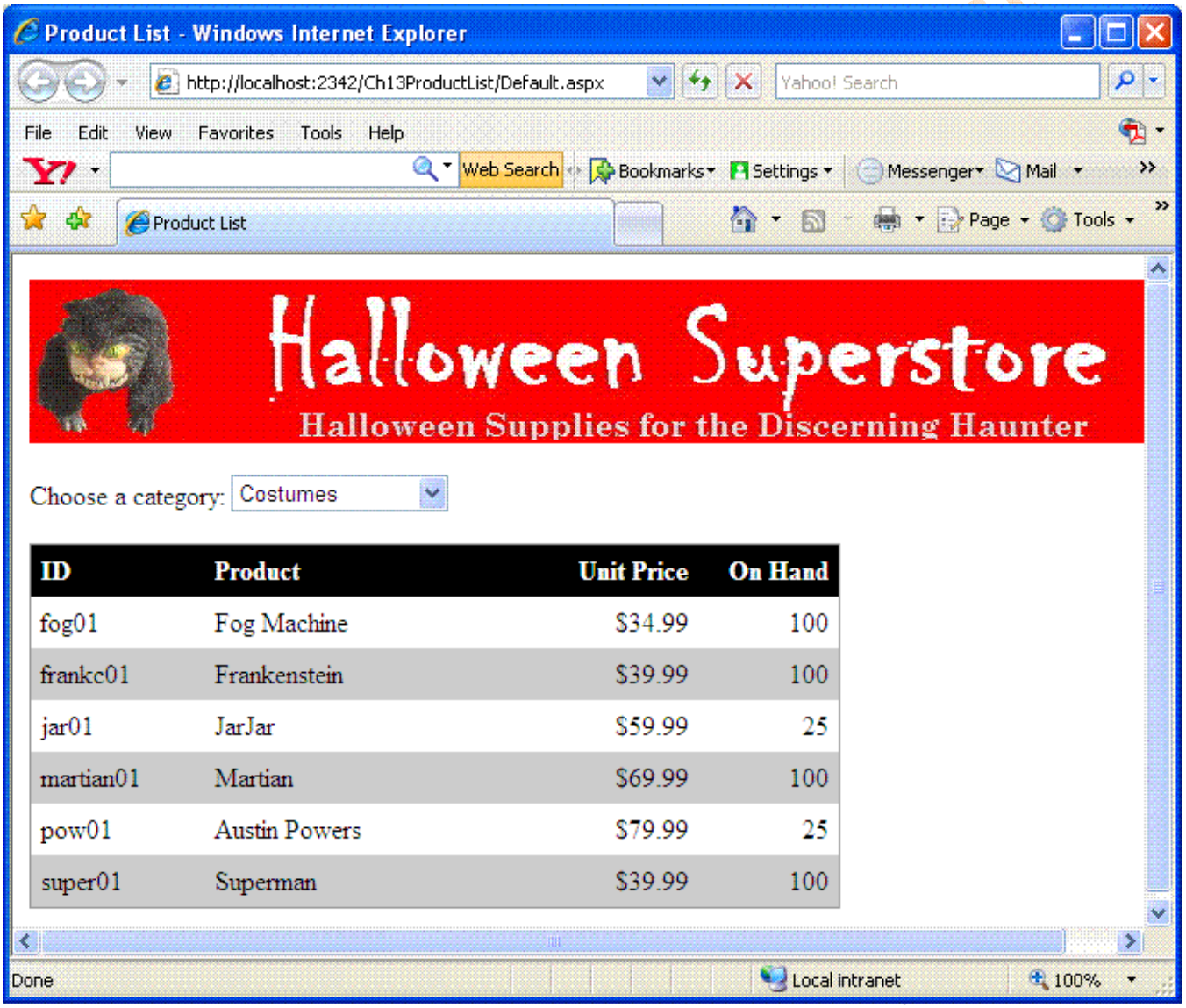
Lecture 9

SQL data sources

The data source controls are one is most important feature of ASP.NET 2.0. *SqlDataSource* control lets you access data from a SQL Server database with little or no programming.

A *Ch13ProductList* application that uses two SQL data sources

The *Ch13ProductList* application displayed in a web browser



Description

- The Product List application uses *SqlDataSource* controls to get category and product data from a SQL Server database and display it in two bound controls.

- The drop-down list near top of the form displays the product categories. This control is bound to the first data source control.
- The *DataList* control, which is bound to the second data source control, displays the data for the products that are in the category that's selected in the drop-down list.
- This application requires no C# code in the code-behind file

SqlDataSource control

Aspx code generated for a SqlDataSource control

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:HalloweenConnectionString %>"
    SelectCommand="SELECT [CategoryID], [LongName]
                   FROM [Categories]
                   ORDER BY [LongName]">
</asp:SqlDataSource>
```

Basic *SqlDataSource* control attributes

Attribute	Description
ID	The *ID* for the *SqlDataSource* control.
Runat	Must specify *"server".*
ConnectionString	The connection string. In most cases, you should use a *<%$* expression to specify the name of a connection string saved in the *web.config* file.
ProviderName	The name of the provider used to access the database. Value can be *System.Data.Odbc, System.Data.Oledb, System.Data.OracleClient,* or *System.Data.SqlClient.* The default is *System.Data.SqlClient.*
SelectCommand	The SQL Select statement executed by the data source to retrieve data.

Description

- To create a *SqlDataSource* control, drag the control from the *Data* group of the Toolbox onto the form. Then, choose *Configure Data Source* from the control's smart tag menu and proceed from there.
- Once the *SqlDataSource* control has been configured, you can bind controls to it so they get their data from the SQL data source.
- You can also create a SQL data source using the *Choose Data Source* command in the smart tag menu of a bindable control. When you use this command, the dialog box is displayed. Then, you can select the *Database* icon and proceed from there.

Defining the connection

Description of the dialog boxes for defining a connection

- The *Configure Data Source* dialog box asks you to identify the data connection for the database you want to use. If you're previously created a connection for the database, you can select it from the drop-down list. To see the connection string for that connection, click the + button below the drop-down list.
- To create a new connection, click the *New Connection* button to display the *Add Connection* dialog box. Then, enter the name of the database server in the *Server Name* text box or select it from the drop-down list. For SQL Server Express, you can use *localhost\sqlexpress* as the server name.
- After you enter the server name, select the authentication mode you want to use (we recommend *Windows Authentication*). Then, select the database you want to connect to from the *Select* or Enter a Database Name drop-down list.
- To be sure that the connection is configured properly, you can click the *Test Connection* button.

A connection string in the *web.config* file

The *ConnectionString* section of the *web.config* file

```
<connectionStrings>
  <add name="HalloweenConnectionString"
    connectionString="Data Source=VPOUGATCHEV;Initial Catalog=Halloween;Integrated
    Security=True" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Aspx code that refers to a connection string in the web.config file

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  ConnectionString="<%$ ConnectionStrings:HalloweenConnectionString %>"
  SelectCommand="SELECT [CategoryID], [LongName]
                 FROM [Categories]
                 ORDER BY [LongName]">
</asp:SqlDataSource>
```

We recommend that you always save the connection string in the *web.config* file. Then, if the location of the database changes, you can change the connection string in the *web.config* file rather than in each page that uses the connection.

Working with parameters

The aspx code for a SqlDataSource control that includes a select parameter

```
<asp:SqlDataSource ID="SqlDataSource2" runat="server"
        ConnectionString="<%$ ConnectionStrings:HalloweenConnectionString %>"
        SelectCommand="SELECT [ProductID], [Name], [UnitPrice], [OnHand]
                       FROM [Products]
                       WHERE ([CategoryID] = @CategoryID)
                       ORDER BY [ProductID]">
        <SelectParameters>
            <asp:ControlParameter  Name="CategoryID" Type="String"
                ControlID="ddlCategory" PropertyName="SelectedValue"  />
        </SelectParameters>
</asp:SqlDataSource>
```

Elements used to define select parameters

Element	Description
SelectParameters	Contains a child element for each parameter used by the data source's Select statement.
ControlParameter	Defines a parameter that gets its value from a control on the page.
QueryStringParameter	Defines a parameter that gets its value from a query string in the URL used to request the page.
FormParameter	Defines a parameter that gets its value from an HTML form field.
SessionParameter	Defines a parameter that gets its value from an item in session state.
ProfileParameter	Defines a parameter that gets its value from a property of the user's profile.
CookieParameter	Defines a parameter that gets its value from a cookie.

The *ControlParameter* element

Attribute	Description
Name	The parameter name.
Type	The SQL data type of the parameter.
ControlID	The ID of the web form control that supplies the value for the parameter.
PropertyName	The name of the property from the web form control that supplies the value for the parameter.

DataList control

A *DataList* control displays items from a repeating data source such as a data table.

A simple list displayed by a DataList control

Austin Power $79.99
Frankenstein $39.99
JarJar $59.99
Martian $69.99
Superman $39.99

The asp tag for the DataList control

```
<asp:DataList ID="dataList1" runat="server" DataSourceID="SqlDataSource2">
    <ItemTemplate>
        <asp:Label ID="lblName" runat="server"
                Text='<%# Eval("Name") %>'></asp:Label>
        <asp:Label ID="lblUnitPrice" runat="server"
                Text='<%# Eval("UnitPrice", "{0:c}") %>'></asp:Label>
    </ItemTemplate>
</asp:DataList>
```

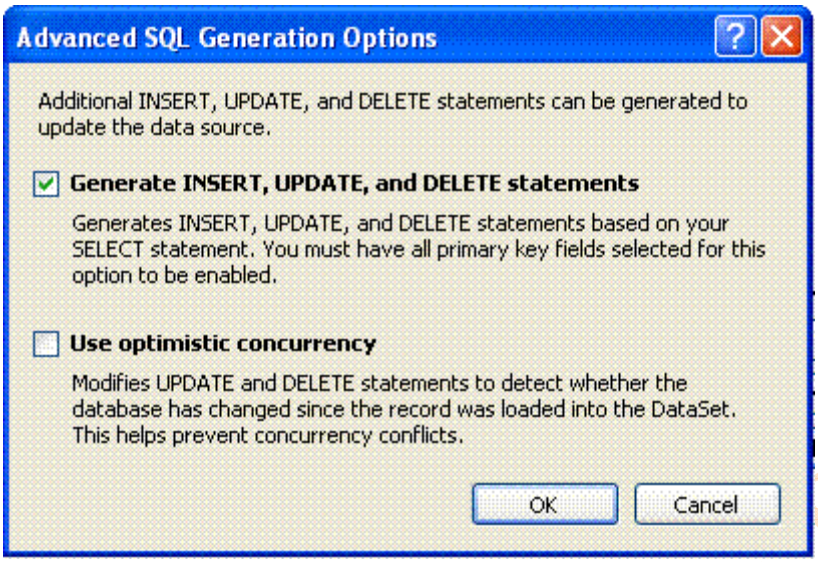
Basic attributes of the DataList control

Attribute	Description
ID	The *ID* for the *DataList*
Runat	Must specify *"server"*
DataSourceID	The *ID* of the data source to bind the data list to.

Advanced features of a SQL data source

A SQL data source can include Insert, Update, and Delete statements that let you automatically update the underlying database based on changes made by the user to bound data controls.

The Advanced SQL Generation Option dialog box



The aspx code for a SqlDataSource control that uses action queries

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:HalloweenConnectionString %>"
    SelectCommand="SELECT [Category], [ShortName], [LongName]
                   FROM {Categories]"
    InsertCommand="INSERT INTO [Categories] ([CategoryID], [ShortName], [LongName])
                   VALUES (@CategoryID, @ShortName, @LongName)"
    UpdateCommand="UPDATE [Categories]
                    SET [ShortName] = @ShortName,
                        [LongName] = @LongName
                    WHERE [CategoryID] = @CategoryID">
    DeleteCommand="DELETE FROM [Categories] WHERE [CategoryID = @CategoryID]"

    <DeleteParameter>
        <asp:Parameter Name="CategoryID" Type="String" />
    </DeleteParameter>

    <UpdateParameter>
        <asp:Parameter Name="ShortName" Type="String" />
        <asp:Parameter Name="LongName" Type="String" />
        <asp:Parameter Name="CategoryID" Type="String" />
    </UpdateParameter>

    <InsertParameter>
        <asp:Parameter Name="CategoryID" Type="String" />
```

```
        <asp:Parameter Name="ShortName" Type="String" />
        <asp:Parameter Name="LongName" Type="String" />
    </ InsertParameter >
</asp:SqlDataSource>
```

Changing the data source mode

ADO.NET provides two basic ways to retrieve data from a database:

- Retrieving the data into a dataset, this retains a copy of the data in memory so it can be accessed multiple times and updated if necessary
- Retrieving the data using a data reader, this lets you retrieve the data in forward-only, read-only fashion.

The *DataSourceMode* attribute

Attribute	Description
DataSourceMode	*DataSet* or *DataReader*. The default is *DataSet*, but you can specify *DataReader* if the data source is read-only.

A SqlDataSource control that uses a data reader

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
        ConnectionString="<%$ ConnectionStrings:HalloweenConnectionString  %>"
        DataSourceMode="DataReader"
        SelectCommand="SELECT [CategoryID], [LongName]
                        FROM [Categories] ORDER BY [LongName]"
</asp:SqlDataSource>
```

Caching

ASP.NET's caching feature lets you save the data retrieved by a data source in cache memory on the server. That way, the next time the data needs to be retrieved, the cached data is used instead of getting it from the database again. Since this reduces database access, if often improves an application's overall performance.

To enable caching, you simply set the EnableCaching attribute for a SQL data source to True. Then, you can use the CacheDuration attribute to specify how long data should be kept in the cache. If, for example, the cached data rarely changes, you can set a long cache duration value such as 30 minutes or more. Or, if the data changes frequently, you can set a short cache duration value, perhaps just a few seconds.

If the data in the database changes before the duration expires, the user will view data that is out of date. Sometimes, that's okay so you don't have to worry about it. Otherwise, you can minimize the chance of this happening by setting a shorter duration time.

SqlDataSource attribute for caching

Attribute	Description
EnableCaching	A *Boolean* value that indicates whether caching is enabled for the data source. The default
CacheDuration	The length of time in seconds that the cached data should be saved in cache storage.
CacheExpirationPolicy	If this attribute is set to *Absolute*, the cache duration timer is started the first time the data is retrieved and is not reset to *zero* until after the time has expired. If this attribute is set to *Sliding*, the cache duration timer is reset to *zero* each time the data is retrieved. The default is *Absolute*.
CacheKeyDependency	A string that provides a key value associated with the cached data. If you provide a key for the cached data, you can use the key value to programmatically expire the cached data at any time.

A SqlDataSource control that uses caching

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
        ConnectionString="<%$ ConnectionStrings:HalloweenConnectionString  %>"
        EnableCaching="True" CacheDuration="60"
        SelectCommand="SELECT [CategoryID], [LongName]
                       FROM [Categories] ORDER BY [LongName]"
</asp:SqlDataSource>
```