasource>

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:

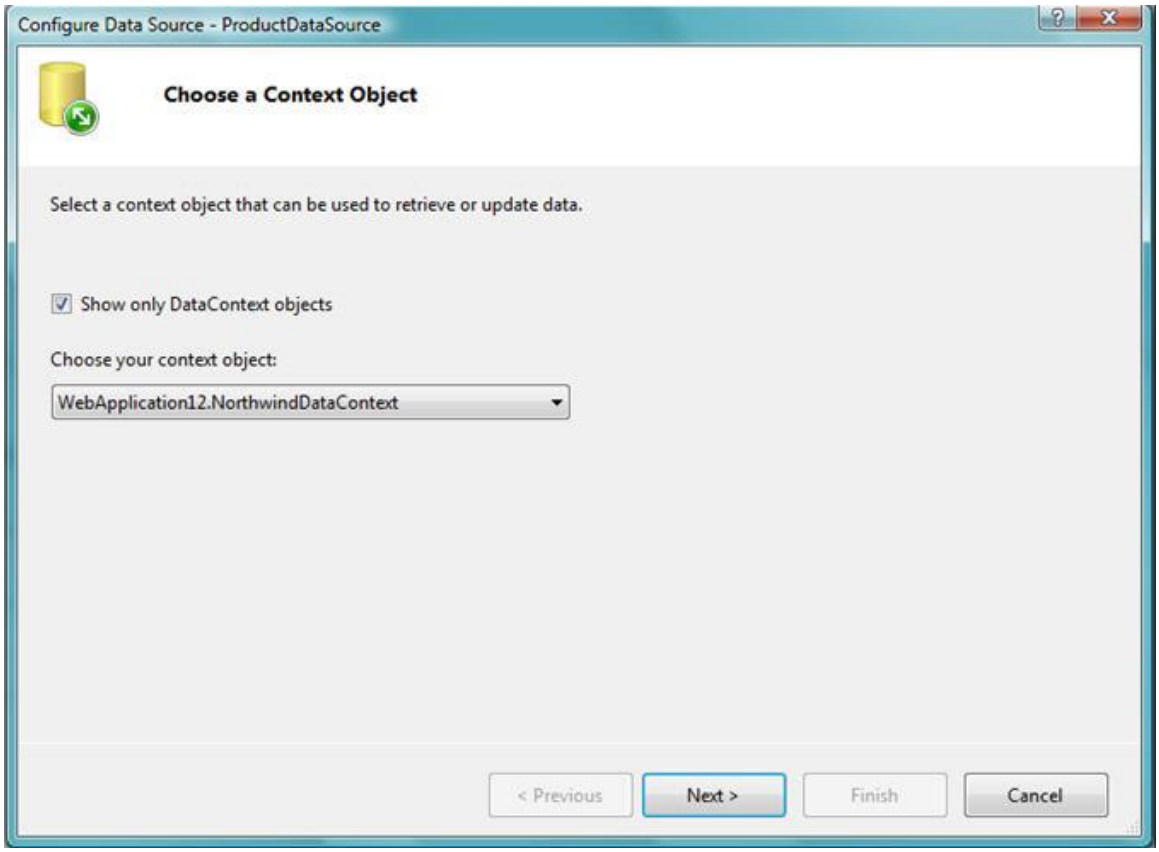
Trình thiết kế trong VS 2008 có sẵn khả năng hỗ trợ làm điều này một cách dễ dàng với GridView (hay bất kỳ control ASP.NET nào khác) vào dữ liệu LINQ. Để gắn nối, chúng ta có thể chuyển sang chế độ thiết kế, chọn GridView, và sau đó chọn “New Data Source…” bên trong dang sách “Choose Data Source:”:



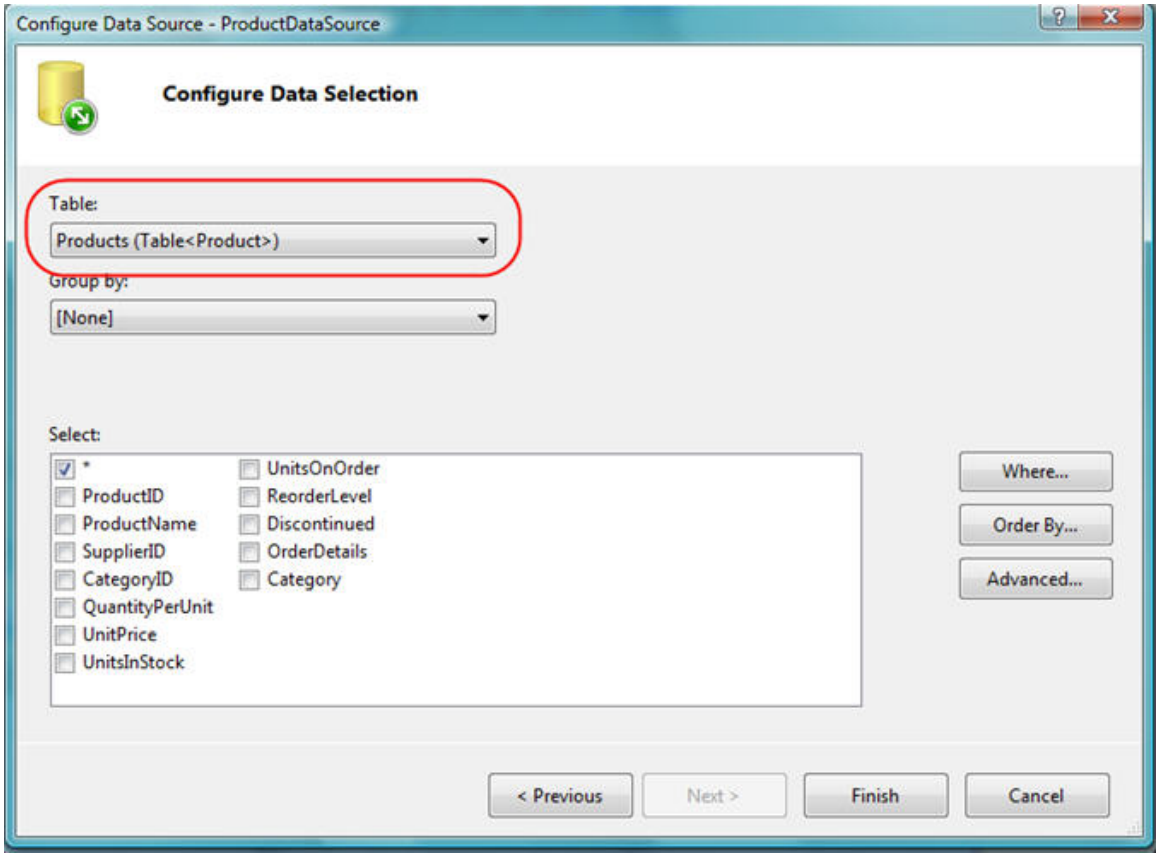
Một hộp thoại sẽ hiện lên, trong đó có danh sách các loại datasource, chọn LINQ trong hộp thoại này và đặt trên cho control <asp:linqdatasource> mà bạn muốn tạo:</asp:linqdatasource>



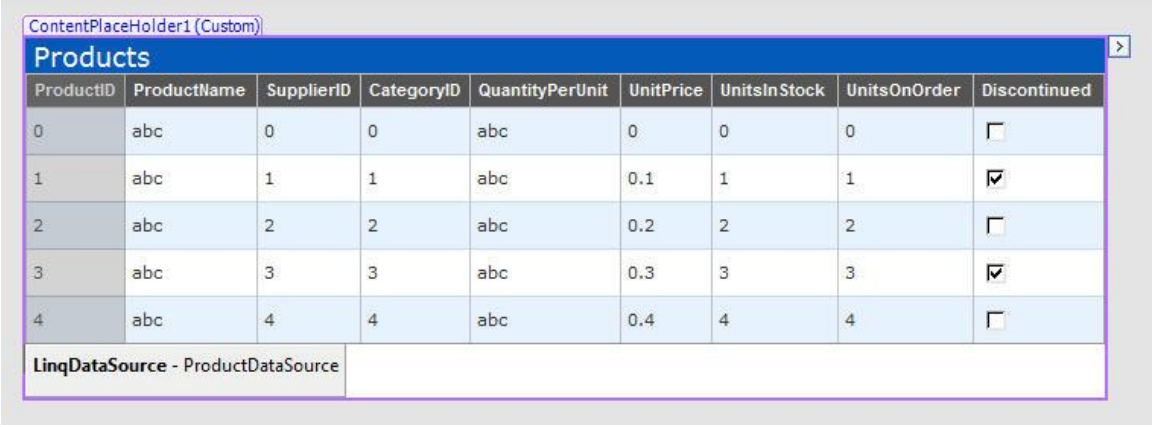
Trình thiết kế <asp:linqdatasource> sẽ hiển thị tiếp các lớp DataContext của LINQ to SQL mà ứng dụng của bạn có thể dùng được bao gồm cả trong các thư viện mà bạn đang tham chiếu tới):</asp:linqdatasource>



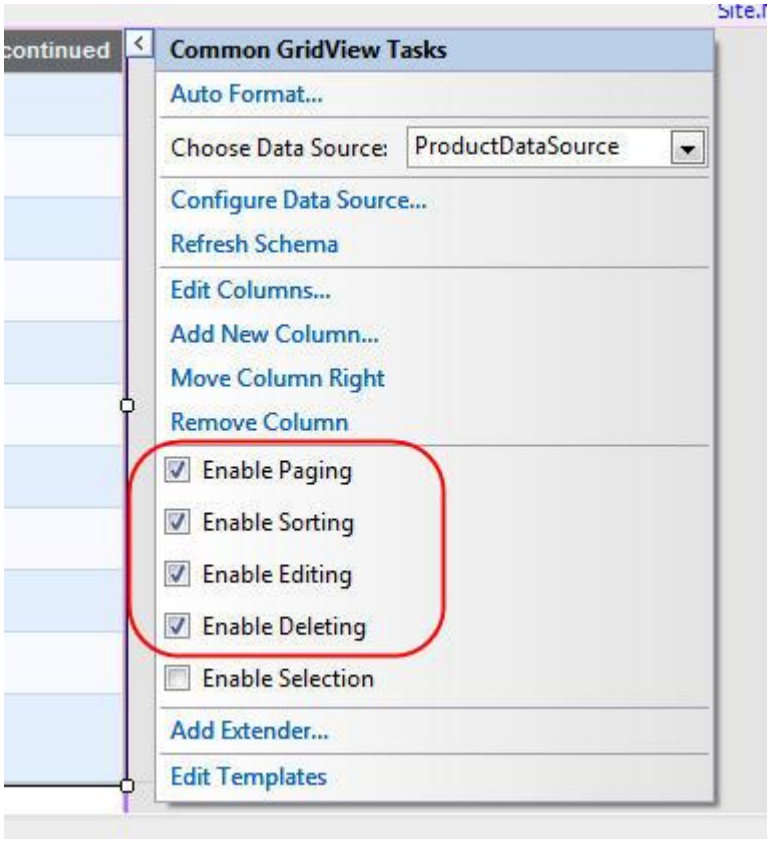
Chúng ta muốn chọn mô hình dữ liệu đã được tạo trước đây với trình thiết kế LINQ to SQL. Chúng ta cũng sẽ muốn chọn bảng dữ liệu bên trong mô hình dữ liệu mà chúng ta sẽ coi như thực thể chính để làm việc với <asp:linqdatasource>. Trong ví dụ này chúng ta sẽ chọn thực thể “Products”. Chúng ta cũng sẽ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



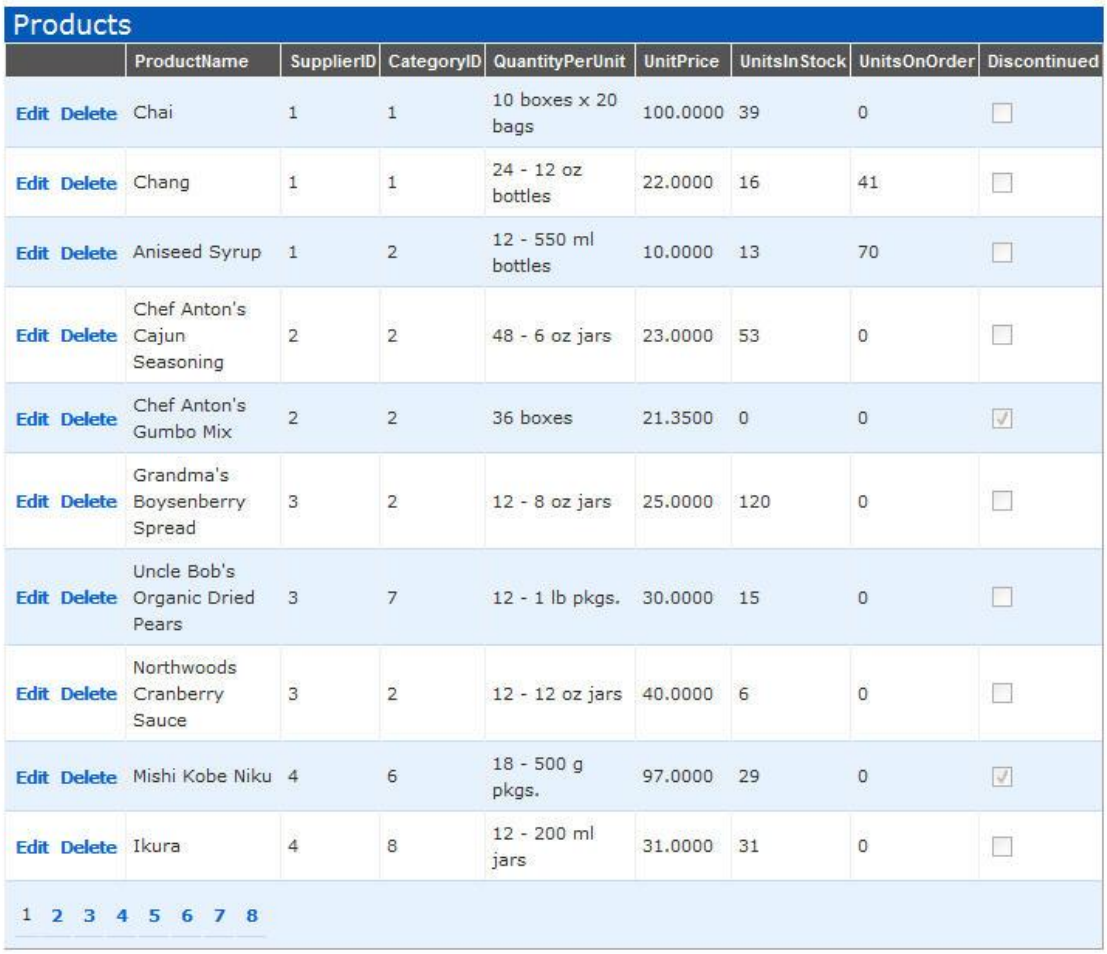
Khi nhấn vào nút “Finish” ở trên, VS 2008 sẽ khai báo một <asp:linqdatasource> trong trang .aspx, và cập nhật <asp:gridview> để trỏ đến nó (thông qua thuộc tính DataSourceID).Nó cũng sẽ tự động tạo ra các cột trong Grid dự trên cấu trúc của thực thể Products mà chúng ta đã chọn:</asp:gridview></asp:linqdatasource>



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:

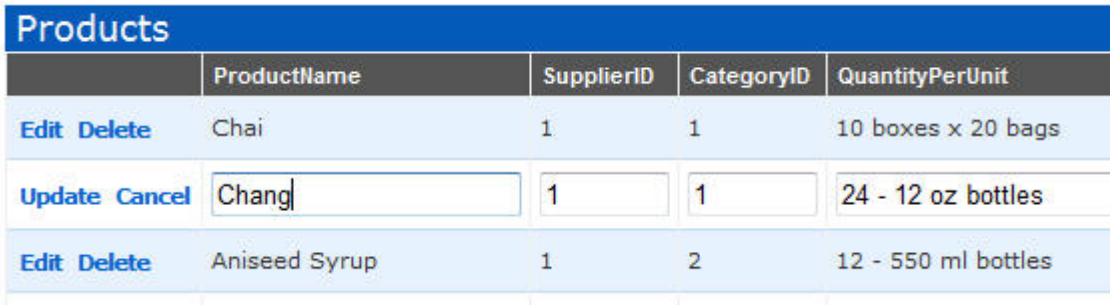


Products

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

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="
            <asp:boundfield DataField="UnitsOnOrder" HeaderText="UnitsOnOrder" SortExpression="
            <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 bớt 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 Delete	Chai	1	1	100.0000	39	0	☐
Edit Delete	Chang	1	1	22.0000	16	41	☐
Edit Delete	Aniseed Syrup	1	2	10.0000	13	70	☐
Edit Delete	Chef Anton's Cajun Seasoning	2	2	23.0000	53	0	☐
Edit Delete	Chef Anton's Gumbo Mix	2	2	21.3500	0	0	☑
Edit Delete	Grandma's Boysenberry Spread	3	2	25.0000	120	0	☐
Edit Delete	Uncle Bob's Organic Dried Pears	3	7	30.0000	15	0	☐
Edit Delete	Northwoods Cranberry Sauce	3	2	40.0000	6	0	☐
Edit Delete	Mishi Kobe Niku	4	6	97.0000	29	0	☑
Edit Delete	Ikura	4	8	31.0000	31	0	☐

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.</asp:objectdatasource>

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.</asp:linqdatasource></asp:linqdatasource>

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 Delete	Chai	1	1	100.0000	39	0	☐
Edit Delete	Chang	1	1	22.0000	16	41	☐

Điều này tuy cần thiết đứng từ góc độ mô hình dữ liệu, nhưng lại không mang lại giá trị gì cho người dùng. Thứ mà chúng ta làm là hiển thị CategoryName và SupplierName, và cung cấp một dang 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" Alternati
    <columns>
        <asp:commandfield ShowDeleteButton="True" ShowEditButton="True"></asp:commandfield>
        <asp:boundfield DataField="ProductName" HeaderText="ProductName" SortExpression="Pro

        <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="UnitPri
        <asp:boundfield DataField="UnitsInStock" HeaderText="UnitsInStock" SortExpression="U
        <asp:boundfield DataField="UnitsOnOrder" HeaderText="UnitsOnOrder" SortExpression="U
        <asp:checkboxfield DataField="Discontinued" HeaderText="Discontinued" SortExpressior

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

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>

```
<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>

```
<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:

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

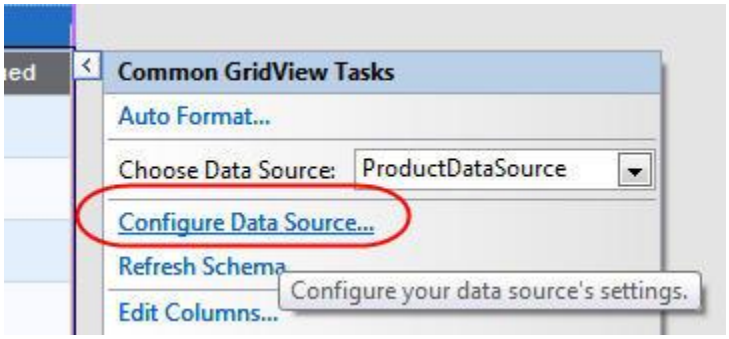
Thay vì hiển thị tất cả các sản phẩm trong CSDL, bạn có thể cập nhật phần giao diện để nó thêm một danh sách cho phép người dùng lọc lại các sản phẩm theo một phân loại nào đó.

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>
```

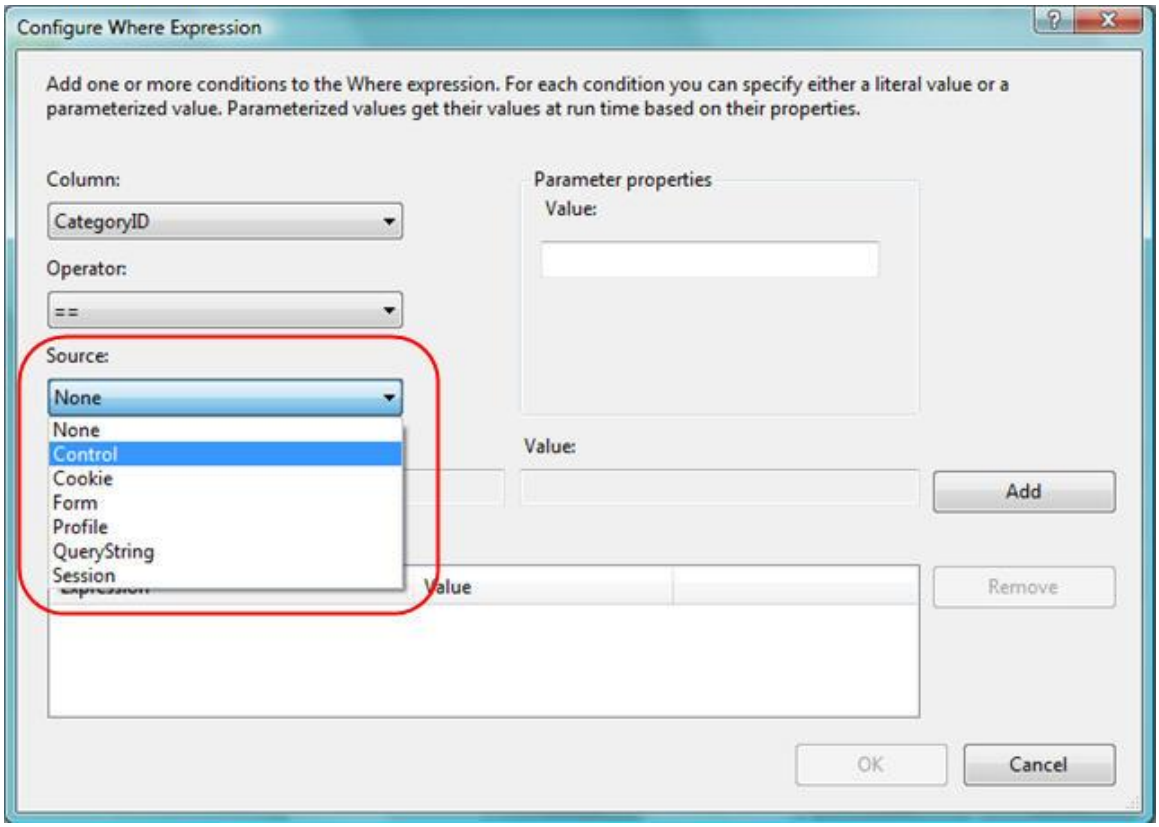
Khi tôi chạy trang web này, tôi sẽ có trên đầu trang một danh sách cho tất cả các mục phân loại:

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:

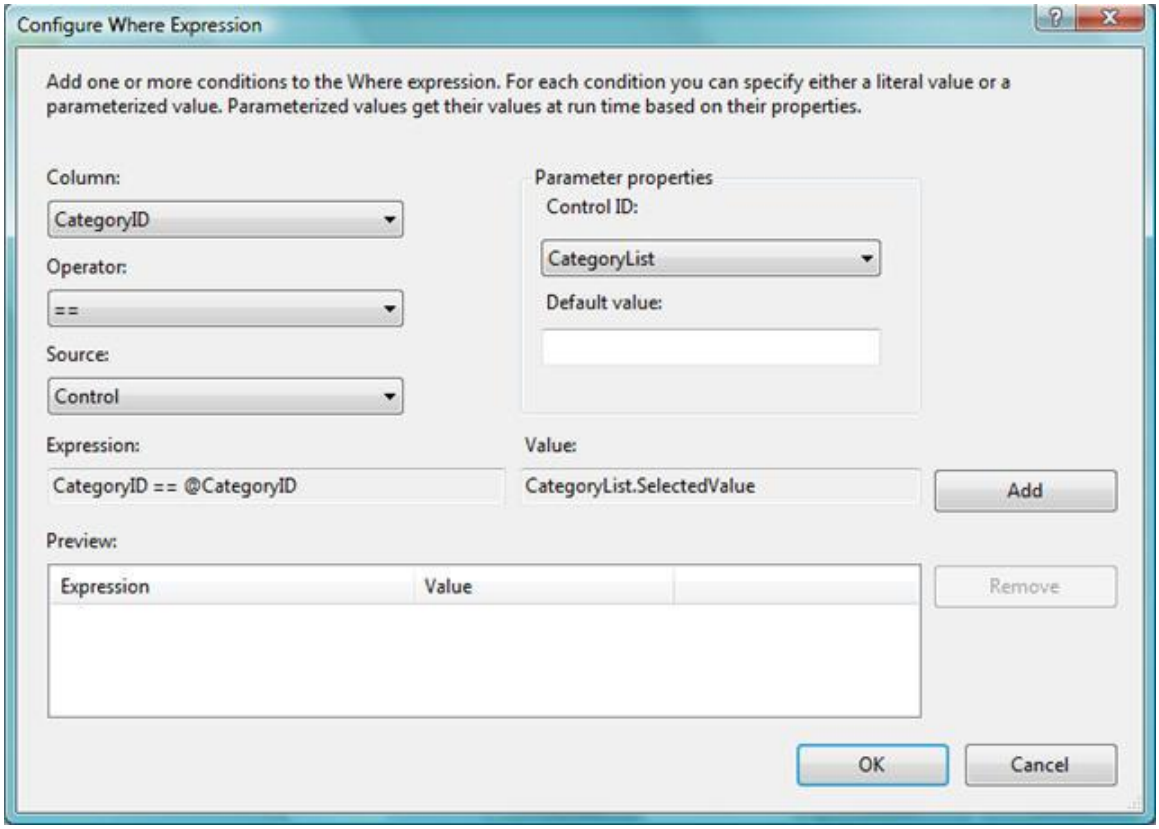


Nó sẽ đưa tôi quay trở lại cửa sổ thiết kế <asp:linqdatasource> mà tôi đã dùng trong phần đầu bài viết này. Tôi có thể chọn nút “Where” trong cửa sổ

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>



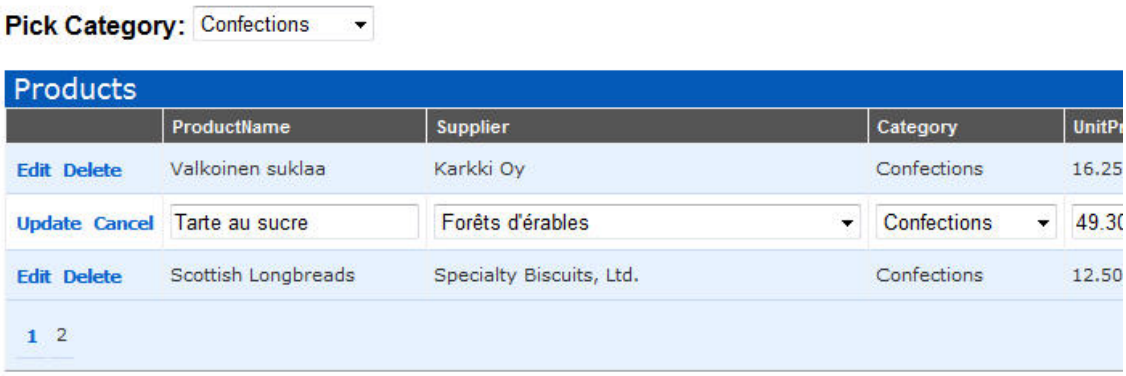
Ở trên, tôi sẽ tạo bộ lọc các Products theo CategoryID, và sau đó lấy giá trị của CategoryID muốn lọc từ danh sách mà chúng ta đã tạo trên trang:



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" Ta
    <whereparameters>
        <asp:controlparameter ControlID="CategoryList" Name="CategoryID" PropertyName="SelectedValue"
    </whereparameters>
</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:



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 dùng 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 nhờ vậy 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	UnitsInStock	UnitsOnOrder	Discontinued
Beverages	$10.00	39	0	☐
Beverages	$5.50	20	44	☐
Beverages	$15.00	111	0	☐
Beverages	13.0000	20	23	☑
Beverages	$263.50	17	0	☐
Beverages	$18.00	69	0	☐

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.

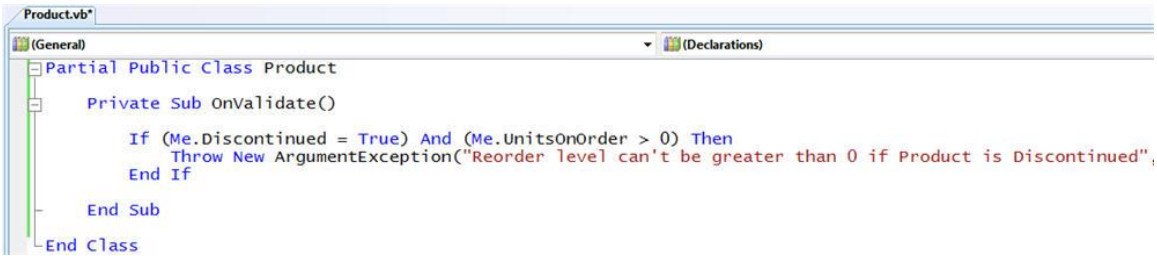
Thêm một quy tắc kiểm tra mô hình dữ liệu

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 ưng 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:



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.

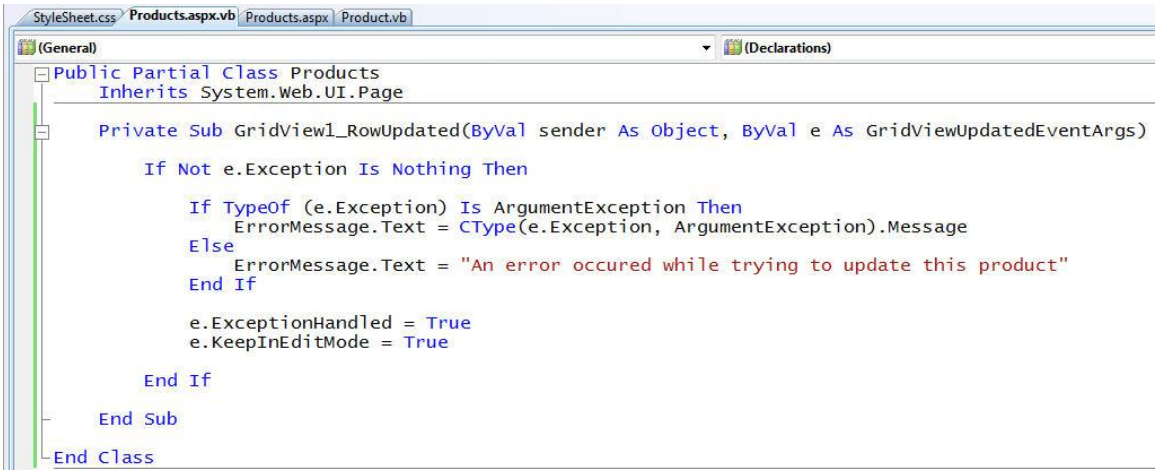
Vì <asp:linqdatasource> mà chúng ta đã định nghĩa ở trên làm việc với các lớp mô hình dữ liệu LINQ to SQL, do vậy các thao tác cập nhật/xóa/thêm đều phải qua được phép kiểm tra ở trên trước khi được áp dụng vào CSDL. Chúng ta không cần làm thêm bất kỳ điều giở lớp UI để phép kiểm tra này được thực hiện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

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.</asp:linqdatasource></asp:linqdatasource>

Đối với ứng dụng của chúng ta, nơi hợp lý nhất đến xử lý các lỗi cập nhật dữ liệu là bắt sự kiện RowUpdatedtrên 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 đo hiển thị thông báo thích hợp cho người dùng.

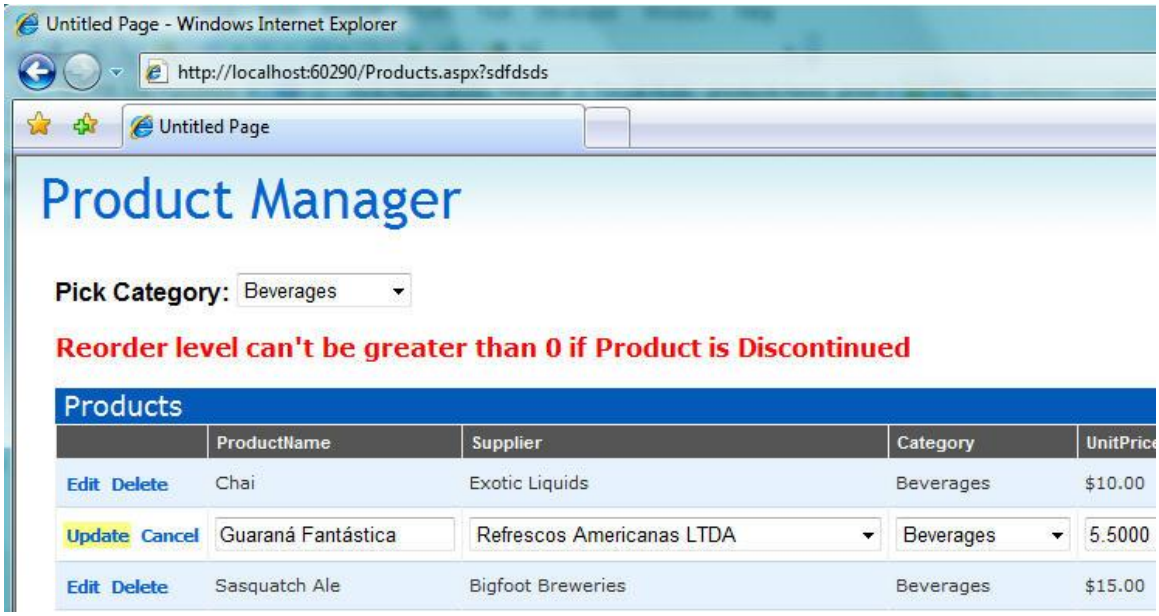


Để ý rằng ở trên tôi không hề thêm bất kỳ hàm kiểm tra nào vào lớp giao diện. Thay vì vậy, tôi sẽ lấy về chuỗi thông báo lỗi của exception đã phát ra từ phần business logic và hiển thị nó cho người dùng.

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>
```

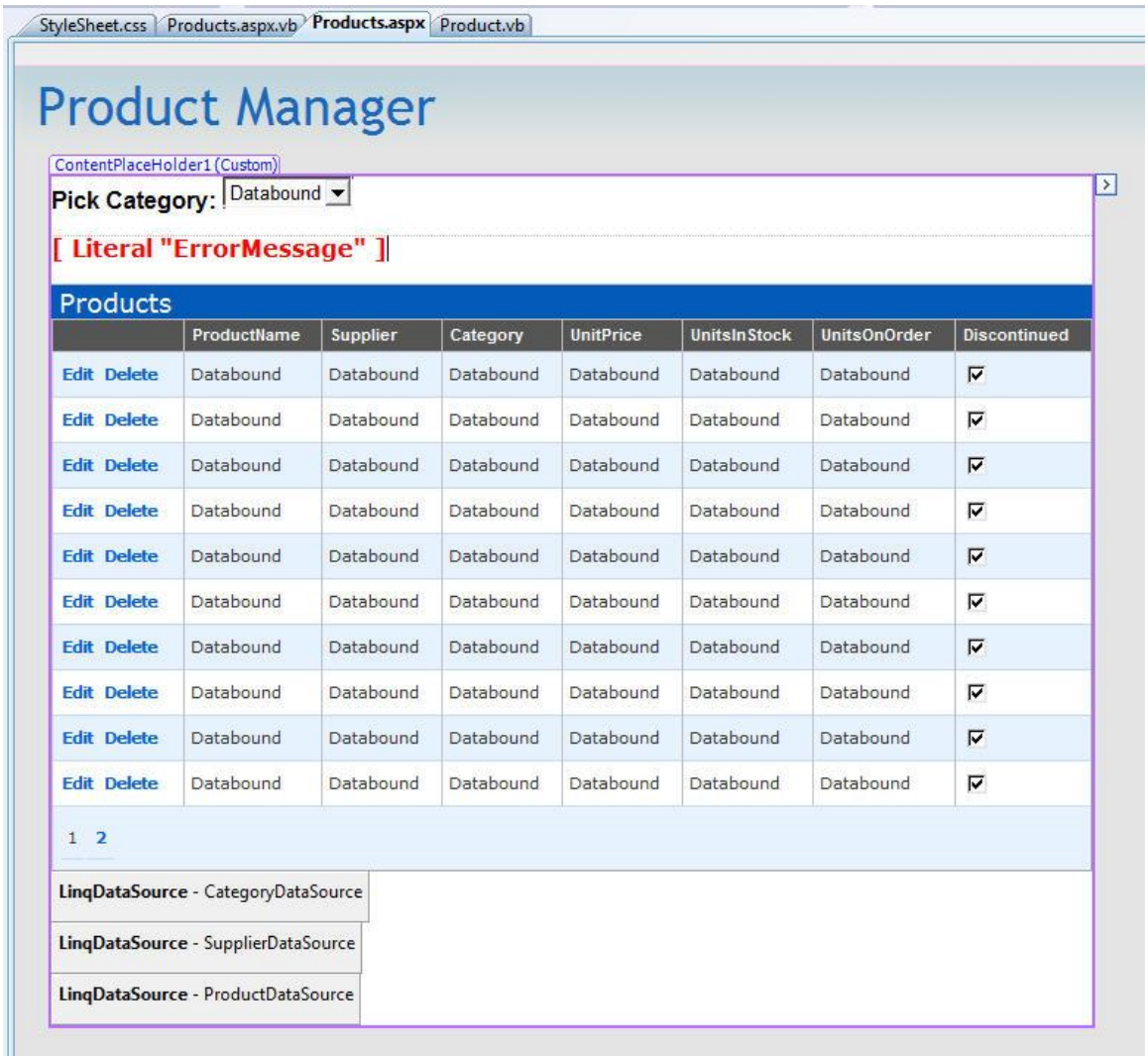
Và bấy giờ chúng ta sẽ thử cập nhật Product với các giá trị kết hợp không hợp lệ, chúng ta sẽ thấy một thông báo lỗi, nhờ đó người dùng sẽ biết cách sửa lại cho phù hợp:



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.</asp:linqdatasource>

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.</asp:linqdatasource>



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.