Linq To Sql Part.3

数据库查询

ScottGu

译者：张子阳

jimmyzhang.cnblogs.com

jimmy_dev@163.com

出处：Linq To Sql (Part.3 - Querying our database)

术语表

Built-in：内置的
Clause：子句
Debugger：调试器
Object Relational Mapper：对象关系映射器
ORM(Object Relation Mapping)：对象关系映射
Visualizer：查看器
plug-in：插件程序
Breakpoint：断点
Shape：构造
object initialization：对象初始化
deferred execution model：延迟执行模型
sequences：序列
Object Initializer：对象初始化器
Collection Initializers：集合初始化器

上个月，我开始发表一个介绍 LINQ TO SQL 的随笔系列。LINQ TO SQL 是一个内置于.Net 框架 3.5 版本的 O/RM (对象关系映射)框架，它使你可以方便地使用.Net 类对关系数据库进行建模。你可以使用 LINQ 表达式来对数据库进行查询、添加、编辑、删除。

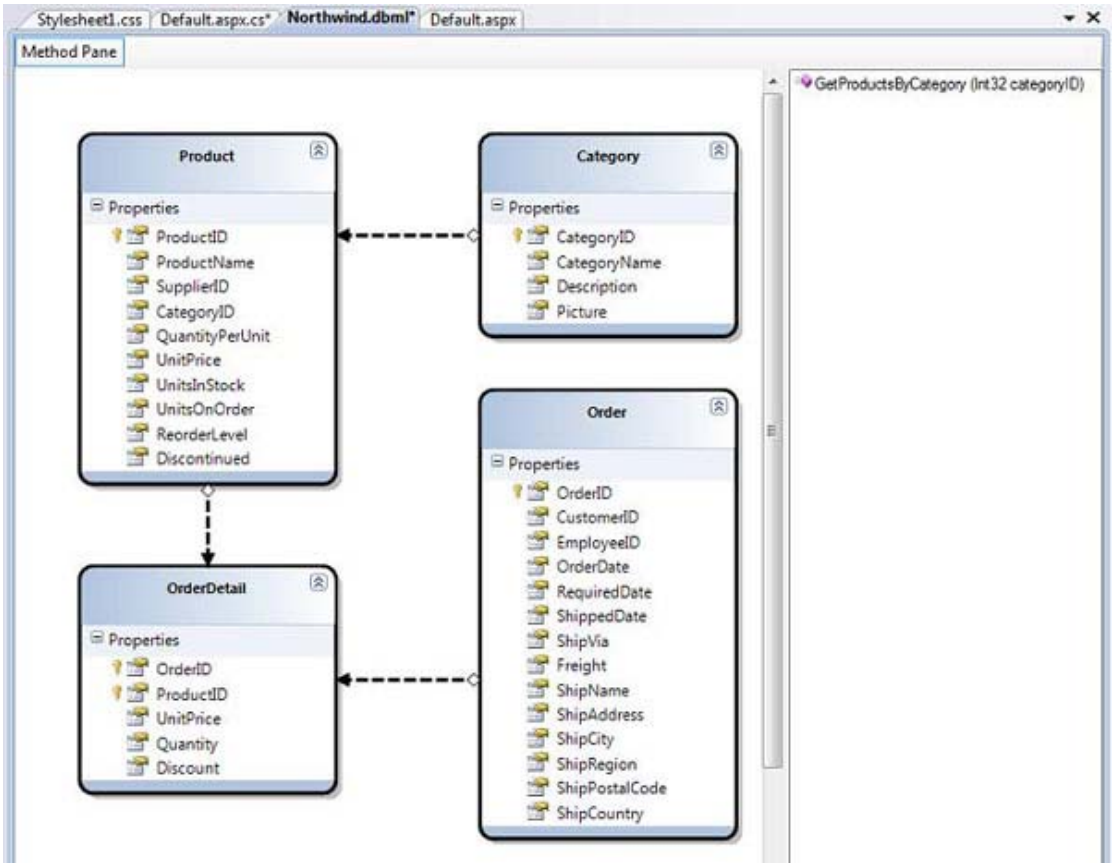
下面是我这系列随笔的前两篇：

- Part 1：LINQ TO SQL 介绍
- Part 2：定义我们的数据模型类

在今天这篇随笔中，我将继续详细为大家介绍如何使用我们在第二篇随笔中创建的这个数据模型，演示如何使用 Asp.Net 项目来对数据进行查询。

使用 LINQ TO SQL 建模了的 Northwind 数据库

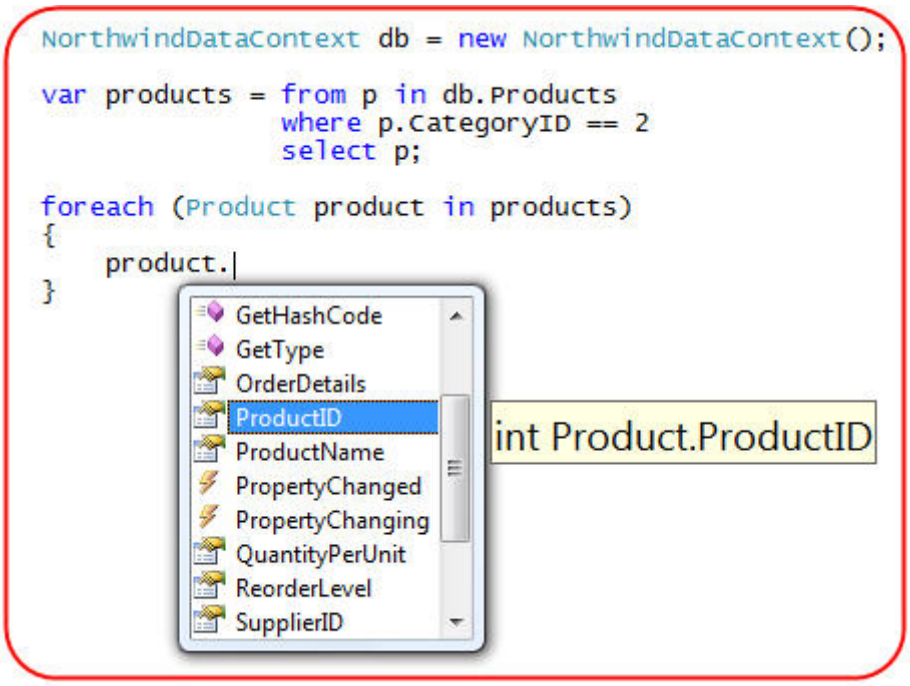
在这一系列随笔中的第二篇，我一步步讲解了如何使用内置于 VS2008 中的 LINT TO SQL 设计器创建一个 LINQ TO SQL 类模型。下面是我们为 Northwind 范例数据库创建的类模型。



获取 Products

一旦我们定义了上面的数据模型类，我们可以方便的从我们的数据库中进行查询和获取数据。LINQ TO SQL 通过对使用 LINQ TO SQL 设计器创建的 NorthwindDataContext 类编写 LINQ 查询语句来完成。

举个例子，如果想要获取一系列的 Products 对象，我可以像下面这个编写代码：



在上面的语句中，我在 LINQ 查询语句中使用了一个“Where”子句以返回属于特定 Category 的 Products。我使用 CategoryId 来进行筛选。

LINQ TO SQL 的优点之一是在定义如何查询数据的时候具有很大的灵活性，而且我还可以利用我在建立 LINQ TO SQL 数据类时建立的关系对数据库进行更加丰富和常见的查询。举个例子，我可以将这个查询修改成根据 Product 的 CategoryName 来实行，而不是像上面那样根据 CategoryId，就像下面这样。

```
NorthwindDataContext db = new NorthwindDataContext();

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

注意上面，我是如何使用 Product 的“ Category”属性去筛选 Products，这些 Products 都属于拥有特定 CategoryName 的 Category。这个属性由 LINQ TO SQL 自动为我们创建，因为我们在为 Category 和 Product 类建模的时候它们之间在数据库中拥有一对多的关系。

为了举一个在查询中使用我们数据模型的相联关系的例子，我们可以使用下面的 LINQ 查询语法，以获得那些有 5 个或者更多订单的 Products。

```
NorthwindDataContext db = new NorthwindDataContext();

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

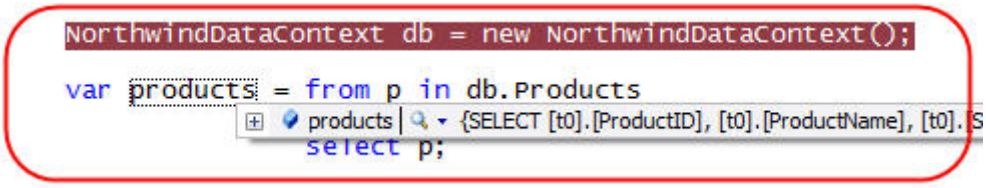
注意上面，我们是如何使用 LINQ TO SQL 为我们在 Product 类上创建的“OrderDetails”集合(这是因为我们在使用 LINQ TO SQL 设计器时会有一个一对多关系)。

在调试时显示 LINQ TO SQL 的实际查询代码

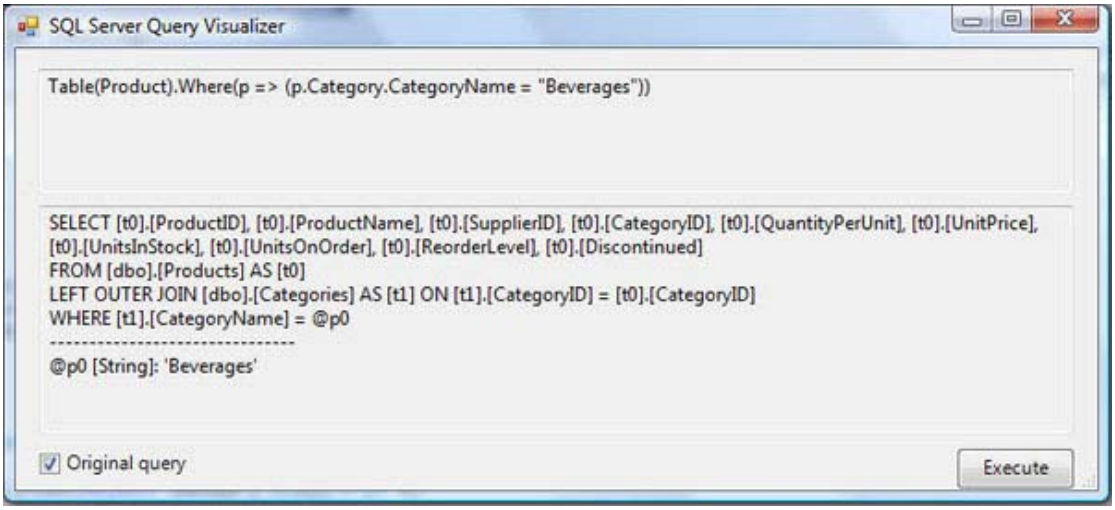
当你在对你的对象进行查询或更新的时候，LINQ TO SQL 对象关系映射器会自动进行创建、执行合适的 SQL 代码的操作。

开发者对于这种新的 ORM 最关心或者最恐惧的一个之一问题是“实际执行的是什么 SQL 代码？”。当你在调试你的应用程序时，LINQ 一个非常好的特性就是它可以非常容易地查看实际执行的究竟是什么 SQL 代码。

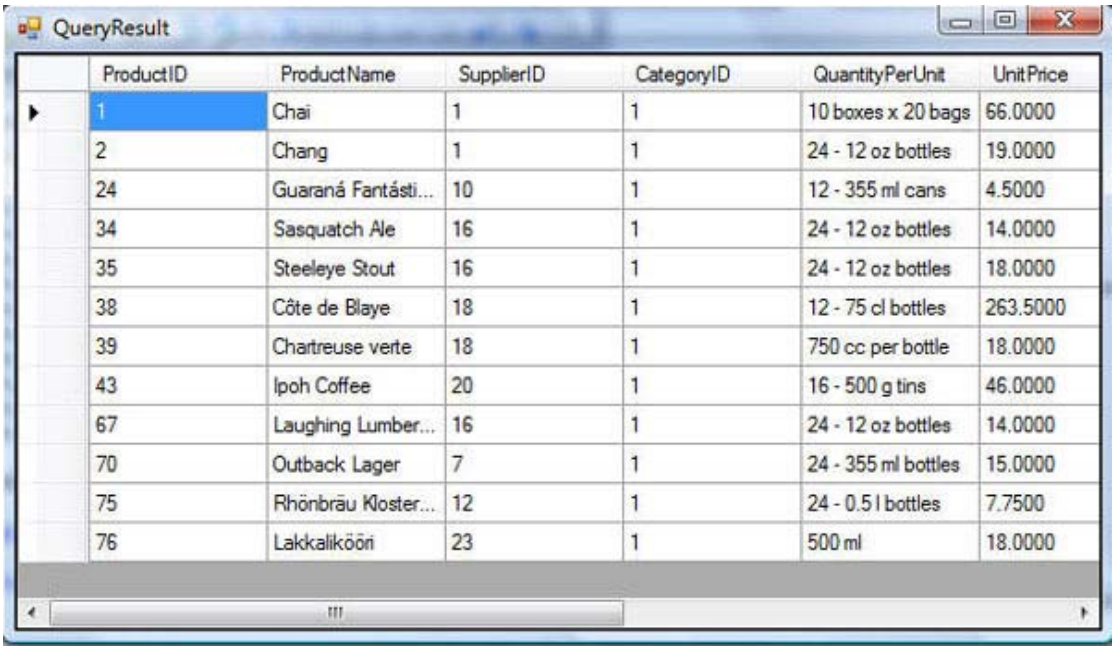
从 Visual Studio 2008 的 Beta2 版本以后，你可以使用新的插件程序 LINQ TO SQL 查看器容易的查看（以及测试）任何 LINQ TO SQL 查询表达式。简单的在 LINQ TO SQL 查询上设置一个断点，然后单击放大图标来开启调试器的表达式查看器。



通过这样，你会看到一个对话框，上面显示了 LINQ TO SQL 在执行获取 Product 对象操作时的实际 SQL 代码：



如果你在这个对话框上点击“Execute”按钮，将会使用调试器直接执行这段 SQL 代码，并且看到从数据库中返回的实际结果集。



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

很显然，这样将非常容易准确地知道 LINQ TO SQL 为你执行的是什么 SQL 语句。值得注意到是在你需要做些改动的时候，你可以有选择的覆盖这个 LINQ TO SQL 所执行的 SQL 语句 - 尽管在 98%的情况下，我想你会发现 LINQ TO SQL 所将执行的 SQL 代码是非常、非常好的。

绑定 LINQ TO SQL 查询到 Asp.Net 控件

LINQ 查询返回实现了 Ienumerable 接口的结果，这个接口也是 Asp.Net 服务器控件支持对象绑定的接口。这就意味着，你可以绑定任何 LINQ，LINQ TO SQL，或者 LINQ TO XML 查询的结果到任何的 Asp.Net 控件。

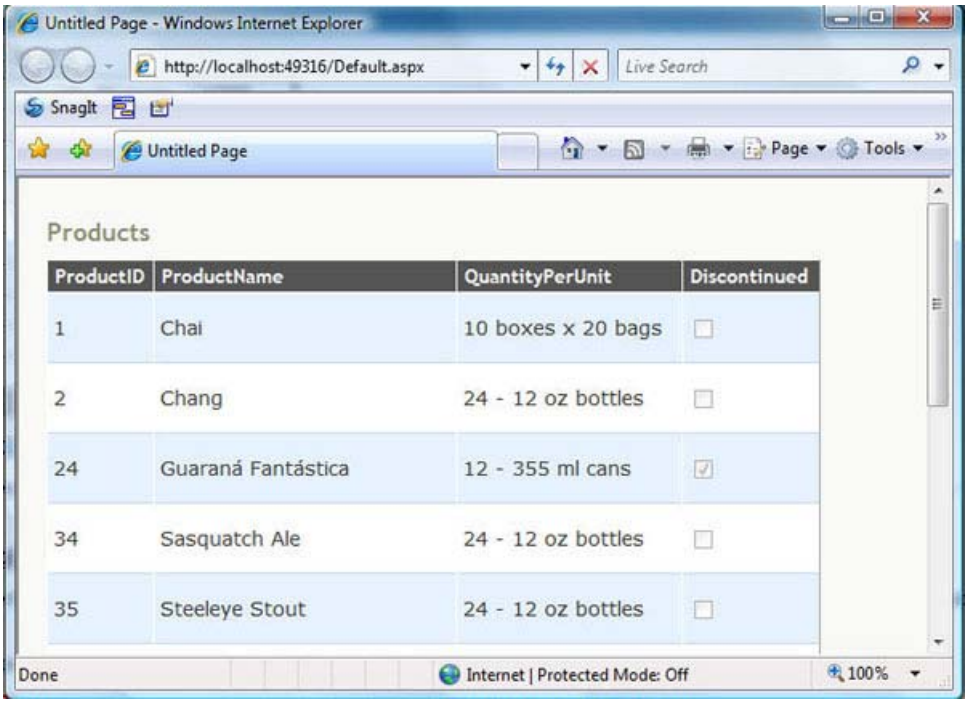
举个例子，你可以像下面这样在.aspx 页面中声明一个<asp:gridview>的控件，

```
<h3>Products</h3>

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

然后，可以像下面这样绑定我们已经写好的 LINQ TO SQL 查询的结果到 GridView 中：

```
NorthwindDataContext db = new NorthwindDataContext();

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

GridView1.DataSource = products;
GridView1.DataBind();
```

这样会产生像下面这样的一个页面：



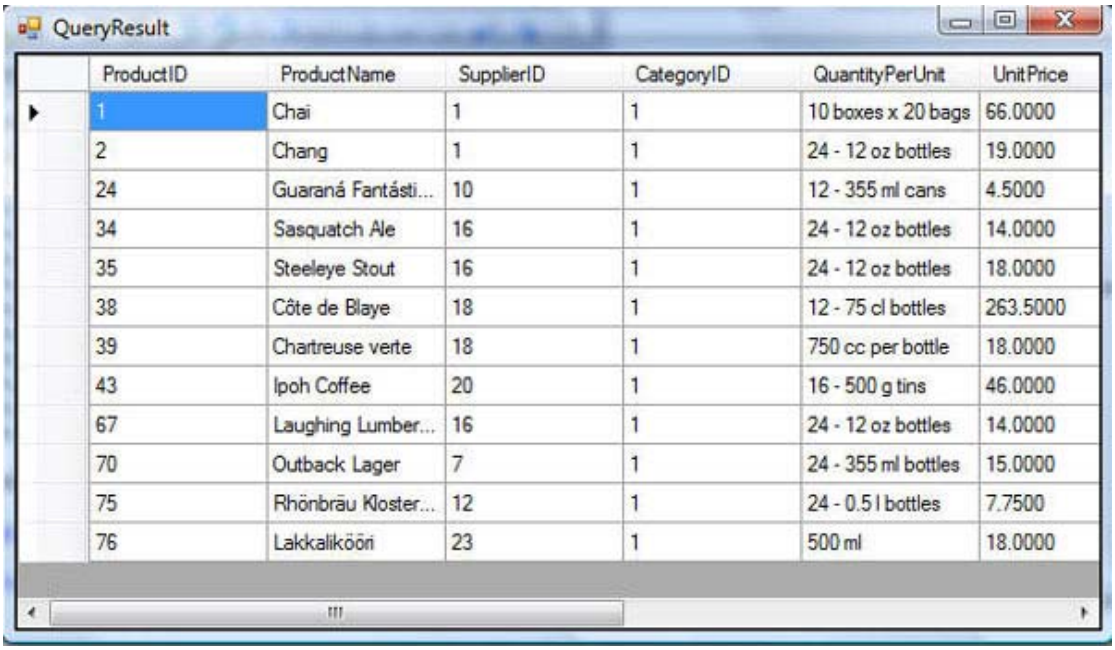
构造我们的查询结果

现在当我们运行我们的 Product 查询时，默认地，我们将获取所有所需列的数据以填充 Product 实体类。

举个例子，这个查询获取 Products：

```
NorthwindDataContext db = new NorthwindDataContext();

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

结果是所有的数据都被返回了。

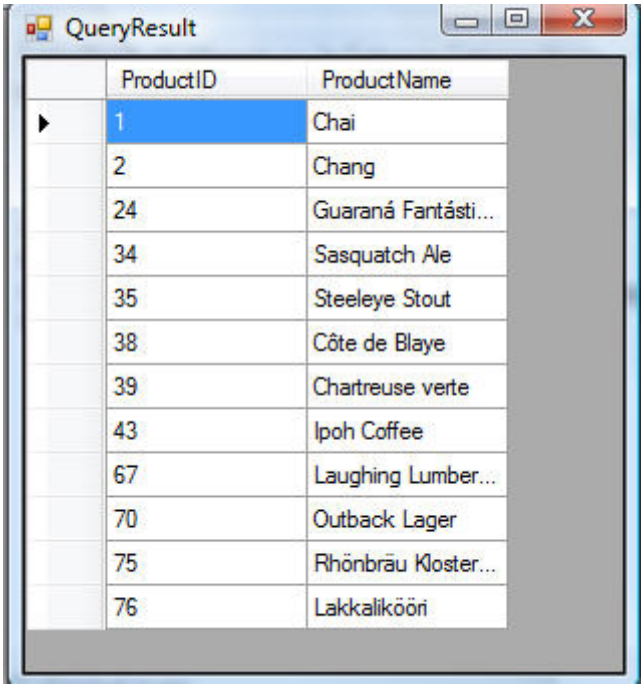


ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice
1	Chai	1	1	10 boxes x 20 bags	66.0000
2	Chang	1	1	24 - 12 oz bottles	19.0000
24	Guaraná Fantásti...	10	1	12 - 355 ml cans	4.5000
34	Sasquatch Ale	16	1	24 - 12 oz bottles	14.0000
35	Steeleye Stout	16	1	24 - 12 oz bottles	18.0000
38	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

通常，我们仅仅想要返回每个Product数据的一个子集。我们可以使用新的、LINQ 和新C#/VB编译器所支持的数据构造特性去指明我们仅仅想要一个数据的子集，这个可以通过像下面这样修改我们的LINQ TO SQL查询来完成。

```
NorthwindDataContext db = new NorthwindDataContext();

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

这样从我们的数据库将仅返回这些数据(如同我们的 debug 查看器所显示的)。

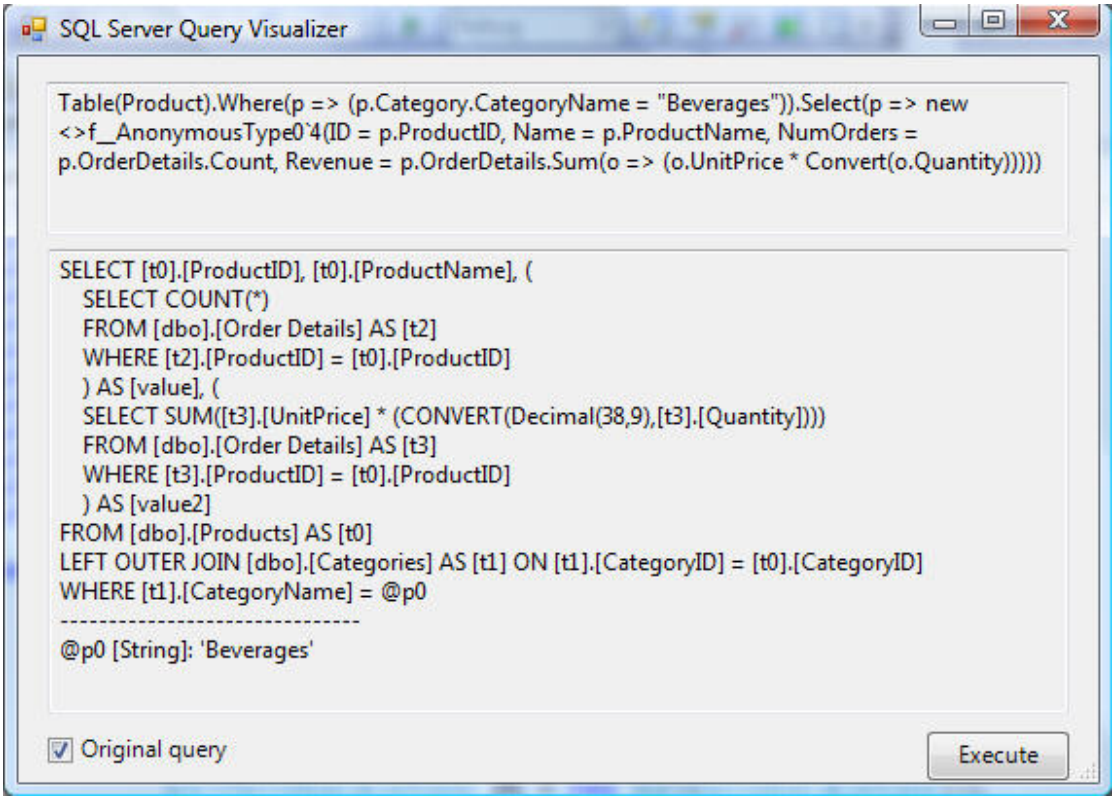


LINQ TO SQL 比较酷的地方是：在构造数据的时候，可以完全利用数据模型类的联系。这就使我可以写出非常有用(也非常高效)的数据查询。举个例子，下面的查询获取来自 Product 实体的 ID 和 Name，此 Product 下的订单的总数，然后结算每个 Product 的订单的税金总和。

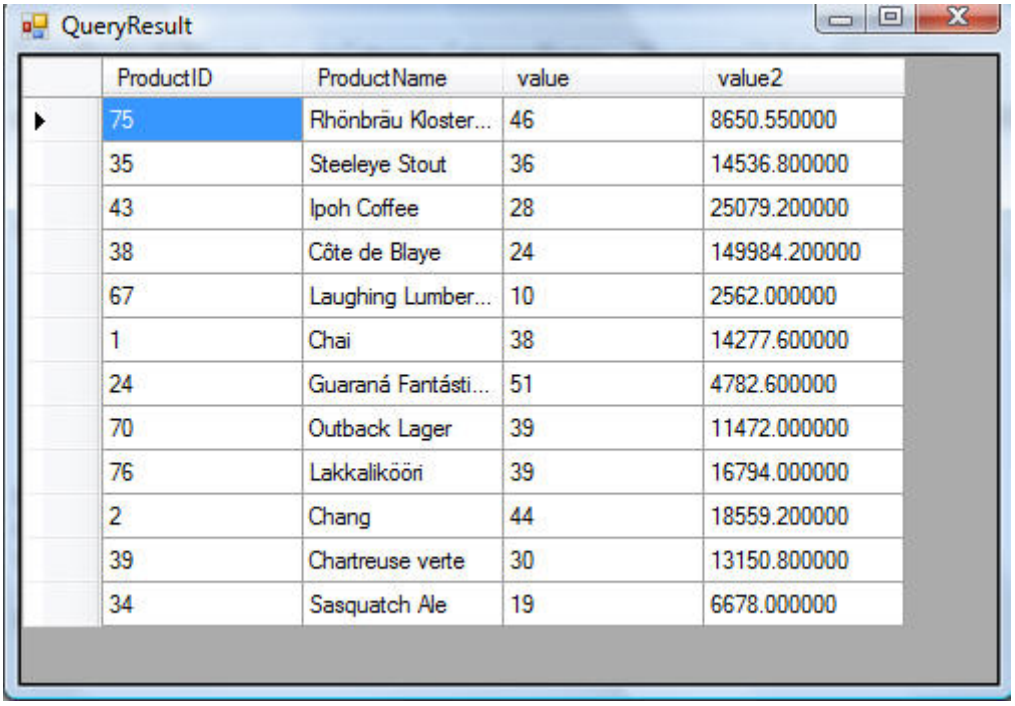
```
NorthwindDataContext db = new NorthwindDataContext();

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)
               };
```

上面“Revenue”属性右侧的表达式是使用由LINQ所提供的“Sum”扩展方法的一个例子。它使用一个返回每个产品订单项的值的 Lambada 表达式 作为一个参数。

LINQ TO SQL 是非常智能的，并可以将上面的 LINQ 表达式在求值时 转换成下面的 SQL 语法(如果我们的 debug 查看器所显示的)。

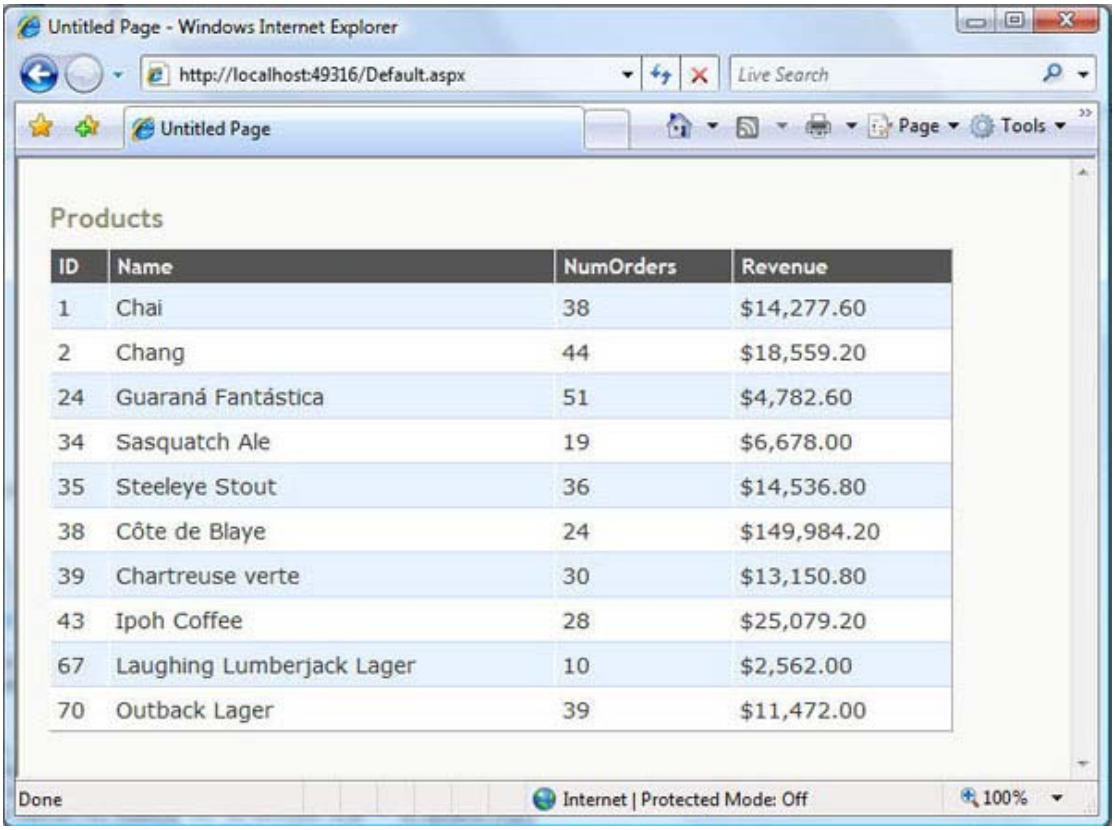


上面的 SQL 将会在 SQL Server 上对所有的 NumOrders 和 Revenue 的值进行计算，并仅从数据库获取下面所显示的数据(这使得它运行得非常快)。



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

我们可以将查询结果绑定到 Gridview 控件上以产生一个不错的用户界面。

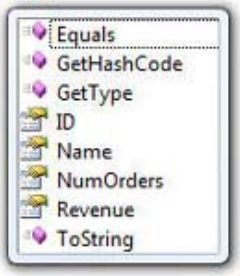


顺便说一下，以免你不断猜测，当你在写这些 LINQ 构造查询时，你可以在 VS2008 中获得完全的智能提示。

```
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.)
               };
```


在上面的例子中，我声明了一个使用对象初始化去构造和定义结果结构的匿名类型。真正酷的地方是：在同这些匿名结果序列打交道的时候，VS2008 同样提供了完全的智能提示、编译检测和 refactoring 支持。

```
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.
}
```



对查询结果进行分页

Web 环境中最常见的一个需求就是高效地对用户界面层数据进行分页。LINQ 对两个使分页又简单又高效的扩展方法提供内置的支持，这两个方法是 Skip() 和 Take()。

我们可以使用下面的 Skip()和 Take()方法，指明我们仅想返回 10 个 Product 对象 - 从我们定义的作为参数的初始 Product 行开始。

```
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();
}
```

注意上面，我没有将 Skip()和 Take()添加到初始的 Products 查询声明上，而是在后面添加到查询上(当绑定到 GridView 数据源时)。常有人问我“这样不是说查询首先从数据库中返回所有的数据，然后在(不好的)中间层中再进行分页？”不会的。原因是 LINQ 使用一种延迟执行模型 - 这就意味着查询并不实际的执行，直到你试图遍历结果的时候。

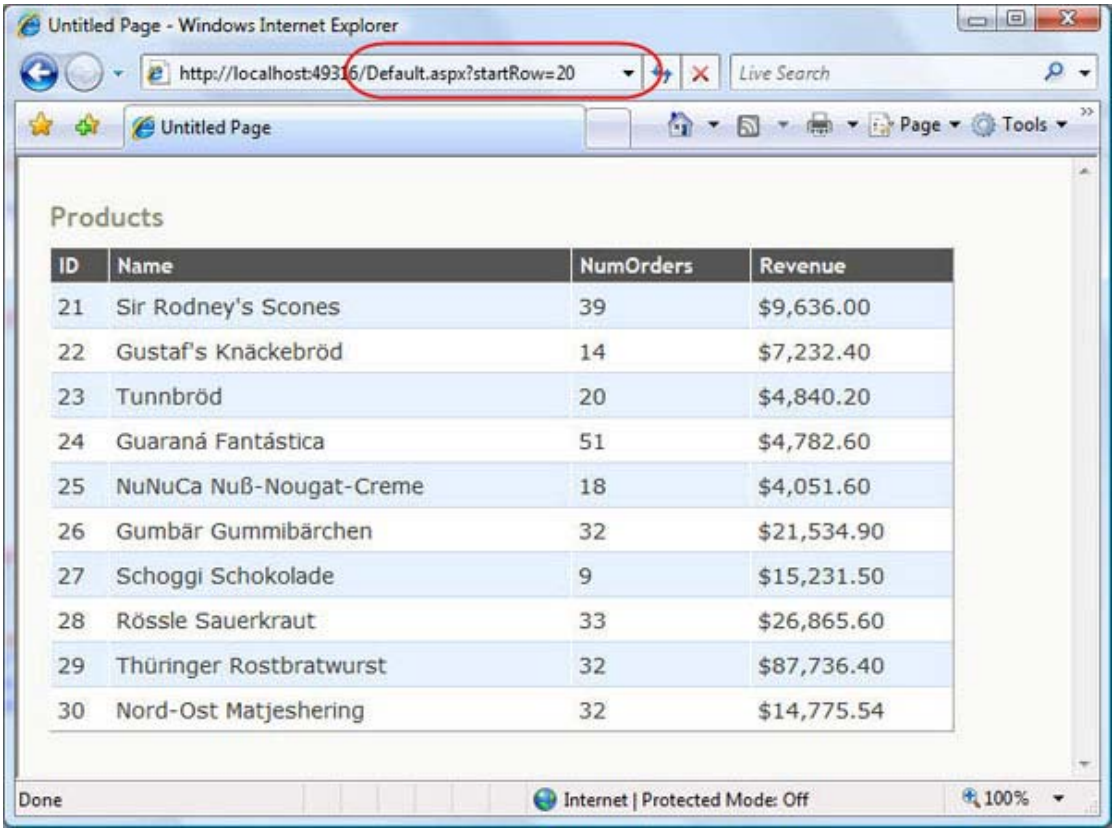
这种延迟执行模型的好处之一是，它可以让你在代码语句间非常好地进行查询组合(这样就

提高了代码可读性)。它同样可以使你从其他查询中组合查询 - 这样就使一些非常灵活的查询组合和重用的情况成为可能。

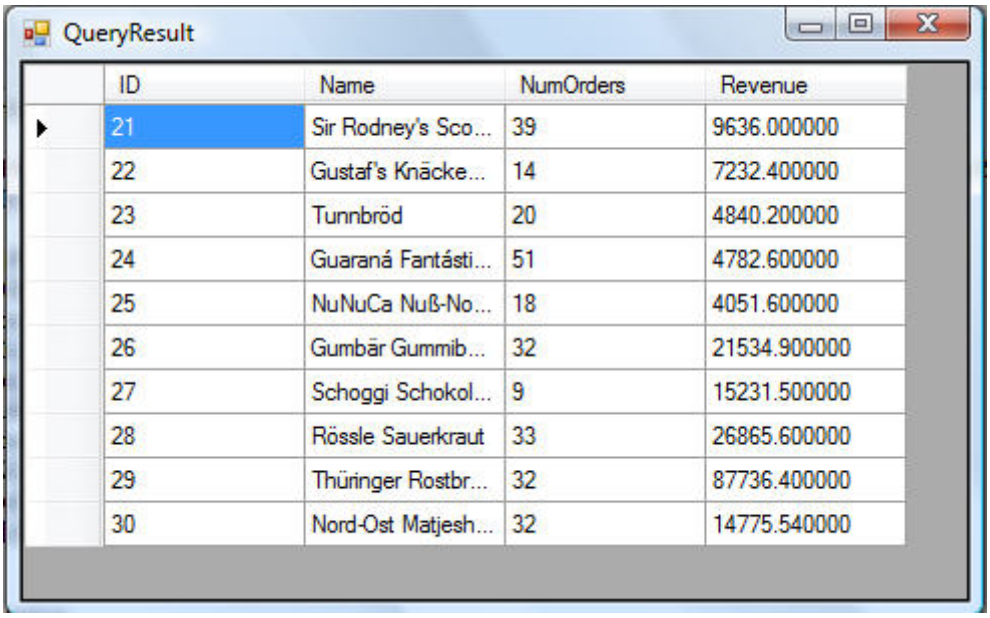
一旦我定义好上面的 BindProduct()方法，我可以在页面中写出下面的代码以获得来自 Querystring 的起始索引，并让 gridview 中的 Products 分页显示。

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

    BindProducts(startRow);
}
```

这样将呈现给我们一个 Products 页面，筛选出那些有 5 个或者以上订单的 Products，动态显示计算的 Product 数据，并且可以通过 querystring 参数来进行分页。



注意：当使用 SQL 2005 进行开发的时候，LINQ TO SQL 将使用 ROW_NUMBER() 函数来进行数据库中的所有分页逻辑。这就确保了在执行上面的代码时，当前显示页只有 10 行我们所需要的数据从数据库中返回。



这就使得对大型数据序列进行分页的时候更加高效和容易。

总结

对于 LINQ TO SQL 提供的一些很酷的数据查询方式，希望上面的这些步骤能提供一个好的概览。想要学习更多的 LINQ 表达式和由 VS2008 中的 C#、VB 编译器所支持的新语言语法，请阅读下面这些我更早时发布的随笔：

- 自动属性，对象初始化器 和 集合初始化器
- 扩展方法
- Lamabada表达式
- 查询语法
- 匿名类型

在我 LINQ TO SQL 系列的下一篇随笔中，我将介绍我们如何清楚地对我们的数据模型类添加验证逻辑，并且演示我们如何使用它封装业务层逻辑，这些业务逻辑在每次我们进行更新、插入或者删除数据的时候都会运行。然后我将解释更高级的 Lazy 和 eager 加载查询场景(Lazy and eager loading query scenarios)，如何使用新的<asp:LINQDataSource>控件来支持 Asp.Net 控件的声明式数据绑定，乐观并发错误处理，以及其他更多内容。