

MỞ ĐẦU

Linq hiện tại đã không còn quá xa lạ đối với lập trình viên và người học lập trình hiện tại. Đây là một phương thức truy vấn cơ sở dữ liệu mới bên cạnh phương thức cũ là ADO.Net. Theo nhận định của nhiều lập trình viên, Linq to SQL là cách thức khá đơn giản và dễ tiếp cận với những bạn mới biết về Linq và những bạn đã làm quen với ADO.Net trước đây.

Với bài viết sau, mong có thể giúp các bạn hiểu thêm về Linq to Sql cũng như cách sử dụng nó.

Mục Lục

I. Tổng quan LINQ:	4
I.1 LINQ là gì?	4
I.2 LINQ và ADO.NET:	4
II. LINQ to SQL	5
II.1 LINQ to SQL là gì?	5
II.2 LINQ to SQL và Entity Framework.	5
III. Làm việc với LINQ to SQL:	8
III.1 Sử dụng LINQ to SQL:	8
III.1.1 Mô hình hóa cơ sở dữ liệu dùng LINQ to SQL:	9
III.1.2 Tìm hiểu lớp DataContext:	10
III.1.3 Các ví dụ cơ bản với LINQ to SQL:	11
III.2 Truy vấn cơ sở dữ liệu:	13
III.2.1 Truy vấn dữ liệu lấy ra tập các sản phẩm.	13
III.2.2 Trực quan hóa các câu truy vấn LINQ to SQL trong trình gỡ lỗi.	15
III.2.3 Gắn nối các câu truy vấn LINQ to SQL vào các control LINQ to SQL:	16
III.2.4 Data Sharping:	18
III.2.5 Phân trang kết quả truy vấn:	20
III.3 Cập nhật cơ sở dữ liệu:	23
III.3.1 Change Tracking và DataContext.SubmitChanges():	23
III.3.2 Các ví dụ về Insert và Delete:	24
III.3.3 Cập nhật thông qua các quan hệ:	26
III.3.4 Transactions:	27
III.3.5 Kiểm tra dữ liệu và Business Logic:	28
III.3.6 Hỗ trợ kiểm tra các giá trị thuộc tính dựa trên schema của CSDL:	29
III.4 Sử dụng <asp:LinqDataSource>	29
III.4.1 <asp:LinqDataSource> là gì và nó giúp ích gì?	29
III.4.2 Ứng dụng mẫu:	30
III.4.3 Dùng <asp:LinqDataSource> với mệnh đề <where> được khai báo:	40
III.5 Lấy dữ liệu dùng Stored Procedure (SPROC):	42
III.5.1 Sử dụng SPROC vào một DataContext của LINQ:	42

<i>III.5.2</i>	<i>Ánh xạ kiểu trả về của phương thức SPROC vào một lớp trong mô hình dữ liệu:</i>	44
<i>III.5.3</i>	<i>Xử lý tham số thủ tục dạng Output:</i>	46
<i>III.5.4</i>	<i>Cập nhật dữ liệu dùng Stored Procedure</i>	46
III.6	Thực thi các biểu thức SQL tùy biến:	50
<i>III.6.1</i>	<i>Dùng các câu truy vấn SQL tùy biến với LINQ to SQL</i>	50
<i>III.6.2</i>	<i>Dùng ExecuteQuery:</i>	51
<i>III.6.3</i>	<i>Tùy biến các biểu thức SQL:</i>	51
<i>III.6.4</i>	<i>Tùy biến các câu SQL cho Inserts/ Updates/ Deletes:</i>	52
IV.	Tổng kết:	52
V.	Phụ lục:	53

I. Tổng quan LINQ:

I.1 LINQ là gì?

LINQ (*Language Integrated Query*) là một thành phần của Microsoft .NET Framework để mở rộng khả năng truy vấn dữ liệu cho các ngôn ngữ lập trình .NET như C# và VB.NET...

LINQ hỗ trợ hầu hết các dạng nguồn dữ liệu khác nhau, bao gồm cả đối tượng bộ nhớ (LINQ to Object), XML (LINQ to XML), cơ sở dữ liệu SQL Server (LINQ to SQL)...

LINQ có từ phiên bản .NET Framework 3.5, do đó nếu muốn truy vấn dữ liệu bằng LINQ thì tối thiểu chương trình phải được chạy trên nền tảng 3.5 hoặc mới hơn.

I.2 LINQ và ADO.NET:

LINQ là phần mở rộng cho phép viết các câu truy vấn ngay trong ngôn ngữ lập trình. Nó cho phép làm việc với các kiểu tập hợp dữ liệu như: XML, collection, array... và cả cơ sở dữ liệu.

ADO.NET là công nghệ cho phép các ứng dụng có thể kết nối và làm việc với các loại cơ sở dữ liệu khác nhau(truy vấn, thêm, xóa, cập nhật, gọi thủ tục...).

Bản thân LINQ không phải là một công nghệ được tạo ra để thay thế ADO.NET, chúng ta có thể làm việc với LINQ mà không dính tới cơ sở dữ liệu. Tuy nhiên, LINQ to SQL, là một phần mở rộng của LINQ, cho phép chúng ta có thể làm việc được với cơ sở

dữ liệu SQL Server, trong trường hợp này chúng ta có thể bỏ qua các câu lệnh ADO.NET mà chỉ quan tâm tới cú pháp mà LINQ cung cấp.

Chú ý: Dù không hề dùng đến ADO.NET khi viết chương trình sử dụng LINQ to SQL, nhưng đằng sau nó, ADO.NET vẫn được dùng để thực hiện kết nối, gửi các câu lệnh, lời gọi thủ tục.

II. LINQ to SQL

II.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.

II.2 LINQ to SQL và Entity Framework.

LINQ to SQL và Entity Framework có rất nhiều điểm chung, nhưng mỗi cái có những đặc tính riêng nhằm đến những trường hợp khác nhau trong .NET Framework 3.5.

LINQ to SQL có các đặc tính hướng đến việc phát triển nhanh ứng dụng với CSDL Microsoft SQL Server. LINQ to SQL cho phép chúng ta có một cái nhìn chặt chẽ về kiểu

với cấu trúc của CSDL của bạn. LINQ to SQL hỗ trợ việc ánh xạ 1-1 trực tiếp cấu trúc dữ liệu vào các lớp; một bảng đơn có thể được ánh xạ vào một cấu trúc phân cấp (ví dụ một bảng có thể chứa person, customer và employee) và các khóa ngoài có thể ánh xạ thành các quan hệ strongly-typed. Chúng ta có thể thực hiện truy vấn trên các các bảng, các view hay thậm chí các kết quả dạng bảng trả về bởi một function thông qua các phương thức. Một trong những mục tiêu thiết kế chính của LINQ to SQL là nhằm làm cho nó có thể dùng được ngay đối với những trường hợp thông thường; vậy nên, ví dụ bạn truy cập một tập các order thông qua thuộc tính Orders của một customer, và các order của customer đó chưa được đọc vào, LINQ to SQL sẽ tự động đọc vào từ CSDL cho bạn. LINQ to SQL dựa trên những quy ước cho trước, bạn có thể dựa trên các quy ước này để tùy biến, chẳng hạn như thay đổi các thao tác mặc nhiên cho việc insert, update và delete bằng cách tạo ra những câu lệnh thao tác với CSDL (ví dụ “InsertCustomer”, “UpdateCustomer”, “DeleteCustomer”). Các phương thức này có thể gọi các thủ tục trong CSDL hay thực hiện thêm các thao tác để xử lý các thay đổi.

Entity Framework có các đặc tính nhắm đến các ứng dụng doanh nghiệp (“Enterprise Scenarios”). Trong một doanh nghiệp, một CSDL thông thường được kiểm soát bởi DBA (người quản trị CSDL), cấu trúc của CSDL thông thường được tối ưu cho việc lưu trữ (hiệu năng, tính toàn vẹn, phân hoạch) hơn là cho một mô hình ứng dụng tốt, và có thể thay đổi qua thời gian khi dữ liệu và việc sử dụng phát triển lên. Với ý tưởng này, Entity Framework được thiết kế xung quanh việc xây dựng một mô hình dữ liệu hướng tới ứng dụng, ít phụ thuộc, thậm chí có thể khác một chút so với cấu trúc CSDL

thực sự. Ví dụ, bạn có thể ánh xạ một lớp đơn (hay “thực thể”) và nhiều table/view, hay ánh xạ nhiều lớp vào cùng một table/view. Bạn có thể ánh xạ vào một cấu trúc phân cấp vào một table/view đơn (như trong LINQ to SQL) hay vào nhiều table/view (ví dụ: person, customer, employee có thể nằm trong các bảng riêng biệt vì customer và employee chỉ chứa thêm một số thông tin không có trong person, hoặc lặp lại các cột từ bảng person). Bạn có thể nhóm các thuộc tính vào các kiểu phức hợp (“complex”, hay “composite”), ví dụ một kiểu Customer có thể có thuộc tính “Address” với kiểu Address có các thuộc tính Street, City, Region, Country và Postal). Entity Framework cũng cho phép bạn biểu diễn quan hệ nhiều-nhiều một cách trực tiếp, mà không cần tới bảng kết nối như một thực thể trong mô hình dữ liệu, và có một đặc tính mới được gọi là “Defining Query”, có thể được dùng cho việc biểu diễn một bảng ảo với dữ liệu lấy từ một câu truy vấn (ngoài trừ việc cập nhật phải thông qua một stored procedure). Khả năng ánh xạ mềm dẻo này, bao gồm tùy chọn dùng các stored procedure để xử lý các thay đổi, có thể được thực hiện chỉ bằng cách khai báo, hoặc chỉnh sửa lại khi yêu cầu thay đổi, mà không cần phải biên dịch lại ứng dụng

Entity Framework bao gồm LINQ to Entities đưa ra nhiều tính năng giống với LINQ to SQL trên mô hình ứng dụng ở mức khái niệm; bạn có thể xây dựng các câu truy vấn trong LINQ (hay trong “entity SQL”, một phiên bản mở rộng của SQL để hỗ trợ các khái niệm như strong-typing, đa hình, kiểu phức hợp...) trả về kết quả ở dạng các đối tượng CLR, thực thi các thủ tục hay các hàm trả về kiểu bảng thông qua các phương thức, và cho phép gọi một phương thức để lưu lại các thay đổi.

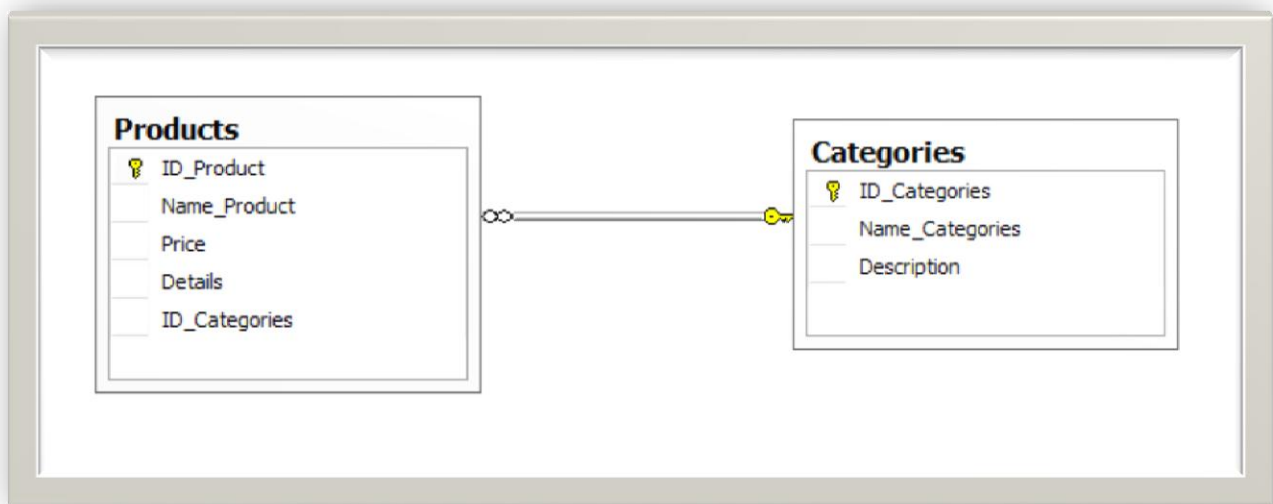
Tuy nhiên, Entity Framework còn hơn cả LINQ to Entities; nó bao gồm một lớp lưu trữ cho phép bạn dùng cùng mô hình ứng dụng mức khái niệm thông qua giao diện ADO.NET ở mức thấp dùng Entity SQL, và trả lại kết quả một cách hiệu quả nhờ các DataReader, giảm thiểu tải khi dùng trong các ngữ cảnh chỉ có đọc và không có các xử lý thêm.

Vậy nên, trong khi có nhiều phần bị trùng lặp, LINQ to SQL được nhắm đến việc phát triển nhanh ứng dụng cùng SQL Server, còn Entity Framework cung cấp các lớp truy xuất đối tượng và lưu trữ dữ liệu cho Microsoft SQL Server cũng như các CSDL khác thông qua khả năng ánh xạ mềm dẻo và ít phụ thuộc vào cấu trúc của CSDL.

III. Làm việc với LINQ to SQL:

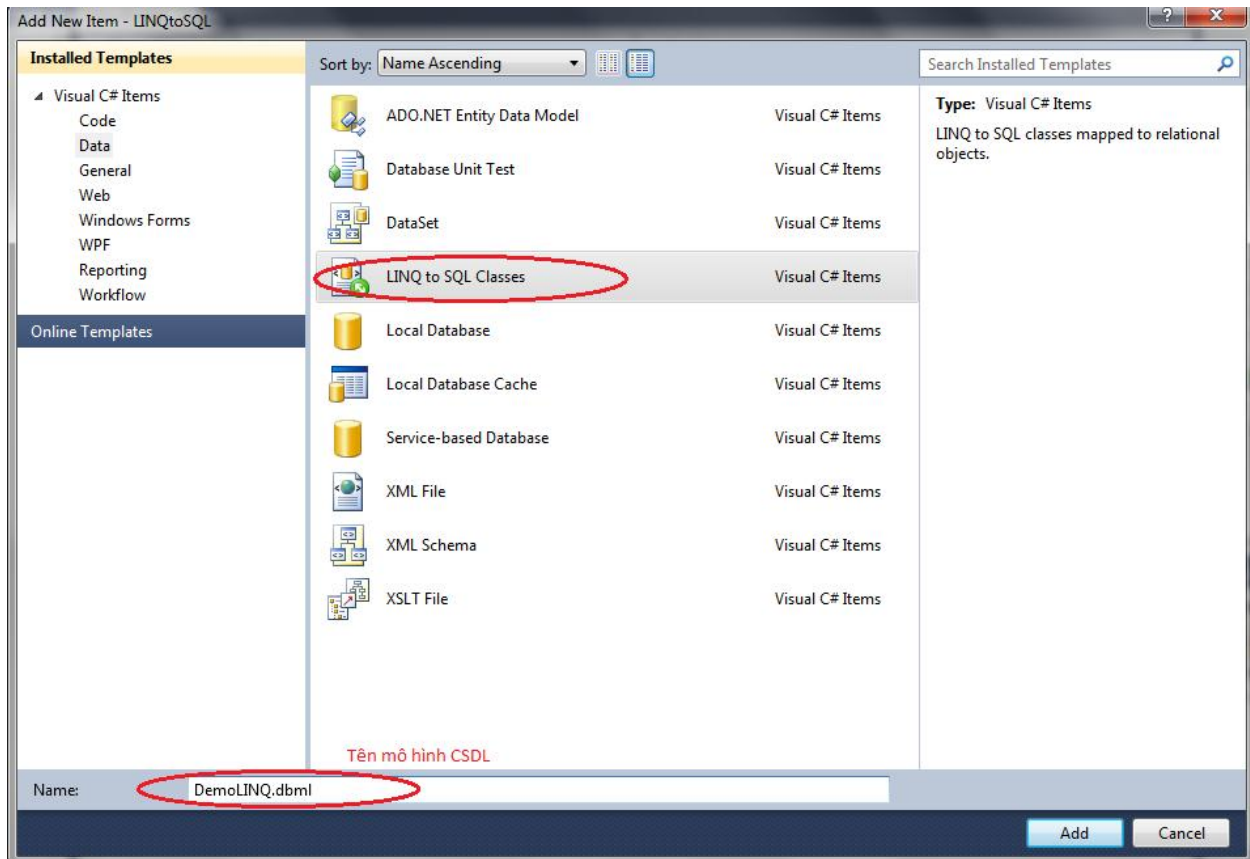
III.1 Sử dụng LINQ to SQL:

Các ví dụ của bài này sẽ được demo trong cơ sở dữ liệu sau:

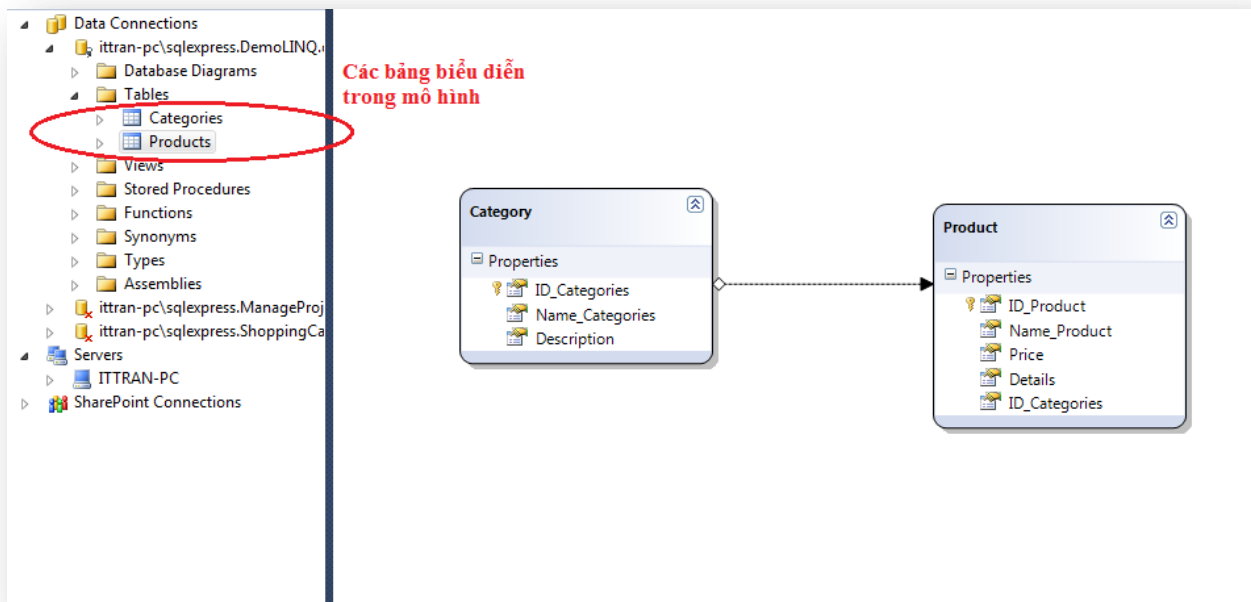


III.1.1 Mô hình hóa cơ sở dữ liệu dùng LINQ to SQL:

Có thể thêm một mô hình dữ liệu LINQ to SQL vào một project ASP.NET, Class Library hay Windows bằng cách dùng tùy chọn “Add New Item” trong Visual Studio và chọn “Linq to SQL classes”.



Việc chọn mục “LINQ to SQL Classes” sẽ khởi chạy LINQ to SQL designer, và cho phép mô hình hóa các lớp mà nó biểu diễn một CSDL quan hệ bằng cách kéo thả các bảng trong Server Explorer vào LINQ to SQL designer.



Sau đó “SAVE” lại là chúng ta đã mô hình hóa được cơ sở dữ liệu vừa chọn vào file LINQ to SQL. 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.

III.1.2 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.

III.1.3 Các ví dụ cơ bản với 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:

a. Lấy các Products từ CSDL

Đoạn code sau đây dùng cú pháp LINQ để lấy ra một tập các đối tượng Product. Các sản phẩm lấy ra thuộc loại sản phẩm có mã “cate003”.

```
DemoLINQDataContext conn = new DemoLINQDataContext();  
var prods = from p in conn.Products  
            where p.Category.ID_Categories == "cate003"  
            select p;
```

b. Cập nhật một sản phẩm trong cơ sở dữ liệu

Đoạn code sau đây cho thấy cách để lấy ra một sản phẩm có mã sản phẩm là “prod001”, cập nhật lại giá cho sản phẩm đó và lưu lại trong cơ sở dữ liệu.

```
using(DemoLINQDataContext conn = new DemoLINQDataContext())  
{  
    Product prod = conn.Products.Single(p => p.ID_Product == "prod001");  
  
    prod.Price = 550;  
  
    conn.SubmitChanges();  
}
```

c. Xóa một sản phẩm trong cơ sở dữ liệu

Đoạn code sau đây cho thấy cách để xóa sản phẩm có mã sản phẩm “prod005” khỏi cơ sở dữ liệu:

```
using (DemoLINQDataContext conn = new DemoLINQDataContext())
{
    var prod = from p in conn.Products
                where p.ID_Product == "prod005"
                select p;

    conn.Products.DeleteAllOnSubmit(prod);

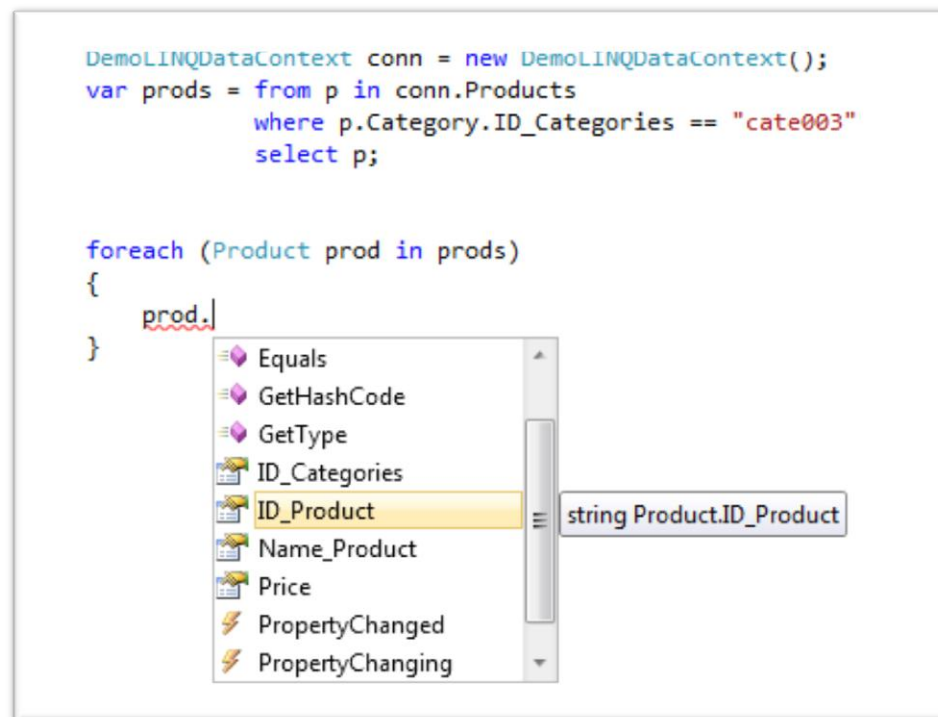
    conn.SubmitChanges();
}
```

III.2 Truy vấn cơ sở dữ liệu:

III.2.1 Truy vấn dữ liệu lấy ra tập các sản phẩm.

Một khi đã định nghĩa mô hình dữ liệu trong LINQ to SQL Classes, chúng ta dễ dàng truy cập và lấy dữ liệu từ CSDL. LINQ to SQL cho phép chúng ta thực hiện điều này bằng cách viết các câu truy vấn dùng cú pháp LINQ với lớp `DemoLINQDataContext` mà chúng ta đã tạo dùng trình thiết kế LINQ to SQL designer ở trên.

Ví dụ, để duyệt qua một tập các đối tượng Product, chúng ta có thể viết code như sau:



Trong câu truy vấn trên, chúng ta sử dụng một mệnh đề “*where*” trong cú pháp LINQ để chỉ trả về các sản phẩm trong một Category cho trước. Chúng ta hiện đang dùng ID_Categories của Product để thực hiện lọc ra các thông tin mong muốn.

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

```
DemoLINQDataContext conn = new DemoLINQDataContext();  
var prods = from p in conn.Products  
            where p.Category.Name_Categories == "Laptop"  
            select p;
```

Chú ý cách dùng thuộc tính “*Categories*” trên mỗi đối tượng Product để lọc theo Name_Categories của Categories chứa Product đó. Thuộc tính này được tự động tạo ra bởi LINQ to SQL vì chúng ta đã mô hình hóa các lớp Categories 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 dưới đây để lấy về chỉ những Categories có 3 hoặc nhiều hơn Product.

```
DemoLINQDataContext conn = new DemoLINQDataContext();  
var caty = from p in conn.Categories  
           where p.Products.Count > 3  
           select p;
```

III.2.2 Thực quan hóa các câu truy vấn LINQ to SQL trong trình gỡ lỗi.

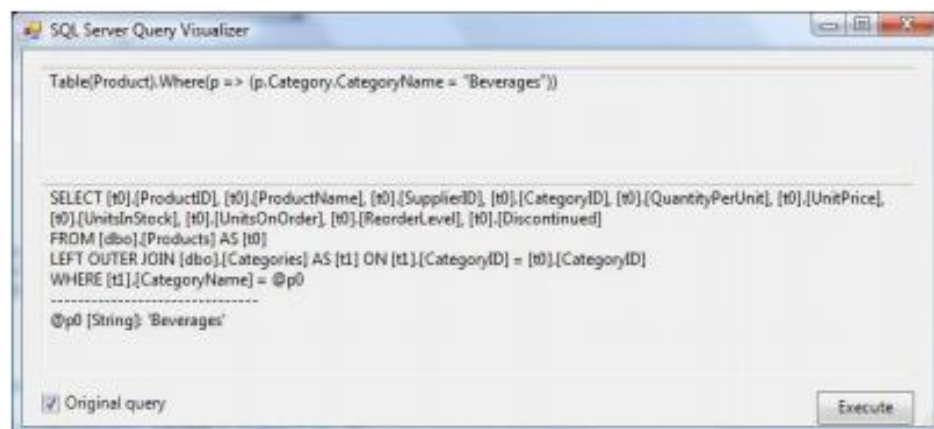
Các trình ánh xạ O/R (Object relational mapper) như LINQ to SQL tạo ra và thực thi các câu lệnh SQL một cách tự động mỗi khi bạn thực hiện một câu truy vấn hay cập nhật mô hình đối tượng của nó.

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 gỡ 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:



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.

Điều này rõ ràng làm cho việc xem những gì LINQ to SQL làm cho bạn trở thành cực kỳ dễ dàng. Nhớ 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.

III.2.3 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 `IEnumerable` – 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.

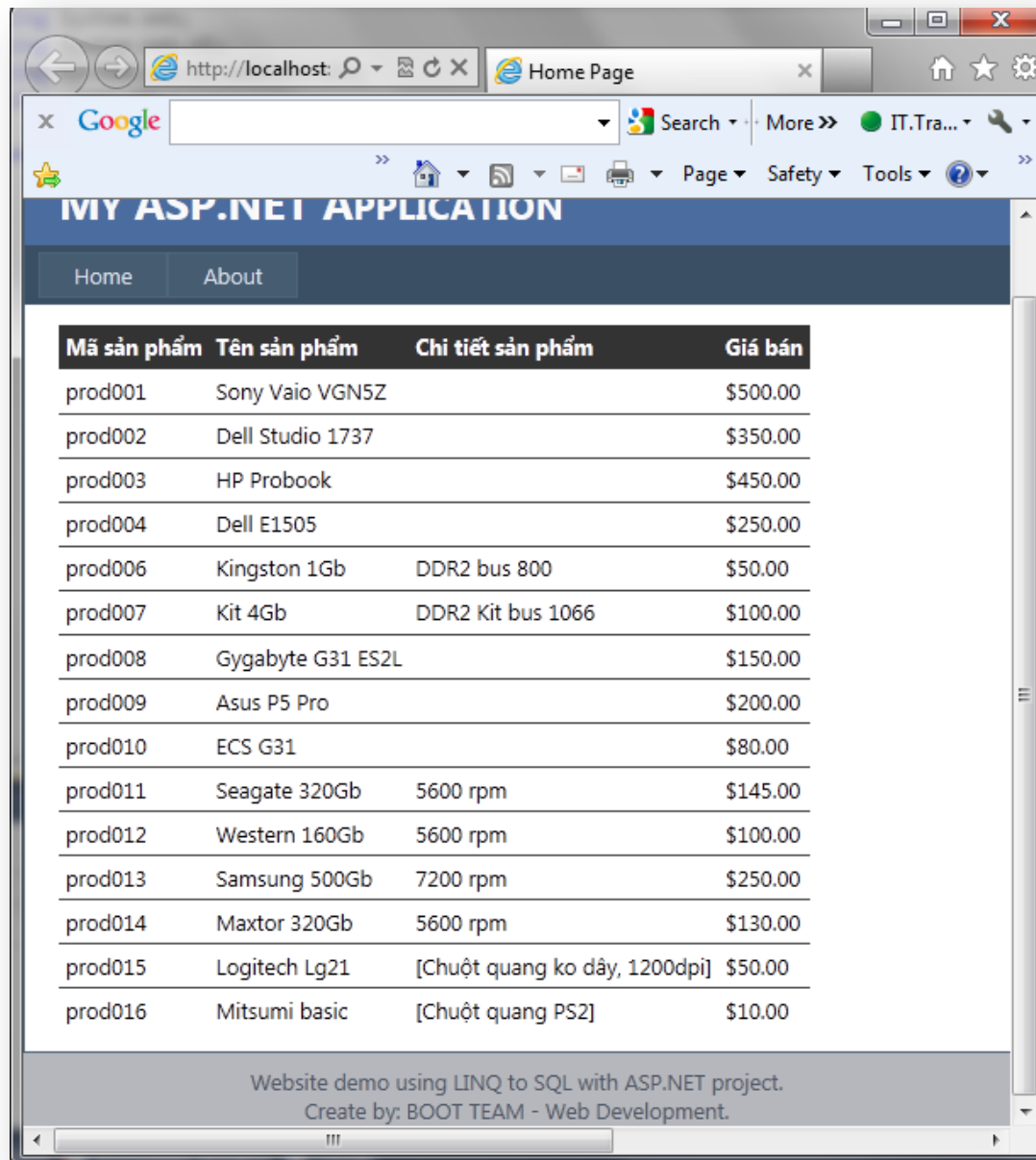
Lấy ví dụ, khai báo một control <asp:DataGrid > trong một trang .aspx như sau:

```
<asp:DataGrid ID="dgproduct" runat="server" DataKeyField="ID_Product"
    AutoGenerateColumns="False" BackColor="White" BorderColor="#CCCCCC"
    BorderStyle="None" BorderWidth="1px" CellPadding="4" ForeColor="Black"
    GridLines="Horizontal">
    <Columns>
        <asp:BoundColumn DataField="ID_Product" HeaderText="Mã sản phẩm">
        </asp:BoundColumn>
        <asp:BoundColumn DataField="Name_Product" HeaderText="Tên sản phẩm">
        </asp:BoundColumn>
        <asp:BoundColumn DataField="Details" HeaderText="Chi tiết sản phẩm">
        </asp:BoundColumn>
        <asp:BoundColumn DataField="Price" DataFormatString="{0:c}"
            HeaderText="Giá bán"></asp:BoundColumn>
    </Columns>
    <FooterStyle BackColor="#CCCC99" ForeColor="Black" />
    <HeaderStyle BackColor="#333333" Font-Bold="True" ForeColor="White" />
    <PagerStyle BackColor="White" ForeColor="Black" HorizontalAlign="Right" />
    <SelectedItemStyle BackColor="#CC3333" Font-Bold="True" ForeColor="White" />
</asp:DataGrid>
```

Chúng ta gắn nối kết quả của 1 câu lệnh LINQ to SQL đã được viết vào DataGrid như sau:

```
DemoLINQDataContext conn = new DemoLINQDataContext();
var caty = from p in conn.Products
            select p;
dgproduct.DataSource = caty;
dgproduct.DataBind();
```

Và chúng ta được một trang như sau:



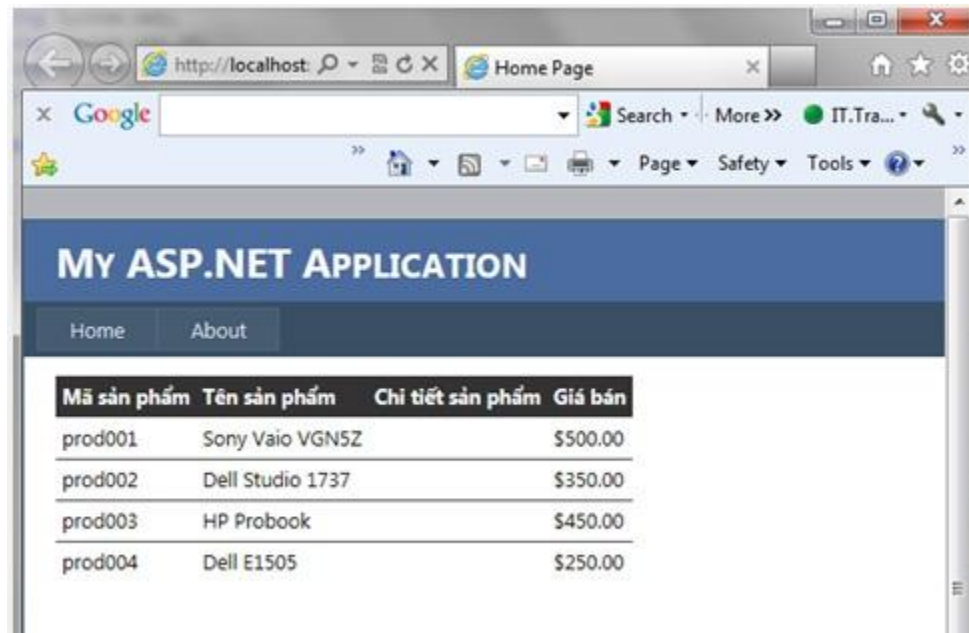
III.2.4 Data Sharping:

Hiện tại, mỗi khi xác định kết quả truy vấn, chúng ta lấy toàn bộ các cột dữ liệu cần thiết cho các đối tượng thuộc lớp Product.

Ví dụ câu lệnh lấy về các sản phẩm:

```
DemoLINQDataContext conn = new DemoLINQDataContext();  
var prods = from p in conn.Products  
            where p.Category.Name_Categories == "Laptop"  
            select p;
```

Và toàn bộ kết quả được trả về:



Mã sản phẩm	Tên sản phẩm	Chi tiết sản phẩm	Giá bán
prod001	Sony Vaio VGN5Z		\$500.00
prod002	Dell Studio 1737		\$350.00
prod003	HP Probook		\$450.00
prod004	Dell E1505		\$250.00

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# mới hỗ trợ để chỉ ra rằng chúng ta chỉ muốn một tập con bằng cách chỉnh sửa lại câu truy vấn như sau:

```
DemoLINQDataContext conn = new DemoLINQDataContext();  
var caty = from p in conn.Products  
           where p.Category.Name_Categories == "Laptop"  
           select new  
           {  
               ID_Product = p.ID_Product,  
               Name_Product = p.Name_Product  
           };|
```

Điều này sẽ chỉ trả về một tập con trong CSDL:

Mã sản phẩm	Tên sản phẩm
prod001	Sony Vaio VGN5Z
prod002	Dell Studio 1737
prod003	HP Probook
prod004	Dell E1505

III.2.5 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();
}
```

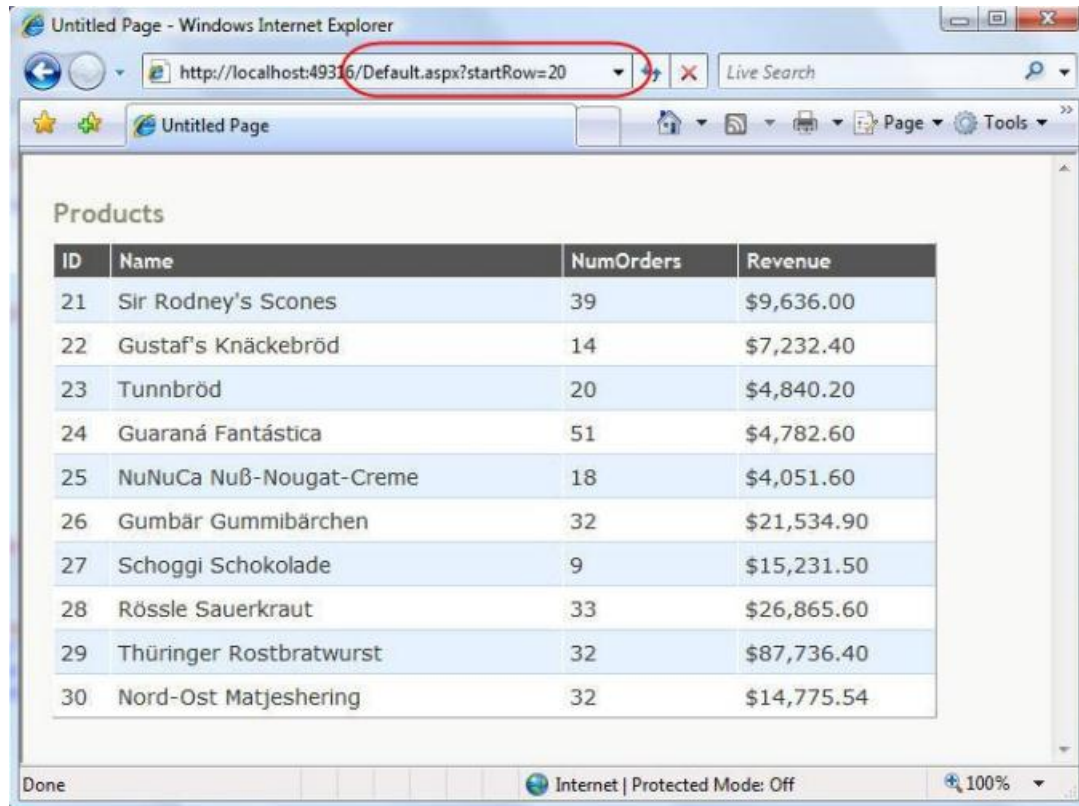
Chú ý ở trên chúng ta đã không dùng Skip() và Take() trong câu khai báo truy vấn các sản phẩm – mà chỉ dùng tới khi gắn kết dữ liệu vào GridView. Mọi người hay hỏi “Có phải làm như vậy thì câu lệnh đầu tiên sẽ lấy toàn bộ dữ liệu từ CSDL về lớp giữa, rồi sau đó mới thực hiện việc phân trang?”. Câu trả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



Products

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

III.3 Cập nhật cơ sở dữ liệu:

III.3.1 Change Tracking và DataContext.SubmitChanges():

Khi chúng ta thực hiện các câu truy vấn và lấy về các đối tượng trong CSDL, 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 chúng ta không hề dùng tài nguyên trên CSDL, hoặc chúng ta 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, đoạn code dưới đây dùng để cập nhật giá tiền của 1 sản phẩm có mã sản phẩm "prod001":

```
DemoLINQDataContext conn = new DemoLINQDataContext();

Product prod = conn.Products.Single(p => p.ID_Product == "prod001");
prod.Price = 150;
|
conn.SubmitChanges();
```

Khi chúng ta gọi `conn.SubmitChanges()` như ở trên, LINQ to SQL sẽ xây dựng và thực thi một câu lệnh SQL “UPDATE” mà nó sẽ cập nhật lại thuộc tính mà chúng ta đã sửa như ở trên.

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 giá của Product có mã là “*prod001*” có giá là 150 thì việc gọi `SubmitChanges()` sẽ chẳng làm thực thi bất kỳ câu SQL nào.

III.3.2 Các ví dụ về 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:

Chúng ta 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:

```
DemoLINQDataContext conn = new DemoLINQDataContext();

Product myProd = new Product();

myProd.ID_Product = "prod015";
myProd.ID_Categories = "cate003";
myProd.Name_Product = "Sản phẩm DEMO";
myProd.Price = 150;
myProd.Details = "Chưa cập nhật";

conn.Products.InsertOnSubmit(myProd);

conn.SubmitChanges();
```

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

Xóa các sản phẩm:

Giống như việc thêm sản phẩm vào CSDL bằng cách thêm một đối tượng Product vào tập hợp Products. Chúng ta cũng có thể xóa một tập các đối tượng khỏi CSDL bằng cách xóa các đối tượng đó trong tập chứa nó trong lớp DataContext:

```
DemoLINQDataContext conn = new DemoLINQDataContext();

var prods = from p in conn.Products
             where p.Category.ID_Categories == "cate003"
             select p;
conn.Products.DeleteAllOnSubmit(prods);

conn.SubmitChanges();
```

Trên đây, chúng ta đã lấy ra tất cả các sản phẩm của Categories có mã là “cate003” sau đó truyền nó cho phương thức DeleteAllOnSubmit() trong tập hợp Products của lớp DataContext. Khi gọi SubmitChanges(), tất cả các sản phẩm đó sẽ được xóa khỏi CSDL.

III.3.3 Cập nhật thông qua các quan hệ:

Điều làm cho các trình ORM như LINQ to SQL cực kỳ mềm dẻo là nó cho phép chúng ta dễ dàng mô hình hóa mối quan hệ giữa các bảng trong mô hình dữ liệu.

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

```
DemoLINQDataContext conn = new DemoLINQDataContext();

//Lấy ra Category có tên Laptop
Category laptop = conn.Categories.Single(c => c.Name_Categories == "Laptop");

//Tạo Product mới
Product myProd = new Product();
myProd.ID_Product = "prod020";
myProd.Name_Product = "Laptop Gygabyte";
myProd.Price = 250;

//Liên kết sản phẩm với Categories Laptop
laptop.Products.Add(myProd);

conn.SubmitChanges();
```

Hãy chú ý cách chúng ta 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 chúng ta gọi SubmitChanges.

III.3.4 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.

III.3.5 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.

III.3.6 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/duy nhất không bị trùng lặp trong CSDL.

Chúng ta 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 chúng ta không cần làm bất kỳ điều gì để cho phép chúng. LINQ to SQL cũng tự động xử lý các chuỗi escape, do vậy chúng ta không cần lo lắng về lỗi SQL injection.

III.4 Sử dụng <asp:LinqDataSource>

III.4.1 <asp:LinqDataSource> là gì và nó giúp ích gì?

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, ta 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 SqlDataSource hay ObjectDataSource gắn nối trực tiếp vào CSDL, còn <asp:LinqDataSource> được thiết kế để gắn vào một mô hình dữ liệu LINQ.

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...). Ta không cần định nghĩa các phương thức query/ insert/ update/ delete cho nguồn dữ liệu để gọi, thay vào đó ta có thể trở <asp:LinqDataSource> đến mô hình dữ liệu, 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.

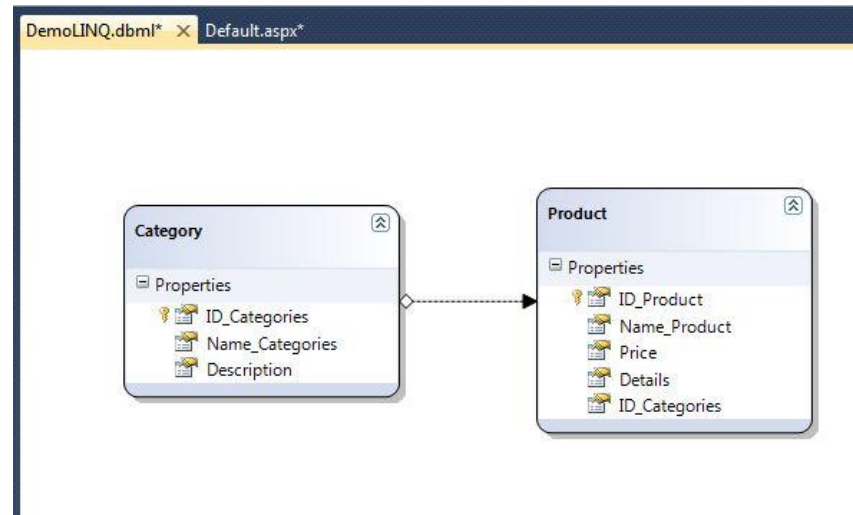
III.4.2 Ứng dụng mẫu:

Ta sẽ xây dựng một ứng dụng Web (với mô hình dữ liệu hướng đối tượng dùng LINQ to SQL) hỗ trợ người dùng các tính năng sau:

1. Cho phép lọc sản phẩm theo phân loại.
2. Cho phép sắp xếp sản phẩm khi nhấp chuột lên tiêu đề cột (Name_Product, Price, Details, ...)
3. Cho phép phân trang các sản phẩm (10 sản phẩm mỗi trang)
4. Cho phép chỉnh sửa và cập nhật các chi tiết sản phẩm ngay trên trang.
5. Cho phép xóa các sản phẩm trong danh sách.

Tiến hành:**1. Bước 1:** Tạo định nghĩa mô hình dữ liệu:

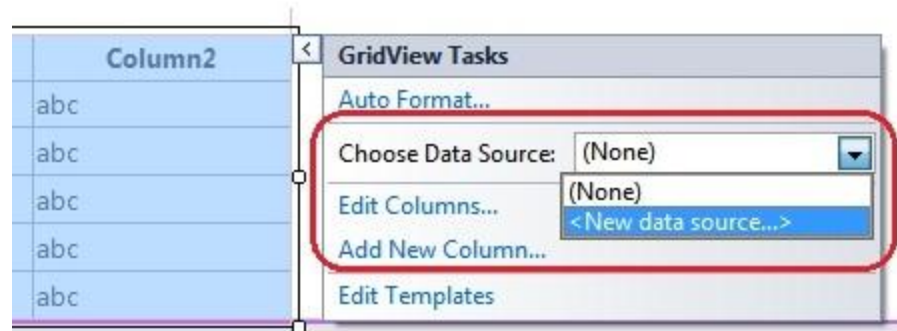
Ở đây, chúng ta tạo một mô hình dữ liệu LINQ to SQL với VS 2010 (như hướng dẫn ví dụ trên):

**2. Bước 2:** Tạo danh sách sản phẩm

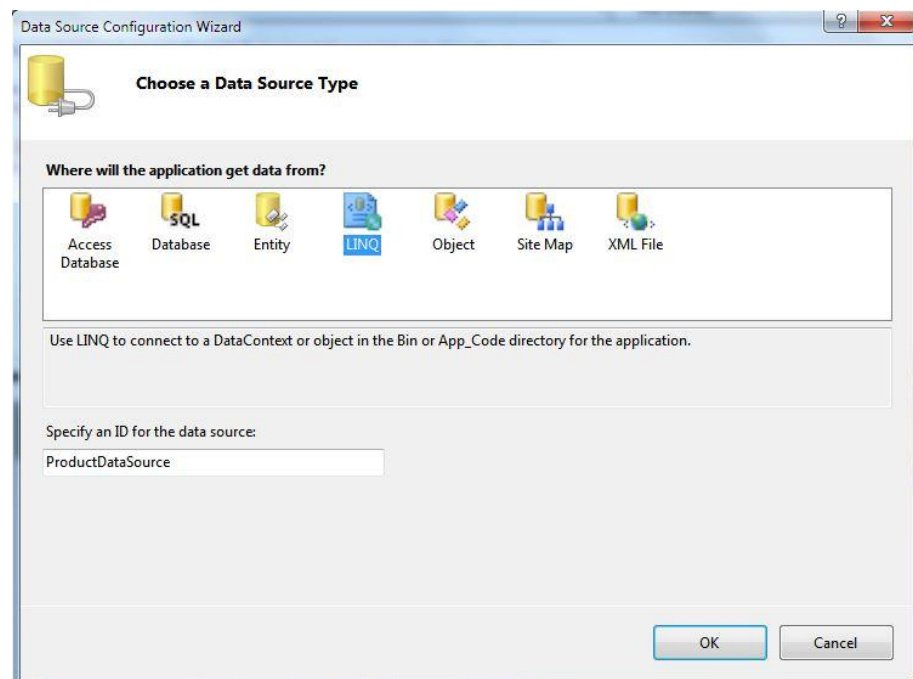
Chúng ta bắt đầu phần giao diện bằng cách tạo một trang ASP.NET đơn giản với một control <asp:GridView>:

```
Client Objects & Events (No Events)
</asp:Content>
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolder="BodyContent">
    <h2>Products</h2>
    <br />
    <asp:GridView ID="GridView1" runat="server">
    </asp:GridView>
</asp:Content>
```

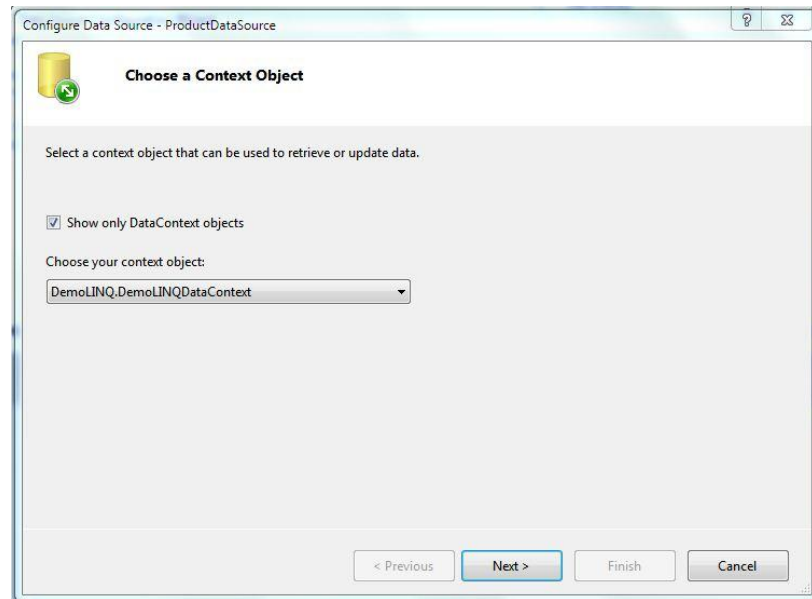
Ta sẽ dùng <asp:LinqDataSource> để kết nối GridView với mô hình dữ liệu:



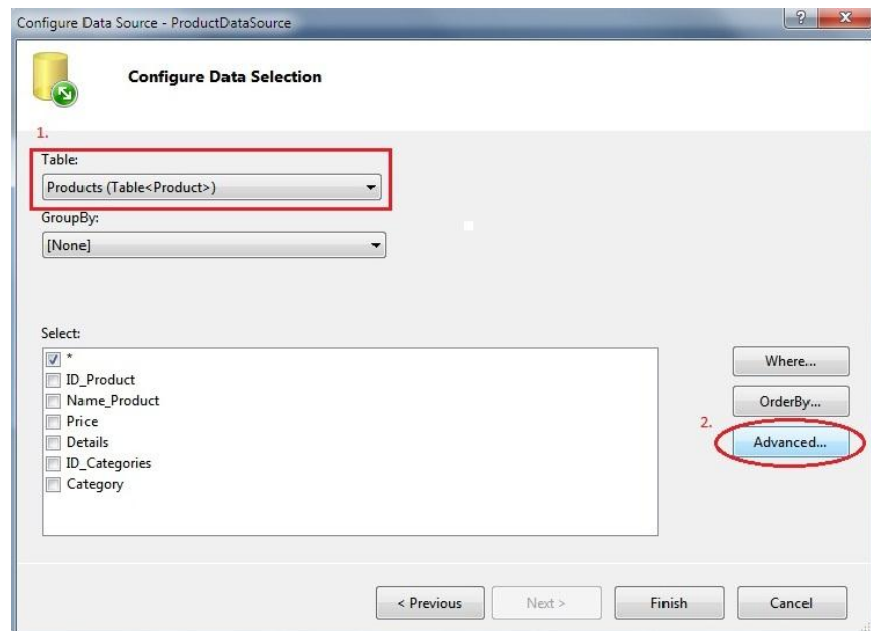
Chọn Data Source và đặt tên cho control `<asp:LinqDataSource>` mà bạn muốn tạo:

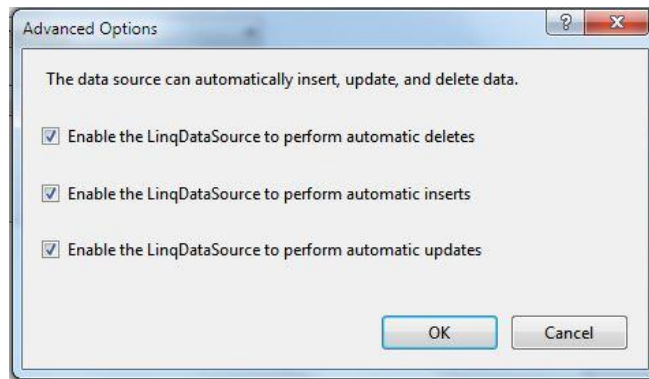


Đến đây sẽ hiển thị các lớp DataContext của LINQ to SQL mà ứng dụng của bạn có thể dùng. Ta chọn mô hình dữ liệu đã tạo:

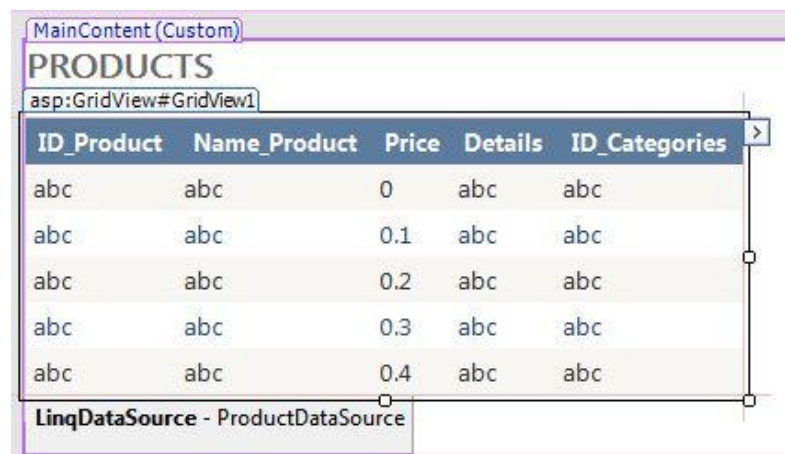


Trong ví dụ này, ta chọn thực thể “Products”. Tiếp tục nhấn Advanced cho phép cập nhật cũng như xóa dữ liệu:

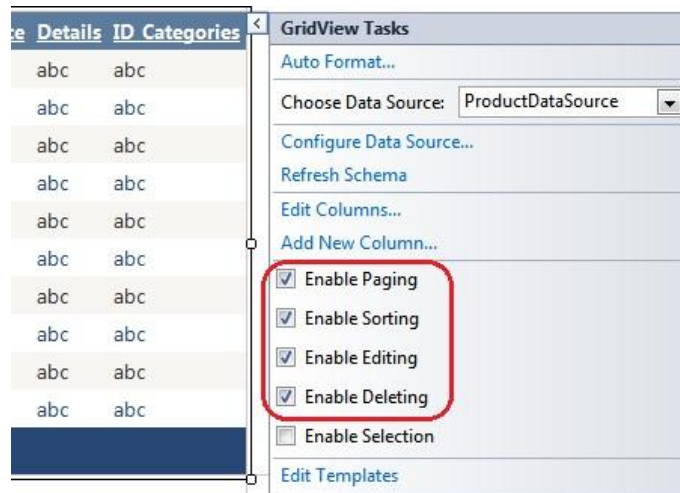




Khi nhấn “Finish”, VS 2010 sẽ khai báo một `<asp:LinqDataSource>` trong trang .aspx, và cập nhật `<asp:GridView>` thông qua thuộc tính `DataSourceID`. Nó cũng sẽ tự động tạo ra các cột trong GridView dựa trên cấu trúc thực thể Products mà ta chọn:



Ta tiếp tục vào GridView Tasks chọn các thuộc tính cho phép phân trang, sắp xếp, chỉnh sửa và xóa dữ liệu:



Ta có thể nhấn F5 để thực thi và một trang hiển thị danh sách Product với các chức năng đã chọn:

My ASP.NET Application

[Home](#)
[About](#)

PRODUCTS

	ID Product	Name Product	Price	Details	ID Categories
Edit Delete	prod001	Sony Vaio VGN5Z	500.0000		cate007
Edit Delete	prod002	Dell Studio 1737	350.0000		cate007
Edit Delete	prod003	HP Probook	450.0000		cate007
Edit Delete	prod004	Dell E1505	250.0000		cate007
Edit Delete	prod005	Kingmax 2Gb	70.0000	Bus 1066	cate002
Edit Delete	prod006	Kingston 1Gb	50.0000	DDR2 bus 800	cate002
Edit Delete	prod007	Kit 4Gb	100.0000	DDR2 Kit bus 1066	cate002
Edit Delete	prod008	Gygabyte G31 ES2L	150.0000		cate001
Edit Delete	prod009	Asus P5 Pro	200.0000		cate001
Edit Delete	prod010	ECS G31	80.0000		cate001

1
2

Ta cũng có thể nhấn Edit hoặc Delete để cập nhật lại dữ liệu:

	ID Product	Name Product	Price	Details	ID Categories
Update Cancel	prod001	Asus	500.0000		cate007
Edit Delete	prod002	Dell Studio 1737	350.0000		cate007
Edit Delete	prod003	HP Probook	450.0000		cate007
Edit Delete	prod004	Dell E1505	250.0000		cate007
Edit Delete	prod005	Kingmax 2Gb	70.0000	Bus 1066	cate002

Trong mã nguồn, thẻ `<asp:LinqDataSource>` chỉ đến lớp `DataContext` của LINQ to SQL đã tạo:

```

<SortedDescendingHeaderStyle BackColor="#6F8DAE" />
</asp:GridView>
<asp:LinqDataSource ID="ProductDataSource" runat="server"
    ContextTypeName="DemoLINQ.DemoLINQDataContext" EnableDelete="True"
    EnableInsert="True" EnableUpdate="True" EntityTypeName="" TableName="Products">
</asp:LinqDataSource>
</asp:Content>

```

Và GridView trỏ đến `<asp:LinqDataSource>` thông qua `DataSourceID` và những cột nào được hiển thị, tiêu đề cột, cũng như cách sắp xếp sẽ dùng khi tiêu đề cột được chọn.

```

<asp:GridView ID="GridView1" runat="server" AllowPaging="True"
    AllowSorting="True" AutoGenerateColumns="False" CellPadding="4"
    DataKeyNames="ID_Product" DataSourceID="ProductDataSource" ForeColor="#333333"
    GridLines="None" Height="143px" Width="390px">
    <AlternatingRowStyle BackColor="White" ForeColor="#284775" />
    <Columns>
        <asp:CommandField ShowDeleteButton="True" ShowEditButton="True" />
        <asp:BoundField DataField="ID_Product" HeaderText="ID_Product" ReadOnly="True"
            SortExpression="ID_Product" />
        <asp:BoundField DataField="Name_Product" HeaderText="Name_Product"
            SortExpression="Name_Product" />
        <asp:BoundField DataField="Price" HeaderText="Price" SortExpression="Price" />
        <asp:BoundField DataField="Details" HeaderText="Details"
            SortExpression="Details" />
        <asp:BoundField DataField="ID_Categories" HeaderText="ID_Categories"
            SortExpression="ID_Categories" />
    </Columns>
    <EditRowStyle BackColor="#999999" />
    <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
    <HeaderStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
    <PagerStyle BackColor="#284775" ForeColor="White" HorizontalAlign="Center" />
    <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
    <SelectedRowStyle BackColor="#E2DED6" Font-Bold="True" ForeColor="#333333" />
    <SortedAscendingCellStyle BackColor="#E9E7E2" />
    <SortedAscendingHeaderStyle BackColor="#506C8C" />
    <SortedDescendingCellStyle BackColor="#FFFDF8" />
    <SortedDescendingHeaderStyle BackColor="#6F8DAE" />
</asp:GridView>

```

3. Bước 3: Bỏ các cột không cần thiết

Xóa:

Với ví dụ này, ta có thể xóa cột khóa ngoại ID_Categories. Ta vào GridView Tasks chọn Edit Columns. Ở phần Selected fields, ta chọn cột cần bỏ và nhấn Remove. Ta sẽ được một giao diện gọn hơn:

	ID Product	Name Product	Price	Details
Edit Delete	prod001	Sony Vaio VGN5Z	500.0000	
Edit Delete	prod002	Dell Studio 1737	350.0000	
Edit Delete	prod003	HP Probook	450.0000	
Edit Delete	prod004	Dell E1505	250.0000	
Edit Delete	prod005	Kingmax 2Gb	70.0000	Bus 1066
Edit Delete	prod006	Kingston 1Gb	50.0000	DDR2 bus 800
Edit Delete	prod007	Kit 4Gb	100.0000	DDR2 Kit bus 1066
Edit Delete	prod008	Gygabyte G31 ES2L	150.0000	
Edit Delete	prod009	Asus P5 Pro	200.0000	
Edit Delete	prod010	ECS G31	80.0000	
12				

Thay thế hiển thị của cột:

Trong ví dụ này, ID_Categories sẽ không mang lại giá trị gì cho người dùng, ta sẽ làm hiển thị Name_Categories, và cung cấp một danh sách xổ xuống trong chế độ Edit cho phép người dùng dễ dàng chọn các giá trị ID_Categories.

Trong GridView, ta thay thế <asp:BoundField>

```

<Columns>
  <asp:CommandField ShowDeleteButton="True" ShowEditButton="True" />
  <asp:BoundField DataField="ID_Product" HeaderText="ID_Product" ReadOnly="True"
    SortExpression="ID_Product" />
  <asp:BoundField DataField="Name_Product" HeaderText="Name_Product"
    SortExpression="Name_Product" />
  <asp:BoundField DataField="Price" HeaderText="Price" SortExpression="Price" />
  <asp:BoundField DataField="Details" HeaderText="Details"
    SortExpression="Details" />
  <asp:BoundField DataField="ID_Categories" HeaderText="ID_Categories"
    SortExpression="ID_Categories" />
</Columns>

```

bởi <asp:TemplateField>. Trong TemplateField, ta có thể thêm bất kỳ nội dung nào để tùy biến cách hiển thị của cột:

```

<Columns>
  <asp:CommandField ShowDeleteButton="True" ShowEditButton="True" />
  <asp:BoundField DataField="ID_Product" HeaderText="ID_Product" ReadOnly="True" SortExpression=
  <asp:BoundField DataField="Name_Product" HeaderText="Name_Product" SortExpression="Name_Product
  <asp:BoundField DataField="Price" HeaderText="Price" SortExpression="Price" />
  <asp:BoundField DataField="Details" HeaderText="Details" SortExpression="Details" />
  <asp:TemplateField HeaderText="Name_Categories" SortExpression="Category.Name_Categories">
    <ItemTemplate>
      <%#Eval("Category.Name_Categories") %>
    </ItemTemplate>
  </asp:TemplateField>
</Columns>

```

Kết quả nhận được:

	ID Product	Name Product	Price	Details	Name Categories
Edit Delete	prod001	Sony Vaio VGN5Z	500.0000	Laptop	
Edit Delete	prod002	Dell Studio 1737	350.0000	Laptop	
Edit Delete	prod003	HP Probook	450.0000	Laptop	
Edit Delete	prod004	Dell E1505	250.0000	Laptop	
Edit Delete	prod005	Kingmax 2Gb	70.0000	Bus 1066	RAM
Edit Delete	prod006	Kingston 1Gb	50.0000	DDR2 bus 800	RAM
Edit Delete	prod007	Kit 4Gb	100.0000	DDR2 Kit bus 1066	RAM
Edit Delete	prod008	Gygabyte G31 ES2L	150.0000	Mainboard	
Edit Delete	prod009	Asus P5 Pro	200.0000	Mainboard	
Edit Delete	prod010	ECS G31	80.0000	Mainboard	
12					

Để Name_Categories là một danh sách xổ xuống, ta sẽ cấu hình tiếp một

LinqDataSource cho bảng Categories:

```

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

```

Ta sẽ tùy biến `<asp:TemplateField>` lại như sau:

```
<asp:TemplateField HeaderText="Name_Categories"
                  SortExpression="Category.Name_Categories">
    <ItemTemplate>
        <%#Eval("Category.Name_Categories") %>
    </ItemTemplate>
    <EditItemTemplate>
        <asp:DropDownList ID="DropDownList1"
                        DataSourceID="CategoryDataSource"
                        DataValueField="ID_Categories"
                        DataTextField="Name_Categories"
                        SelectedValue='<%#Bind("ID_Categories") %>'
                        runat="server" />
    </EditItemTemplate>
</asp:TemplateField>
```

Và thực thi chọn Edit ta sẽ được:

	ID Product	Name Product	Price	Details	Name Categories
Update Cancel	prod001	Sony Vaio VGN5Z	500.0000		Laptop
Edit Delete	prod002	Dell Studio 1737	350.0000		Mainboard
Edit Delete	prod003	HP Probook	450.0000		RAM
Edit Delete	prod004	Dell E1505	250.0000		HDD
Edit Delete	prod005	Kingmax 2Gb	70.0000	Bus 1066	UPS
Edit Delete	prod006	Kingston 1Gb	50.0000	DDR2 bus 800	PSU
Edit Delete	prod007	Kit 4Gb	100.0000	DDR2 Kit bus 1066	CASE
Edit Delete	prod008	Gygabyte G31 ES2L	150.0000		Laptop
Edit Delete	prod009	Asus P5 Pro	200.0000		Computer
Edit Delete	prod010	ECS G31	80.0000		Keyboard
					Mouse
					Mainboard
					Mainboard
					Mainboard

III.4.3 Dùng `<asp:LinqDataSource>` với mệnh đề `<where>` được khai

báo:

Ta đã có một mô hình dữ liệu LINQ to SQL của CSDL DemoLINQ, 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 loại nào đó.

Ta có thể trỏ một control <asp:gridview> đến datasource đã tạo và cho phép phân trang, chỉnh sửa và sắp xếp(như các bước trên). Ở phần <asp:LinqDataSource> ta sẽ thêm một <WhereParameter> và có thể cấu hình như sau: (ở đây ta sẽ hiển thị các sản phẩm có cùng mã loại “cate001”):

```
<asp:LinqDataSource ID="ProductsDataSource" runat="server"
    ContextTypeName="DemoLINQ.DemoLINQDataContext" EnableDelete="True"
    EnableInsert="True" EnableUpdate="True" TableName="Products"
    Where="ID_Categories==@ID_Categories">
    <WhereParameters>
        <asp:QueryStringParameter DefaultValue="cate001"
            Name="ID_Categories" QueryStringField="ID_Categories" Type="String" />
    </WhereParameters>
</asp:LinqDataSource>
```

Kết quả hiển thu được:

PRODUCTS

	ID Product	Name Product	Price	Details	ID Categories
Edit Delete	prod001	Sony Vaio VGN5Z	500.0000		cate001
Edit Delete	prod008	Gygabyte G31 ES2L	150.0000		cate001
Edit Delete	prod009	Asus P5 Pro	200.0000		cate001
Edit Delete	prod010	ECS G31	80.0000		cate001

Dùng cách khai báo các tham số cho where giống như trên có thể làm việc tốt trong hầu hết trường hợp.

Tóm lại: Control <asp:LinqDataSource> giúp việc gắn nối bất kỳ control ASP.NET vào một mô hình dữ liệu LINQ to SQL một cách dễ dàng. Nó cho phép các control dùng hiển thị giao diện có thể vừa lấy dữ liệu từ LINQ to SQL, cũng như thao tác thêm/ xóa/ sửa vào mô hình dữ liệu.

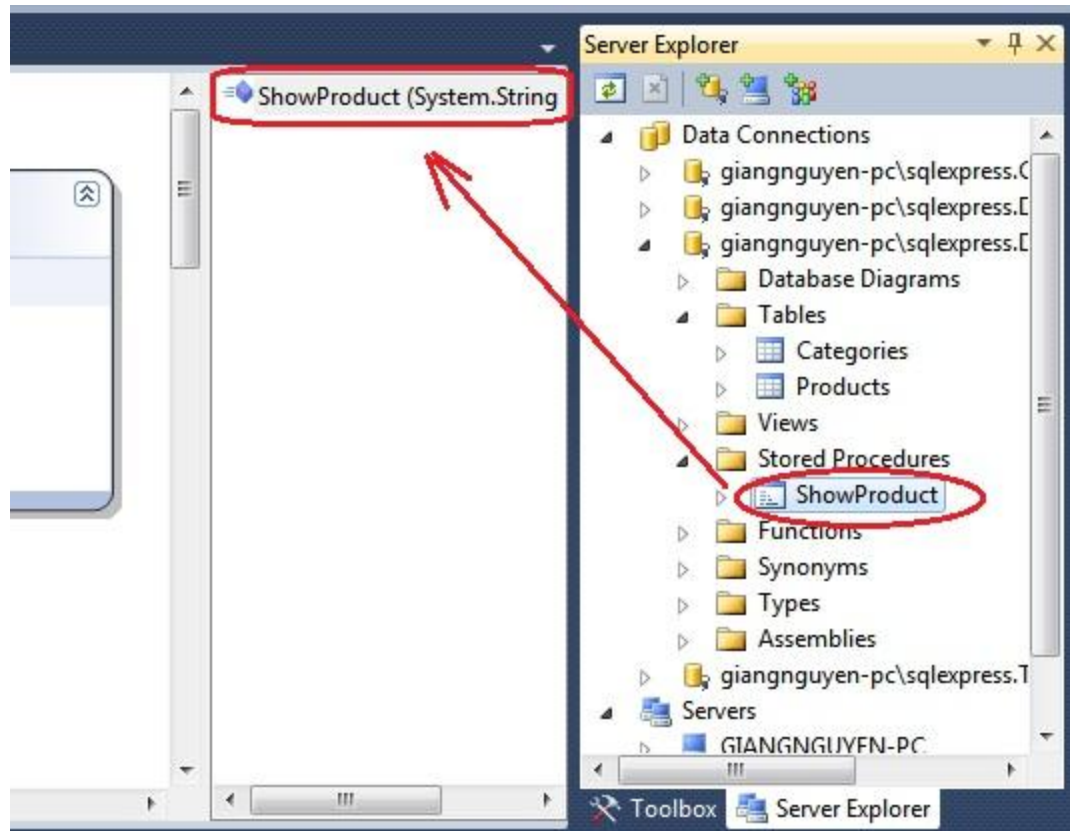
III.5 Lấy dữ liệu dùng Stored Procedure (SPROC):

III.5.1 Sử dụng SPROC vào một DataContext của LINQ:

Trong Server Explorer, ta kéo thả một Stored có sẵn hoặc mới được tạo vào cửa sổ định nghĩa mô hình dữ liệu. Ở đây ta có 1 Proc hiển thị thông tin tên Sản phẩm và giá cả theo Mã sản phẩm đưa vào (chọn cate001):

```
Create Proc ShowProduct
@ID_Categories varchar(50)
as
select Name_Product, Price from Products
where ID_Categories=@ID_Categories
```

Việc này sẽ tự động sinh ra một thủ tục trong lớp DataContext của LINQ to SQL:



Mặc nhiên, tên của phương thức được tạo trong DataContext chính là tên SPROC. Ta cũng có thể đổi tên cho phương thức bằng cách chọn Property của nó. Ta sẽ sử dụng một control GridView để hiển thị ra màn hình. Và gọi Proc như sau:

```
protected void Page_Load(object sender, EventArgs e)
{
    DemoLINQDataContext pr=new DemoLINQDataContext();
    GridView1.DataSource = pr.ShowProduct("cate001");
    GridView1.DataBind();
}
```

Kết quả thu được:

PRODUCTS

Name_Product	Price
Gygabyte G31 ES2L	150.0000
Asus P5 Pro	200.0000
ECS G31	80.0000

III.5.2 Ánh xạ kiểu trả về của phương thức SPROC vào một lớp trong mô hình dữ liệu:

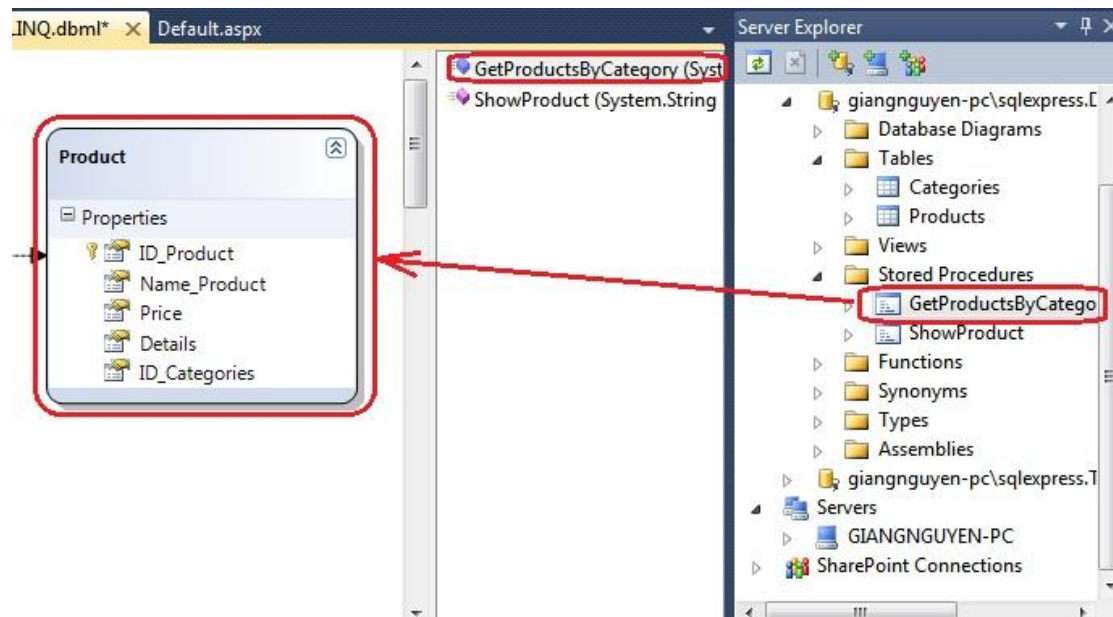
Trong ví dụ trên, thủ tục ShowProduct trả về một danh sách dữ liệu gồm 2 cột: Name_Product và Price. LINQ to SQL sẽ tự động tạo ra một lớp có tên ShowProduct để biểu diễn kết quả này.

Chúng ta cũng có thể chọn cách gán kiểu trả về của thủ tục là một lớp thực thể trong mô hình dữ liệu, ví dụ Products hay Categories.

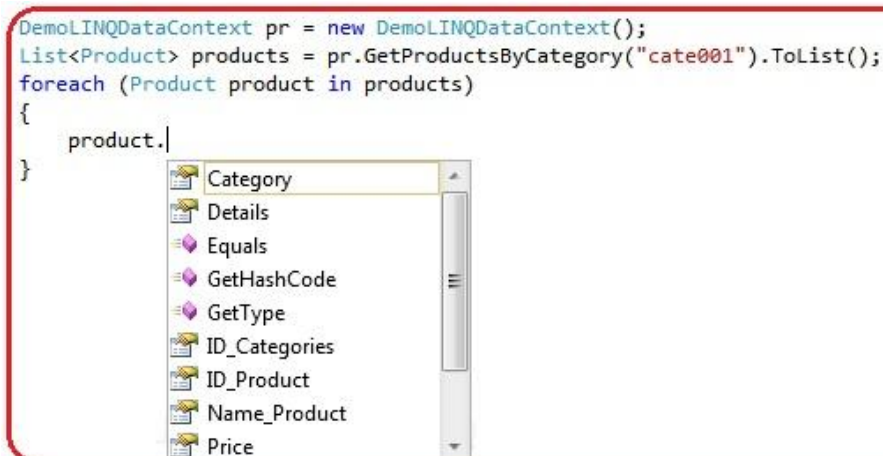
Ví dụ ta có một thủ tục GetProductsByCategory trong CSDL trả về thông tin sản phẩm như sau:

```
Create Proc GetProductsByCategory
    @ID_Categories varchar(50)
As
    Select * from Products where ID_Categories= @ID_Categories
```

Thay vì kéo thả Proc vào một vị trí bất kỳ, ta sẽ thả nó lên trên lớp Product mà ta đã tạo sẵn trên cửa sổ này.



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



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.

III.5.3 Xử lý 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#), và với các tham trị.

Ví dụ, thủ tục “GetProductName” sau sẽ nhận vào một ID_Product như tham số đầu vào, và trả về tên sản phẩm như một tham số dạng OUTPUT:

```
Create Proc GetProductName
(
    @ID_Product varchar(50),
    @Name_Product varchar(50) output
)
As
    Select @Name_Product=Name_Product from Products
    where ID_Product=@ID_Product
```

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

```
string nameProduct= "";
DemoLINQDataContext pr = new DemoLINQDataContext();
List<Product> products =
    pr.GetProductName("cate001", ref nameProduct).ToList();
```

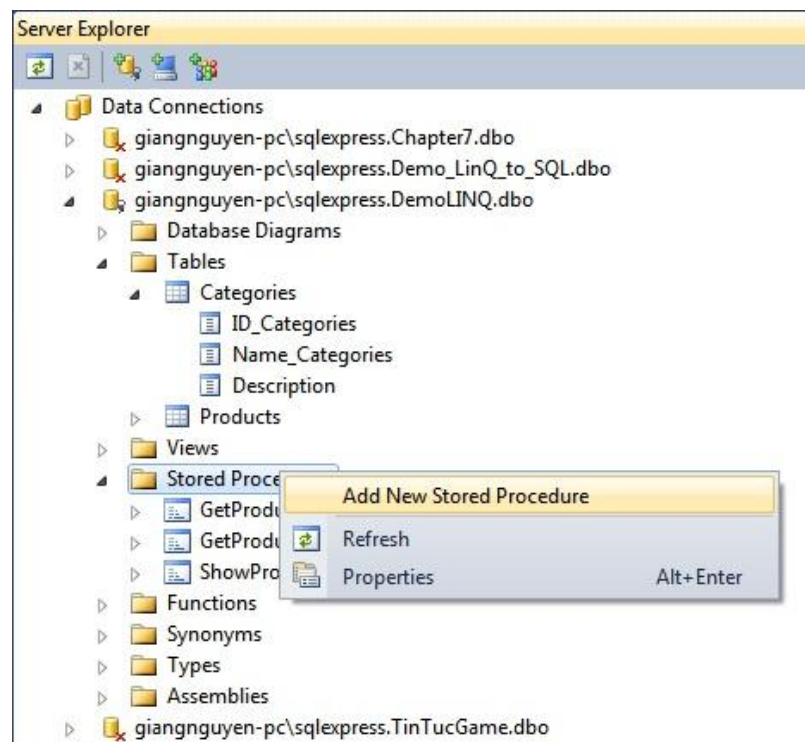
Đây là một số cách dùng các thủ tục SPROC để dễ dàng truy xuất cũng như cập nhật các lớp mô hình dữ liệu.

III.5.4 Cập nhật dữ liệu dùng Stored Procedure

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 SQL động. Các quy tắc kiểm tra cũng hoàn toàn tương tự. Thế nên ở đây ta chỉ xét trường hợp Insert, các trường hợp còn lại cũng tương tự.

Thêm một Categories dùng SPRO:

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

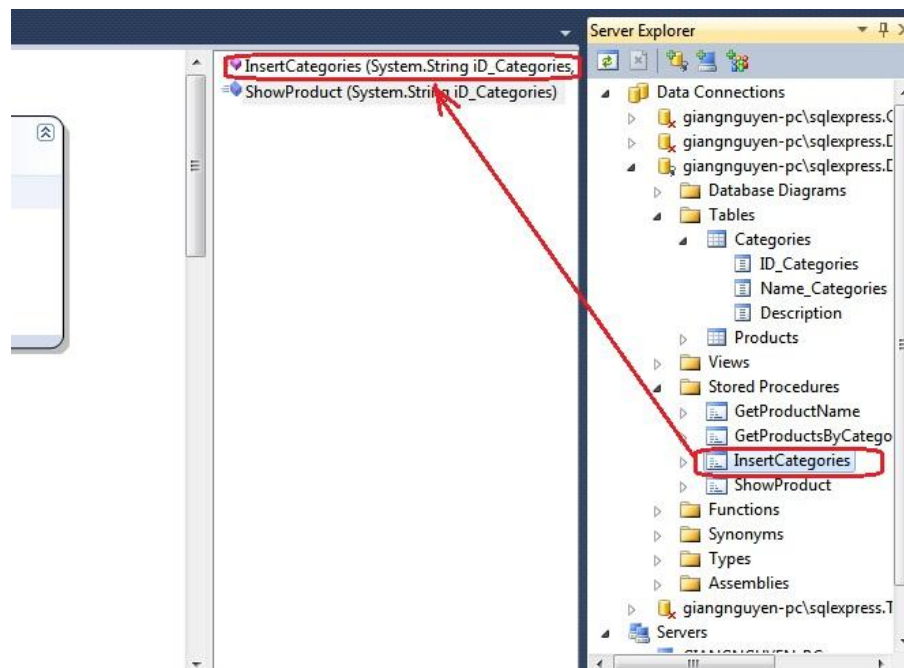


Sau đó ta tạo thêm một thủ tục có tên "InsertCategories":

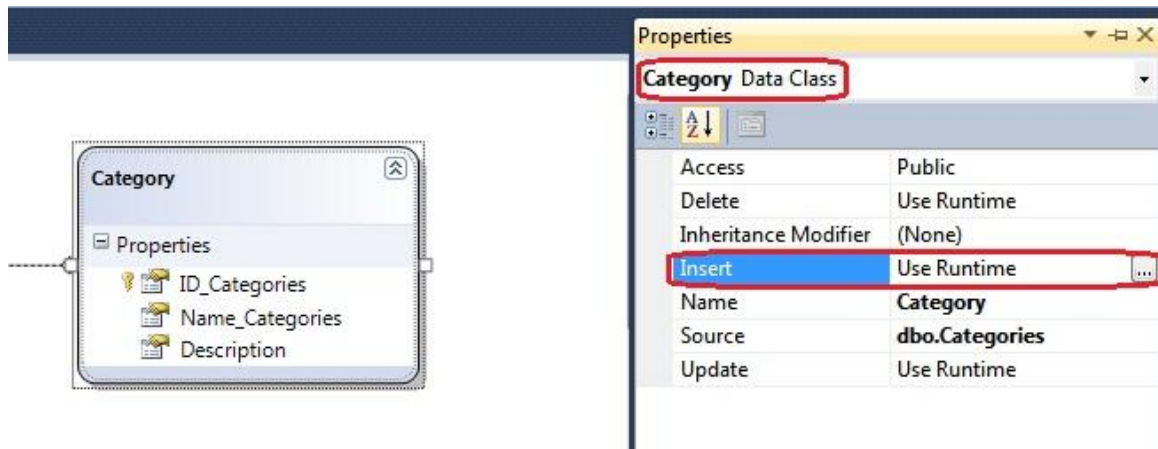

```
dbo.InsertCategori...express.DemoLINQ)* X Default2.aspx dbo.StoredProcedu...express.DemoLINQ)

CREATE PROCEDURE dbo.InsertCategories
(
    @ID_Categories varchar(50),
    @Name_Categories nvarchar(MAX),
    @Desc ntext
)
AS
Insert into Categories values(@ID_Categories, @Name_Categories, @Desc)
```

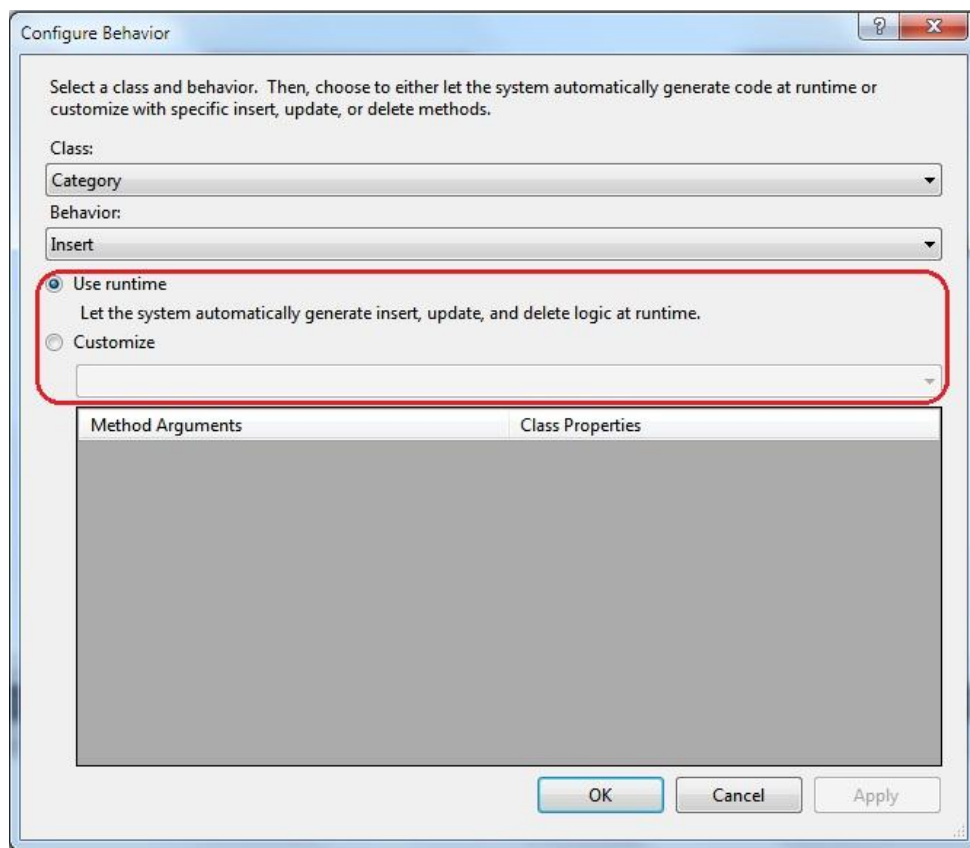
Sau khi tạo SPROC, ta sẽ mở LINQ to SQL designer của lớp truy cập dữ liệu và kéo/thả SPROC InsertCategories từ Server Explorer lên trên màn hình chính của trình thiết kế:



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

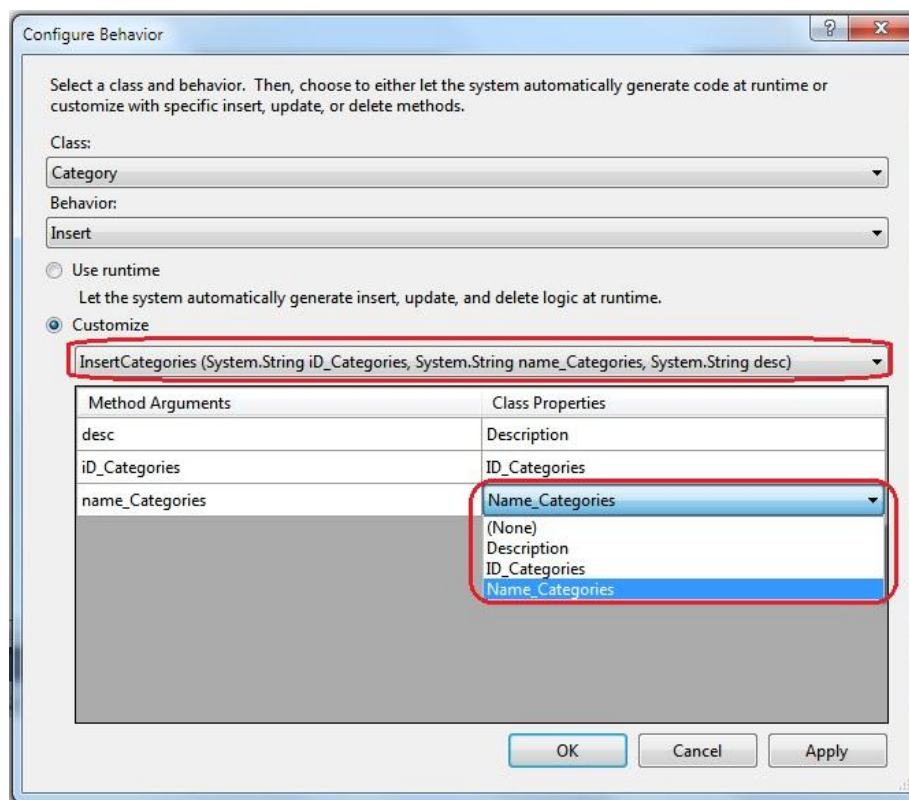


Khi nhấn nút này, cửa sổ sau sẽ hiện ra để có thể tùy biến hành vi Insert:



Nếu ta 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. Ta có thể thay đổi bằng

cách nhấn chuột vào Customize và chọn thủ tục InsertCategories 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 Category và các tham số của InsertCategories. Mặc nhiên, LINQ to cũng tự động xác định các tham số tương ứng theo tên, tuy nhiên ta vẫn có thể sửa lại nếu muốn.

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

III.6 Thực thi các biểu thức SQL tùy biến:

III.6.1 Dùng các câu truy vấn SQL tùy biến với LINQ to SQL

Trong ví dụ mẫu ở trên 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 và thực thi nó trong CSDL.

Nhưng nếu 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. Một cách khác là dùng phương thức “ExecuteQuery” trong lớp DataContext để thực thi một câu SQL do ta cung cấp.

III.6.2 Dùng ExecuteQuery:

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

Điều làm cho ExecuteQuery thực sự hữu dụng là nó cho phép ta chỉ ra cách nó trả về dữ liệu. Ta 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.

III.6.3 Tùy biến các biểu thức SQL:

Mặc nhiên, khi ta lấy về một mô hình dữ liệu dùng LINQ to SQL, nó sẽ lưu lại các thay đổi mà ta làm. Nếu gọi phương thức “SubmitChanges()” trên lớp DataContext, nó sẽ lưu lại các thay đổi vào CSDL.

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

III.6.4 Tùy biến các câu SQL cho Inserts/ Updates/ Deletes:

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

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 ta muốn thay đổi. Và sau đó ta có thể dùng phương thức ExecuteCommand để thực thi các câu SQL cần thiết.

Tóm lại:

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 ta muốn kiểm soát hoàn toàn câu lệnh SQL được thực thi, ta có thể dùng các thủ tục SPROC, hay cũng có thể viết các câu SQL. Điều này cung cấp khả năng tùy biến mạnh mẽ.

IV. Tổng kết:

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

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.

Với LINQ to SQL, nó cho phép ta 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. Ta 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 cũng có thể cấu hình lớp dữ liệu để dùng SPROC. Điều hay là các đoạn lệnh của ta 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.

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.

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`.

V. Phụ lục:

Tài liệu tham khảo:

[1]. <http://msdn.microsoft.com/en-us/library/bb386976.aspx>

[2]. LINQ to SQL-Tutorial dịch bởi Đào Hải Nam

