Cuando EntityDataSource y Cuando LinqDataSouce

* Use LinqDataSource cuando se trata de una sola table
* Use EntityDataSource cuando hay que realizar relaciones,

<asp:EntityDataSource ID="dsSubCategory"runat="server"   
   ConnectionString="name=AdventureWorksModelConnection"   
   DefaultContainerName="AdventureWorksModelConnection"

   EntitySetName="ProductSubcategory"   
   Select="it.[ProductSubcategoryID], it.[Name]"

Where="it.ProductCategory.ProductCategoryID = @ProductCategoryID">  
    <WhereParameters>  
        <asp:ControlParameter ControlID="ddCategories"   
        Type="Int32"

Name="ProductCategoryID" />  
    </WhereParameters>

</asp:EntityDataSource>

<asp:LinqDataSource ID="SalesOrderDetailsLinqDataSource" runat="server"

ContextTypeName="AdventureWorks.AdventureWorksLT2008"

EntityTypeName="" TableName="SalesOrderDetails"

Where="SalesOrderID == @SalesOrderID">

<WhereParameters>

<asp:ControlParameter

ControlID="SalesOrdersListView"

Name="SalesOrderID"

PropertyName="SelectedValue"

Type="Int32"

DefaultValue="0" />

</WhereParameters>

</asp:LinqDataSource>

Choosing a SQLDataSource, LINQDataSource or EntityDataSource

**SQLDataSource**

This is the oldest technology of the three and has the advantage of simplicity of use for people whom are adept at SQL. The control has separate SQL commands for inserting, deleting and updating data.

If you are not fluent in writing SQL then you will find that you need to get up to speed with SQL to make best use of the control.

**SQLDataSource -Advantages**

The SQL in each command can make full use of SQL Server functions and SQL syntax.

Everything uses standard SQL syntax.

**SQLDatasource – Disadvantages**

The control cannot utilise SQL Server TIMESTAMP data for optimistic concurrency.

Whilst the control does support concurrency conflicts by checking that old data values have not changed during a write-back this can get quite complicated and messy.

You can find issues when performing write-backs with concurrency control (compare all values) and null values. VS 2008 will automatically generate the SQL commands to avoid for these problems (we use a slightly different technique to that used by Visual Studio).

Data Binding from controls such as the GridView to field names containing spaces or other special characters become read-only.

**LINQDataSource**

This data source is a step down the road in taking away a lot of the technical detail behind making the mechanics of updating data work and freeing up the developer to focus on more application related issues.

In order to use LINQ you need to either use an external program called SQLMetal (supplied with Visual Studio or a better choice is to use the Free utility SQL Metal Builder which provides a more friendly interface for running SQLMetal which is a command line program), or a manual process to create a dbml file in your VS web site. The dbml file is a roadmap of all your tables and views in the application and contains a lot of clever technology that enables LINQ to offer certain advantages over the SQLDataSource.

**LINQDataSource – Advantages**

The control automatically supports SQL Server TIMESTAMP data types for concurrency control. Checking only timestamp’s and primary keys during write-backs.

Fieldnames with spaces or special characters get aliased automatically, which allows them to be updated.

A graphical picture in a dbml file gives an easy to use interface to the database tables and views.

Views can be manually annotated as having a unique field that acts like a primary key allowing them to be updated (by default they are not updateable).

All the optimistic write-back code is handled for you.

The visual complexity of the control is significantly reduced over the SQLDataSource, as all the clever concurrency control is no longer required and the control does not require explicit SQL commands for inserting, updating and deleting data.

Any spaces in entity names are removed; this is simply an aspect how the technology operates.

**LINQDataSource – Disadvantages**

The following applies to Binding in controls on a FormView controls, but NOT ListView, GridView or DetailsView. Updating existing records which have null values in numerical or date fields fail to update. Similarly numerical and date fields can not be easily cleared of any value as this causes failures in the Binding. LINQ is not translating empty strings into null values. The only solution we have found is to ensure that all numeric and date data have default values. For existing records default values need to be supplied if they are missing.

LINQ Where clauses do not support direct date comparisons, so a work-around is required here.

The calculation (Year + Month\*31 + Day) can be used for date comparisons, for a date range you need ((RequiredDate.Value.Year > OtherDate.Value.Year)||((RequiredDate.Value.Year == OtherDate.Value.Year) && (RequiredDate.Value.Month\*31 + RequiredDate.Value.Day) >= (OtherDate.Value.Month\*31 + OtherDate.Value.Day))).

The dbml must be manually updated, or the complete file re-generated.

Views require the manual addition annotation of the unique keys to support updateability.

An external program SQLMetal (supplied) is required to automatically generate a dbml file.

When writing where clause these are written in a C# style, and those adept at SQL but not C# have new skills to learn (this could be considered an advantage for those with C# skills).

A view with a name like “Orders Qry” will get changed to OrdersQry, whereas a field called “Qrt 1” becomes Qrt\_1. So in the case of table/view names the spaces are removed, but for field names they are replaced with an underscore. This inconsistency has been changed in the EntityDataSource where the “\_” is used in both cases.

If you are using SQL Server schemas such as for example a view called Reporting.vw\_tblParameters, then SQLMetal gives this object the same name, but interactive drag and drop in Visual Studio drops of the schema prefix in our applications, making it vw\_tblParameters. The different behaviour here is a little confusing.

The most significant problem in the above is the first point which relates to null values in date and numeric fields, this can be resolved by ensuring these fields always have values and updating existing data. This problem is no longer an issue with an EntityDataSource.

One other point to consider is the lack of support for the automatic updating of the dbml file. However, using manual techniques new tables and views can be quite easily dragged and dropped onto the dbml layout, and existing tables removed and added back to reflect structural changes.

**EntityDataSource**

The very latest choice of technology is the EntityDataSource, which extends the approach taken with the LINQDataSource by adding further capabilities.

**EntityDataSource – Advantages**

The newest type of Data Source has most of the advantages of LINQ and offers additional features.

The problems with null data values in LINQDataSources do not occur here.

The graphical schema can be automatically generated within Visual Studio, and automatically updated for underlying changes made to the database structures.

Combinations of fields for Views are automatically identified as forming a key.

As with LINQDataSources, TIMESTAMP concurrency is supported, and problem field names dealt with automatically.

Any spaces in entity names are replaced with an “\_” character, unlike LINQ where they were removed.

Unlike with LINQ the like keyword can be used in a where clause.

**EntityDataSource – Disadvantages**

To make Views updateable you need to write stored procedures for the insert, update and delete operations. (In some ways this is not necessarily a disadvantage, but it is inconvenient when compared against LINQ's ability to do this automatically once given the the keys.

Views which do not have any underlying unique keys are not able to be added to the data model, in such situations for example if used with DropDownLists consider using an SQLDataSource as an alternative. NOTE this normally affects only a few views, and if the EDM can work out the underlying keys from the base tables then this is not a problem. Using a DropDownList based on a view containing for example (SELECT DISTICT Country FROM Customers) would be a problem.

Field names in tables such as \_Country, get assigned a character prefix C, becoming C\_Country. LINQ did not do this and allowed field names to start with a \_.

Foreign keys are not exposed in a table, you need to refer to the other side of the relationship to access key values, for example in an Orders table with a SupplierID related to a Suppliers table, you would refer to the field as Suppliers.SupplierID. If you do not like this approach, then by using views, you can directly refer to the foreign keys.

**Summary**

If you don’t already have a good reason for choosing a type of datasource and you are willing to learn a little C# style syntax then our recommendation would be to use the EntityDataSource. If you are a hardened SQL fan, and want to explicitly use SQL Server specific functions outside of encapsulating them in views then the older SQLDataSource maybe a better choice.

Have you instead considered using the ObjectDataSource? It could provide the best of the two - you provide a clean, declarative binding so there's no binding code required yet the DataProvider (or Repository) class which ultimately provides the data has to be written in C#. From such class you can use any data access technology, EF, Linq, SQL, anything