CHAPTER 3

Displaying Data on a Page

The previous two chapters covered the fundamentals required to start using databases. Chapter 1 showed the various types of data sources available. Chapter 2 then moved on to talk about relational databases, and by the end of the chapter, you had fundamentally the same database in three different flavors: SQL Server 2005, MySQL 5.0, and Microsoft Access.

We've covered an awful lot of theory so far. Now, you'll see how to put some of this into practice. In this chapter, you'll look at how to display read-only data on a page using the new (to ASP.NET 2.0) `SqlDataSource` and `GridView`.

Now, the phrase *read-only* may sound limiting to you, but Web sites send you a lot more read-only data than data you can alter. Consider search engine results, product details on an e-commerce Web site, news reports from the innumerable feeds on the Web, auction pages, and so on. They're all read-only, unless you have the administrative privileges to change them. Regardless, you may be itching to get straight onto editable data, and you'll look at how to handle that in Chapters 8 and 9.

For the less impetuous, you'll see that working with read-only data isn't a yoke to bear, but on the other hand, it isn't a bag of feathers. You have less to do when you don't have to accommodate creating, updating, and deleting data, but you still have plenty of ways to make mistakes.

We'll spend most of the chapter looking at how to interact with SQL Server 2005. However, there are a few wrinkles when you want to use MySQL 5.0 or Microsoft Access. At the end of the chapter, we'll look at those differences and how they affect the pages you build.

This chapter covers the following topics:

- The data source Web controls that have been added in ASP.NET 2.0, focusing on how the `SqlDataSource` simplifies the process of connecting to a data source
- How to use `SELECT` queries to return the results you require, starting with basic `SELECT` queries and then expanding to those that order results and work across multiple tables in the database
- How to combine a `SqlDataSource` with a `DropDownList` to filter the results using a `WHERE` clause and display them in a `GridView`
- How to modify the `WHERE` clause to support both filtered and nonfiltered results returned from the same query

Introducing the Data Source Web Controls

When dealing with databases prior to ASP.NET 2.0, you had to write quite a lot of code, even for the simplest of pages. You needed to create a connection to the database using a Connection object, create a query to execute against the connection using a Command object, and then decide whether you wanted to retrieve the data using a DataAdapter object (into a `DataSet` object) or via a DataReader object.

Compare this to how you retrieved the data in the very brief example in Chapter 1 using a `SqlDataSource` object. You didn't write any code, but instead used a wizard to create the correct object. The `SqlDataSource` is perfect when data-binding the results to a `GridView` or similar Web control.

This drag-and-drop approach is a far simpler way of retrieving data than writing code to access the database. However, it isn't suitable for every situation, and you probably won't be able to build an entire Web site that doesn't contain code to access the database. We'll look at writing code to access the database in Chapter 4.

ASP.NET 2.0 introduces five Web controls that retrieve data from a particular data source. These Web controls can be broadly categorized into two categories, depending on the type of data that they're retrieving:

- **Set-based:** *Set-based data* is the type of data that you've seen in the two previous chapters—data that is organized into columns and rows. ASP.NET 2.0 provides three set-based data source Web controls:
  - `AccessDataSource`: This control works exclusively with Access databases. It's limited to retrieving data from the database and won't allow data to be written back to the database. If you need to do more than retrieve data from the database, you need to use the `SqlDataSource`.
  - `ObjectDataSource`: This control provides an abstraction layer between an object and a data-bound Web control such as the `GridView`. Commonly used with a multitiered architecture, the control enables you to use data-bound Web controls while still retaining that architecture.
  - `SqlDataSource`: This control acts as the bridge between the data-bound Web controls and a database. Although the name `SqlDataSource` sounds as though it will work with only SQL Server databases, it actually works with any data provider (and therefore with any ODBC- or OLE DB-compliant data source).
- **Hierarchical:** Hierarchical data sources are used to retrieve data that is, well, hierarchical—data that isn't in column-and-row form. ASP.NET 2.0 provides two Web controls that work with hierarchical data:
  - `SiteMapDataSource`: ASP.NET 2.0 introduces a complete new set of features that make building navigational functions into a Web site extraordinarily easy. For more information about site maps, see `http://msdn.microsoft.com/en-us/library/k36h0dfh.aspx`.
  - `XmlDataSource`: This control is designed to work with XML data. For more information about the `XmlDataSource`, see `http://msdn.microsoft.com/en-us/library/51ew3eby.aspx`.

As you can see, ASP.NET 2.0 provides five different data source Web controls that you can use in different circumstances, depending on the data that you need to view. By far, the one that you'll use most often is the `SqlDataSource`, as this allows you to connect to any data source that can be accessed using a data provider, including databases with an ODBC driver or OLE DB provider. As you'll see when you look at using the `GridView` to modify data in Chapter 9, the `SqlDataSource` can also modify the data within the database.

Introducing SELECT Queries

To return data from a database, you use a `SELECT` query. The basic format of the `SELECT` query is as follows:

```
SELECT <select column list>
FROM <table list>
[ WHERE <constraints> ]
[ ORDER BY <order column list> ]
```

A `SELECT` query has four parts; two of them are required, and two are optional. A `SELECT` query must always have a *`select column list`* and a *`table list`*, but the `WHERE` and `ORDER BY` clauses are optional (indicated by the square brackets in the definition). These optional clauses allow you to filter and sort the data you're retrieving.

■**Note** I'm not trying to teach you every little nuance of SQL here; I'm showing only a small subset of what's possible. Appendix C contains more details of the various SQL commands. For a complete reference, refer to *The Programmer's Guide to SQL* by Christian Darie and Karli Watson (1-59059-218-2; Apress, 2003).

The `SELECT` query can get confusing quickly, so we'll start with a simple example of querying a single table. Then we'll expand on the example, demonstrating how to sort the results, select data from more than one table, and filter the results returned from the database.

■**Note** In the examples in this chapter, you'll use the SQL Server 2005 database. Using MySQL 5.0 or Microsoft Access is very similar. For details about the differences, see the "Connecting to MySQL 5.0 and Microsoft Access" section later in this chapter.

Try It Out: Querying a Single Table

The easiest form of `SELECT` query you can make against the database is to query for values from a single table. In this case, you'll list all the Players in the database. This example is the basis for the rest of the examples in this chapter.

Follow these steps:

1. Start Visual Web Developer and create a new Web site called `Chapter03` in the `C:\BAND\` folder.
2. Delete the `Default.aspx` file and create a new Web Form by selecting Add New Item from the Web site's context menu in the Solution Explorer.
3. Enter `Select.aspx` as the name for the Web Form. Make sure that the language is Visual C# and that the Place Code in a Separate File option is unchecked. Click Add to create the Web Form.
4. Right-click `Select.aspx` in the Solution Explorer and select Set As Start Page.
5. In the Source view, change the name of the page to `SELECT` by changing the `<title>` tag as follows:

   ```
   <title>SELECT</title>
   ```
6. Switch to the Design view by clicking the Design tab at the bottom of the window.
7. Expand the Data entry in the Toolbox on the left side of the screen (if the Toolbox is not visible, select View ➤ Toolbox) and add a `SqlDataSource` onto the page.
8. From the `SqlDataSource` Tasks menu, shown in Figure 3-1, select the Configure Data Source option. (If the Tasks menu isn't shown, click the right-facing arrow at the top right of the `SqlDataSource` to open the menu.)

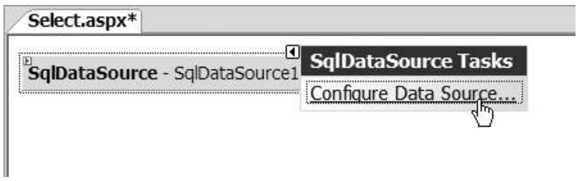
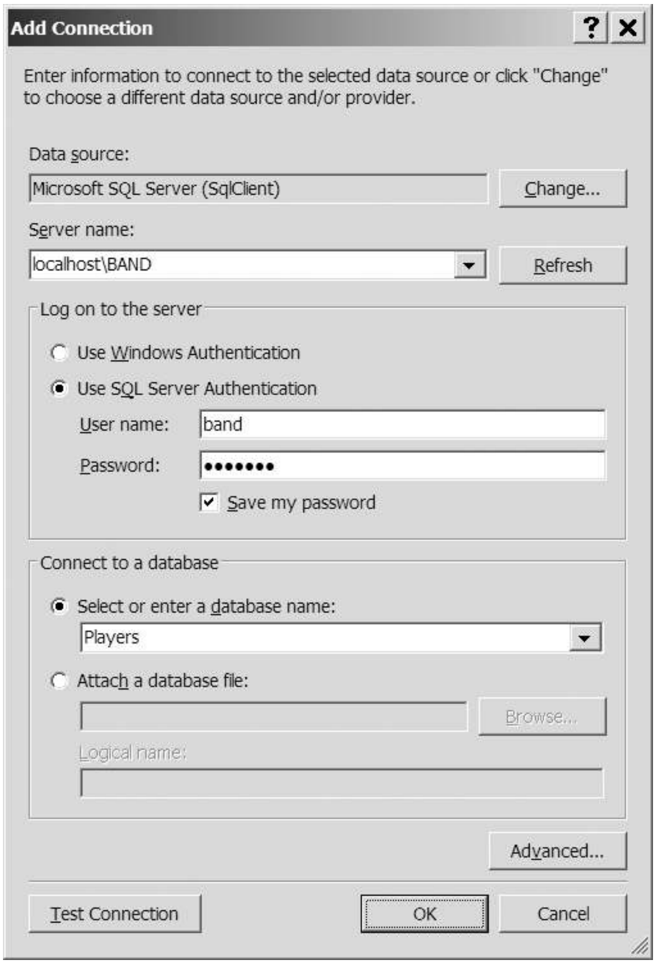


Figure 3-1. *The SqlDataSource Tasks menu*

9. You'll see that the connections to the databases you created in Chapter 2 are available in the Data Connections drop-down list. However, you want to use the reduced privilege account, rather than the administrator account, so you need to create a new connection. Click the New Connection button.
10. In the Add Connection dialog box, enter the server name as `localhost\BAND` and select SQL Server Authentication as the authentication method. Enter a username of **band** and a password of **letmein**. Check the Save My Password option. Also select the Players database from the Select or Enter a Database Name drop-down list. Your dialog box should look like Figure 3-2. Click OK to create the new connection.



Data source: Microsoft SQL Server (SqlClient)

Server name: localhost\BAND

( ) Use Windows Authentication

(●) Use SQL Server Authentication

User name: band

Password: •••••••

[✓] Save my password

(●) Select or enter a database name: Players

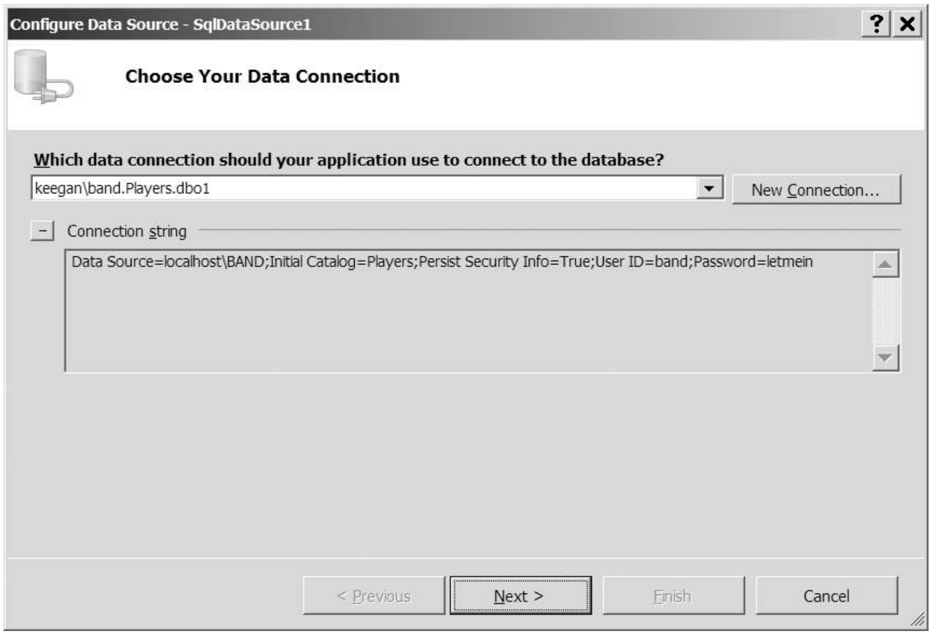
( ) Attach a database file:

Figure 3-2. *Creating a new connection in Visual Web Developer*

11. In the Configure Data Source wizard, the drop-down list will have been prepopulated with a new entry for the reduced privilege account. The entry will have a name that ends with `band.BAND.dbo1`; in this case, `keegan\band.BAND.dbo1`. As shown in Figure 3-3, you can expand the Connection String entry to show the actual connection string being used. Click Next to continue.

12. On the Save the Connection String step, make sure the Yes, Save This Connection option is selected, and enter a name for the connection of `SqlConnnectionString`. Then click Next.

13. The next step of the wizard allows you to create the `SELECT` query that the `SqlDataSource` will use. Select the Player table in the Name drop-down list, and check the PlayerName and PlayerManufacturerID columns, as shown in Figure 3-4. Click Next.


Configure Data Source - SqlDataSource1
Choose Your Data Connection
< Previous
Next >
Finish
Cancel

Which data connection should your application use to connect to the database?

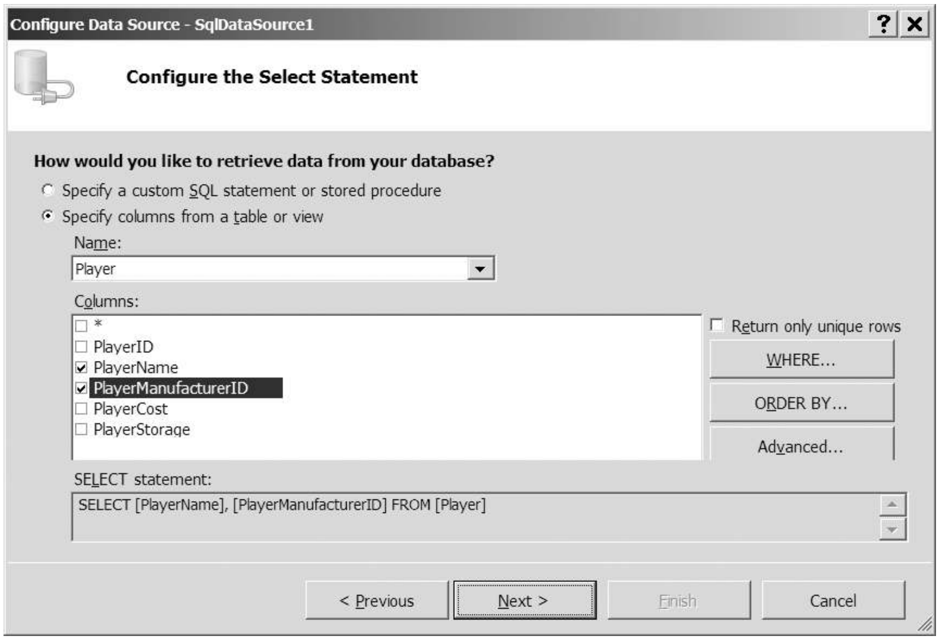
keegan\band.Players.dbo1

New Connection...

Connection string

```
Data Source=localhost\BAND;Initial Catalog=Players;Persist Security Info=True;User ID=band;Password=letmein
```

Figure 3-3. *Selecting the correct connection and verifying the connection string*


Configure Data Source - SqlDataSource1
Configure the Select Statement
WHERE...
ORDER BY...
Advanced...
< Previous
Next >
Finish
Cancel

How would you like to retrieve data from your database?

Specify a custom SQL statement or stored procedure
[✓ Specify columns from a table or view]

Name: Player

Columns:

- ☐ *
- ☐ PlayerID
- ☑ PlayerName
- ☑ PlayerManufacturerID
- ☐ PlayerCost
- ☐ PlayerStorage

☐ Return only unique rows

SELECT statement:

```
SELECT [PlayerName], [PlayerManufacturerID] FROM [Player]
```

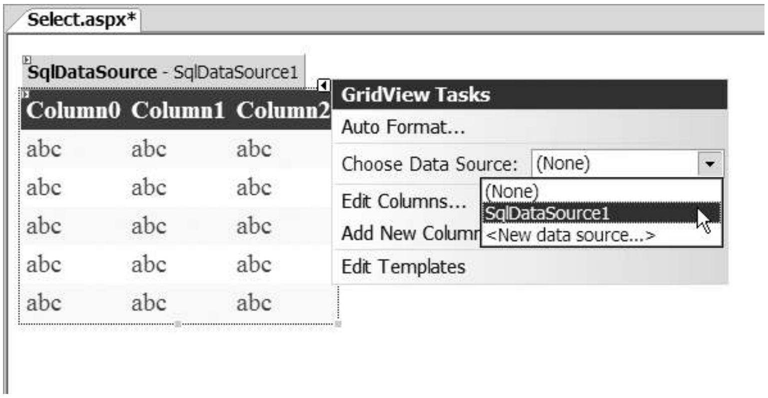
Figure 3-4. *Creating the SELECT query*

14. On the Test Query step, you can click the Test Query button to test that the query is correct. Once you're happy that it is (if it isn't, click Previous to go back to the previous step and modify the query), click Finish to close the wizard.

15. Switch back to the Design view of the page. Add a `GridView` from the Data section of the Toolbox to the page, below the `SqlDataSource`.

16. From the `GridView` Tasks menu, select Auto Format, and then select Colorful from the list. (If you don't like Colorful, you can pick another format, although the pages you build won't look exactly like the illustrations in this chapter.) Click OK.

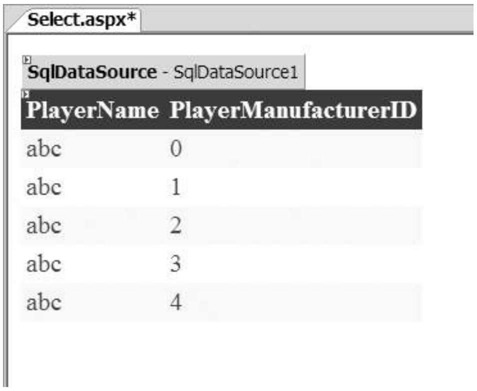
17. From the Choose Data Source drop-down list, select `SqlDataSource1`, as shown in Figure 3-5.



Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

Figure 3-5. *Selecting the correct DataSource from the GridView Tasks menu*

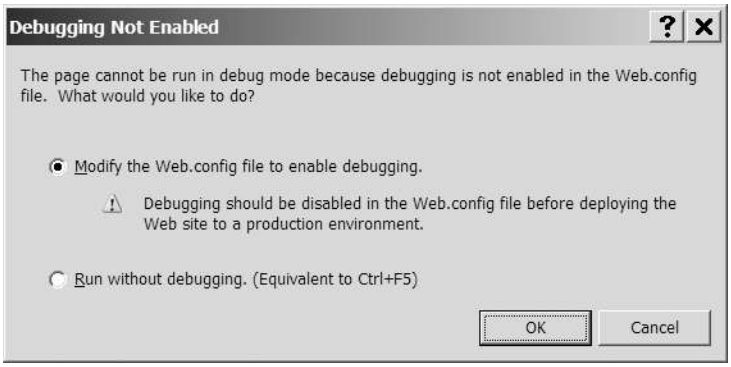
18. As shown in Figure 3-6, the `GridView` will change to show the columns from the `SELECT` query that you defined for the `SqlDataSource` earlier.



PlayerName	PlayerManufacturerID
abc	0
abc	1
abc	2
abc	3
abc	4

Figure 3-6. *At design time, the GridView shows correct column information.*

19. Run the page in debug mode by clicking the Start Debugging button in the toolbar, selecting Debug ➤ Start Debugging, or pressing F5. All three methods will start the debugging process. Debugging isn't enabled by default for Web sites, so the Debugging Not Enabled dialog box will appear, as shown in Figure 3-7. Accept the Modify option by clicking OK.



Debugging Not Enabled

The page cannot be run in debug mode because debugging is not enabled in the Web.config file. What would you like to do?

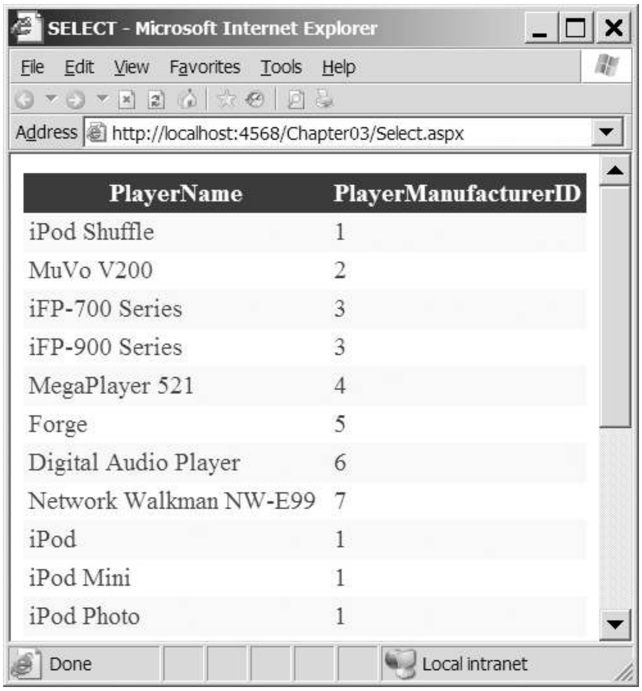
[✓ Modify the Web.config file to enable debugging.]

Debugging should be disabled in the Web.config file before deploying the Web site to a production environment.

Run without debugging. (Equivalent to Ctrl+F5)

Figure 3-7. *Enabling debugging for the Web site*

20. Visual Web Developer's built-in Web server will start (which you'll see in the system tray), and the page will be displayed in your browser, as shown in Figure 3-8.



SELECT - Microsoft Internet Explorer

Address http://localhost:4568/Chapter03/Select.aspx

PlayerName	PlayerManufacturerID
iPod Shuffle	1
MuVo V200	2
iFP-700 Series	3
iFP-900 Series	3
MegaPlayer 521	4
Forge	5
Digital Audio Player	6
Network Walkman NW-E99	7
iPod	1
iPod Mini	1
iPod Photo	1

Figure 3-8. *Results of running a SELECT query against the Player table*

How It Works

In this example, you built a relatively simple page, showing the name of all of the Players in the database alongside details of their Manufacturer. (Granted, the PlayerManufacturerID data isn't particularly helpful, and you'll see how to improve this in the "Try It Out: Querying Multiple Tables" section later in this chapter.)

As you saw in step 19, Visual Web Developer needs to modify the `Web.config` file to enable debugging. By clicking OK in the Debugging Not Enabled dialog box (Figure 3-7), you've told Visual Web Developer to add the code shown in bold to the `Web.config` file:

```
<configuration>
  <system.web>
    <compilation debug="true"/>
  </system.web>
</configuration>
```

This tells ASP.NET that the Web site is running in debug mode and to allow a debugger to attach to the Web site.

Visual Web Developer has also made a further change to `Web.config`. In step 12, you told it to save the connection string, and it has done just that, placing the connection string in the `<connectionStrings>` element. In previous versions of ASP.NET, all user settings—database connection strings, file paths, and so on—were usually stored in `Web.config` in the `<appSettings>` element. ASP.NET 2.0 introduces the new `<connectionStrings>` element, which is used exclusively to store database connection strings:

```
<configuration>
  <connectionStrings>
    <add name="SqlConnectionString"
      connectionString="Data Source=localhost\BAND;Initial Catalog=Players;
        Persist Security Info=True;User ID=band;Password=letmein"
      providerName="System.Data.SqlClient"/>
  </connectionStrings>
</configuration>
```

Entries are added to `<connectionStrings>` in much the same way as they were added to `<appSettings>` by using an `<add>` element. But unlike `<appSettings>`, where you could supply only a key and a value, you add connection-specific information, as follows:

- `connectionString`: The string that is used to make the connection to the database. The actual value depends on the provider that you're going to use (as determined by the `providerName` attribute).
- `name`: Define is the name that is used to refer to this particular connection string.
- `providerName`: Tells ASP.NET which data provider this connection uses. It can be one of the following values: `System.Data.Odbc`, `System.Data.OleDb`, `System.Data.OracleClient`, or `System.Data.SqlClient`.

In this example, you specified `SqlConnectionString` as the name of the connection string, and this is used as the value of the `name` attribute. The remaining two attributes were completed by Visual Web Developer using the information that you supplied.

You're connecting to a SQL Server database, so `System.Data.SqlClient` is used, and the `connectionString` attribute, which is the connection string itself, is built from the details that you entered in the Add Connection dialog box, as follows:

```
connectionString="Data Source=localhost\BAND;Initial Catalog=Players;
  Persist Security Info=True;User ID=band;Password=letmein"
```

The properties are as follows:

- `Data Source`: This is the server to which you're connecting. This can either be a server name or an IP address, as well as the specific instance that you require. For example, `localhost\BAND` is the `BAND` instance of SQL Server on the local machine.
- `Initial Catalog`: The database on the server to which you want to connect; `Players` in this example.
- `Persist Security Info`: When set to `True`, this property allows the username and password that are used to connect to the database to be viewed when programmatically looking at the connection string. Ideally, it should always be set to its default value of `False` but it is set to `True` by Visual Web Developer when you're using an existing connection that has its password stored. You can remove this property from the `connectionString` without causing any problems.
- `User ID`: The username to use when connecting to the database; `band` in this example.
- `Password`: The password to use when connecting to the database; `letmein` in this example.

■**Note** You have a multitude of options for connecting to data sources. For a comprehensive list, see `http://www.connectionstrings.com`.

Now that we've covered the various "background" parts of the sample page, it's time to look at the two Web controls that you used to do all the work: the `SqlDataSource` for connecting to the database and retrieving the results and the `GridView` for displaying those results.

The SqlDataSource Web Control

The `SqlDataSource` is the brains of the operation. It handles connecting to the database and executing the query against the database to return the results you requested. Here is the markup produced for it:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  ConnectionString="<%$ ConnectionStrings:SqlConnectionString %>"
  SelectCommand="SELECT [PlayerName], [PlayerManufacturerID] FROM [Player]">
</asp:SqlDataSource>
```

In its simplest form, there isn't a lot to it. As well as the obligatory `ID` and `runat` properties, it has only two other properties: `ConnectionString` and `SelectCommand`.

The `ConnectionString` property is used to specify the data source that you want to use. In this instance, the value indicates that you want to use a connection string, `SqlConnectionString`, which is stored in the `Web.config` file:

```
<%$ ConnectionStrings:SqlConnectionString %>
```

As you saw earlier, Visual Web Developer created an entry in the `<connectionStrings>` section of `Web.config` that provided the full details of the connection to the database. By specifying the `ConnectionString` in this way, you're telling the `SqlDataSource` that you want to use it.

STORING CONNECTION STRINGS

You've stored all of your connection strings within `Web.config`, as this allows the most flexibility when building Web sites. You have one connection string that you need to change should the database details change.

You should always store your connection strings in `Web.config`, as this makes for far easier maintenance of the Web site. If the database details change, you can easily modify the connection string in `Web.config`. However, you can also store the connection string directly within the page, as follows:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  ConnectionString="Data Source=localhost\BAND;Initial Catalog=BAND;
    Persist Security Info=True;User ID=band;Password=letmein"
  ProviderName="System.Data.SqlClient"
  SelectCommand="SELECT [PlayerName], [PlayerManufacturerID] FROM [Player]">
</asp:SqlDataSource>
```

You need to provide enough information to fully configure the connection, so you must also specify a `ProviderName` to tell the `SqlDataSource` which provider to use.

The property of most interest here is `SelectCommand`. This contains the `SELECT` query that will be run against the database when data is requested.

The `SELECT` query you use to select all of the Players is about as simple as it can get. You specify the columns you want to retrieve and the table you want to retrieve them from, like so:

```
SELECT [PlayerName], [PlayerManufacturerID]
FROM [Player]
```

Okay, it could have been simpler—you could have returned only one column, but returning two is just a little more complex. The columns to retrieve are specified as a comma-separated list (in this case, PlayerName and PlayerManufacturerID) after the `SELECT` query, and the table from which you want the data to be returned is specified after the `FROM` statement.

■**Note** Instead of listing the columns you want to retrieve with the query, you can use `SELECT *` to return all the columns from the query. However, it's better to explicitly list the columns to return, rather than relying on returning all the columns. Not only is it quicker (the database doesn't have to retrieve a list of columns to return before it executes the query) and produces less network traffic (you're not returning columns you don't need), but it also makes the code more readable, because anyone can see what the query is returning without having to look at the database structure.

Notice that the column names have square brackets around them, so the PlayerName column is retrieved as `[PlayerName]`. Although not required in this instance, the square brackets are used by SQL Server to delimit the name of the column. Column names can contain spaces, so the square brackets set off the entire name. For example, a column called Player Name not surrounded by brackets would cause an error, as the database would look for a column called Player. You should avoid using spaces in names within the database—whether for tables, columns, or any other database object—at all costs. It's quite easy to forget that you need to have square brackets around the names, which can lead to all kinds of problems with the queries that you're trying to execute. As you've been good and designed your database without any table or column names without any spaces, we will not need to use square brackets at all. From now on, we will not show any SQL queries with square brackets. Just be aware that Visual Web Developer may add them to the queries that it creates.

■**Note** Although you can refer to the columns using only the name of the column, such as `PlayerID`, the correct name of the column—its *full name syntax*—is `Player.PlayerID`. The database allows you to use this shorthand version because there's no confusion as to which table the column belongs. There's no restriction on the names of columns, and when you join tables, it's likely that columns with the same name will appear in multiple tables. The only way you can distinguish which column you're after is to use the full name syntax for specifying the column name.

The GridView Web Control

The `GridView` is a new control introduced in ASP.NET 2.0 that replaces the `DataGrid` from previous versions of ASP.NET. As you've seen, it's used to display data from a data source in a tabular format. But it can do a whole lot more than that, including automatic sorting and paging of results. We'll take a much more detailed look at this control starting in Chapter 7. For now, we'll focus on the way that you're using the `GridView` in this example.

Looking at the markup generated for the `GridView`, you can start to appreciate how it works (this snippet has formatting instructions removed for simplicity):

```
<asp:GridView ID="GridView1" runat="server"
  AutoGenerateColumns="False" DataSourceID="SqlDataSource1">
  <Columns>
    <asp:BoundField DataField="PlayerName"
      HeaderText="PlayerName"
      SortExpression="PlayerName" />
    <asp:BoundField DataField="PlayerManufacturerID"
      HeaderText="PlayerManufacturerID"
      SortExpression="PlayerManufacturerID" />
  </Columns>
</asp:GridView>
```

The `DataSourceID` property of the `GridView` tag specifies the name of the data source being used; in this case, you're using `SqlDataSource1`, which is the `SqlDataSource` described in the previous section.

If the `AutoGenerateColumns` property is set to `True`, then the `DataSourceID` is the only property that you need to set on a `GridView` for it to display the data from the data source Web control. The columns that it displays will be generated automatically from the results returned from the data source, and every column in the results will be displayed. However, you may not want this automatic generation—you may want to display just a subset of that data. In this case, you would set the `AutoGenerateColumns` property to `False`, and then manually define the columns you want to display using the `<Columns>` collection. In the example, Visual Web Developer has created a `Columns` collection for you, because you selected the PlayerName and PlayerManufacturerID columns (in step 13).

The `<Columns>` collection contains the columns, or *fields* in `GridView` terminology, that will be displayed. Although you defined only one column type in this example, `BoundField`, a total of seven types are available. Table 3-1 gives you an idea of what is possible using the `GridView`. We'll come back to this list starting in Chapter 7. For now, we're interested only in the `BoundField`.

Table 3-1. *GridView Column Types*

Name	Description
`BoundField`	Displays data directly from the data source. The `DataField` property indicates which particular column of the results you want to show, and you can use the optional `DataFormatString` property to format the results as required.
`ButtonField`	Shows a button that causes a postback to the server, allowing an action to be performed on the selected row.
`CheckBoxField`	Shows a check box. This type of column is usually used to display columns from the database with Boolean values.
`CommandField`	Creates a column in the results that contains Edit, Update, and Cancel buttons (as appropriate) to allow editing of the data within the selected row.
`HyperLinkField`	Shows a hyperlink.
`ImageField`	Shows an image.
`TemplateField`	Allows full control over what is displayed for the column: the header, the item displayed, and the footer. The `TemplateField` contains various templates controlling what is displayed and, within these templates, you can place whatever content you desire. This makes the `TemplateField` the most customizable of the column types.

A `BoundField` has only one property that is always required when displaying data: `DataField`. This property indicates which column in the data source you want to display. In this example, you have two columns in the `GridView`, and the `DataField` properties are set to `PlayerName` and

PlayerManufacturerID. These are the names of the columns that you want to show from the data source, and they correspond to the columns that you're returning from the SELECT query.

Visual Web Developer also adds two further properties for you: HeaderText is the text that is displayed in the header row of the table, and SortExpression is the value used to control the sorting applied to this column if sorting is enabled. The default for both of these is the name of the column that is being displayed, so you'll see that DataField, HeaderText, and SortExpression all have the same value. For this example, what is displayed in the header for the column is not important, and you're free to change it to whatever you like. Similarly, you're not concerned with sorting, so the SortExpression is not relevant at the moment; you're free to remove it. We'll come back to sorting in Chapter 7.

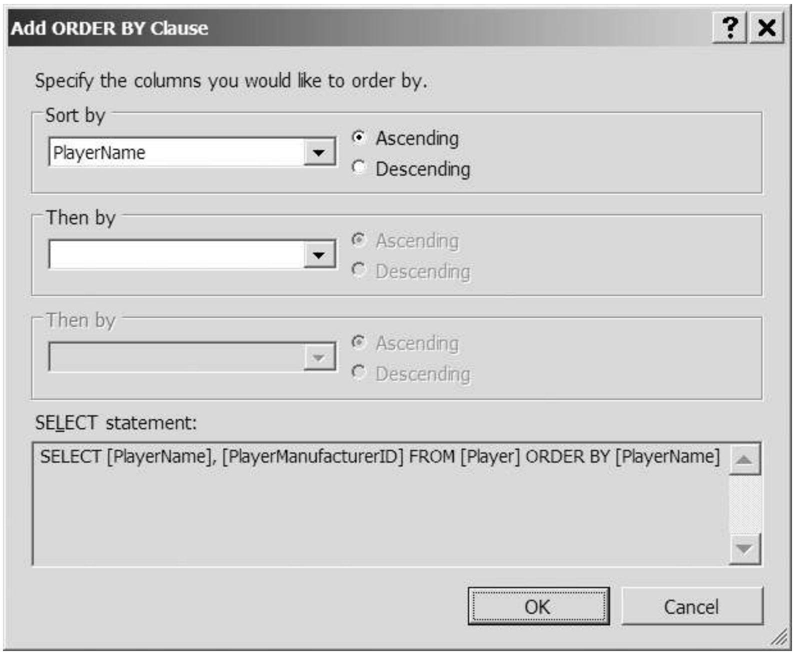
■**Note** You'll notice that the GridView on the page that you built has a lot more properties than we've looked at here. You'll see six *xxx*Style elements defined, as well as some style properties applied directly to the GridView. Feel free to experiment with the elements and properties, as they won't affect how the GridView works. You can change them directly from the Source view of the page or from the Styles section of the Properties window for the Web control. Just remember that if you get to a point where you can't see anything (blue text on a blue background, anyone?), you can always select Auto Format from the GridView Tasks menu to remove any style information that you've applied or to change to one of the predefined styles.

Try It Out: Ordering the Results

One of the problems with the results you've received in the previous example is that they're not ordered. You've returned all of the Players in the database in whatever order they were entered. If you look back at Figure 3-8, you'll see that trying to find a specific Player will require a lot of work on the behalf of the user.

What you need to do is order the results, and SQL provides this ability with the ORDER BY clause of the SELECT query. In this example, you'll build on the simple SELECT query of the last example and sort the Players alphabetically. Follow these steps:

1. If you've closed Select.aspx from the previous example, reopen it.
2. Switch to the Design view of the page. Open the Tasks menu for SqlDataSource1 and click the Configure Data Source option.
3. You've already configured the data source in the previous example, so click Next to skip past the Choose Your Data Connection step. On the Configure the Select Statement step, click the ORDER BY button.
4. In the first drop-down list of the Add ORDER BY Clause dialog box, select the PlayerName column, as shown in Figure 3-9. Then click OK.
5. Click Next, and then click Finish to close the wizard.
6. Open the page. The results will be the same as the previous example, except that the Players will be in alphabetical order, as shown in Figure 3-10.

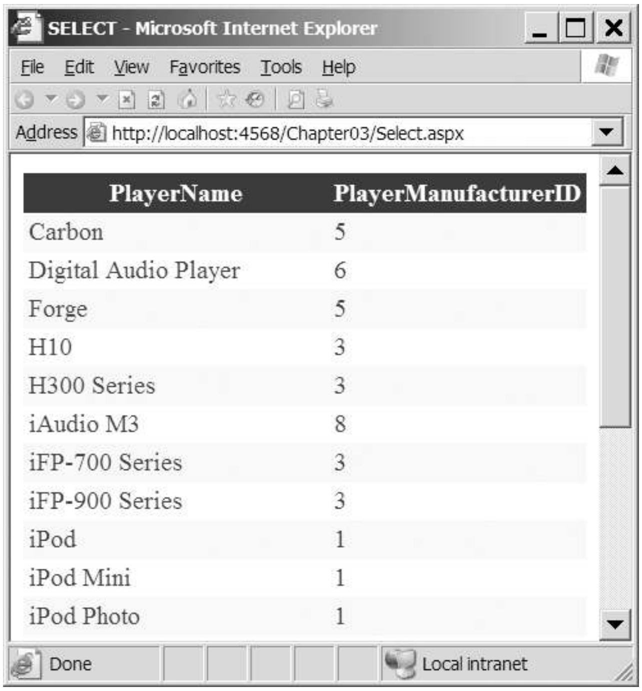


Sort by: PlayerName [✓ Ascending] Descending

SELECT statement:

```
SELECT [PlayerName], [PlayerManufacturerID] FROM [Player] ORDER BY [PlayerName]
```

Figure 3-9. *Specifying ORDER BY clauses for a SELECT query*


SELECT - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:4568/Chapter03/Select.aspx
Done
Local intranet

PlayerName	PlayerManufacturerID
Carbon	5
Digital Audio Player	6
Forge	5
H10	3
H300 Series	3
iAudio M3	8
iFP-700 Series	3
iFP-900 Series	3
iPod	1
iPod Mini	1
iPod Photo	1

Figure 3-10. *Ordering the results that are returned*

How It Works

You used the Configure Data Source wizard to modify the query that you're executing against that database. You appended an ORDER BY clause to return the results in alphabetical ascending order. If you look at the Source view of the page, you'll see that the SelectCommand property of SqlDataSource1 has been changed like so:

```
SELECT PlayerName, PlayerManufacturerID
FROM Player
ORDER BY PlayerName
```

■ **Tip** Although you used the wizard to add the ORDER BY clause, you could have quite easily modified the SelectCommand property manually and added the clause.

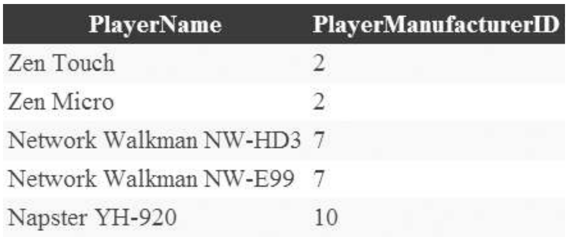
Adding the ORDER BY clause and specifying a text column sorts the results alphabetically on that column. If you specify a column for the ORDER BY clause, the sort order is, by default, ascending. The query you've used to sort the results is equivalent to specifying a sort order of ASC along with the column, like so:

```
SELECT PlayerName, PlayerManufacturerID
FROM Player
ORDER BY PlayerName ASC
```

You can also sort in descending order using DESC, like so:

```
SELECT PlayerName, PlayerManufacturerID
FROM Player
ORDER BY PlayerName DESC
```

Executing this query has the effect of sorting the results by Player name in descending alphabetical order, as shown in Figure 3-11.



PlayerName	PlayerManufacturerID
Zen Touch	2
Zen Micro	2
Network Walkman NW-HD3	7
Network Walkman NW-E99	7
Napster YH-920	10

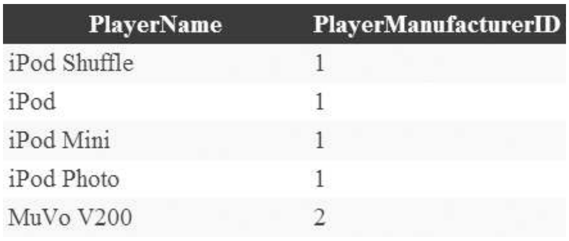
Figure 3-11. *Sorting by PlayerName in descending alphabetical order*

Although the examples you've seen sort on a text column, PlayerName, you can sort on any type of column. Dates will be sorted earliest to latest if you specify an ascending order, and latest to earliest if you specify descending. Numbers are sorted smallest to largest or largest to smallest. Each type of column follows its own rules, and the ordering they impose generally makes sense.

You can see how ordering applies to nontext columns if you order the results by PlayerManufacturerID, like so:

```
SELECT PlayerName, PlayerManufacturerID
FROM Player
ORDER BY PlayerManufacturerID
```

Executing this query returns the results in ascending PlayerManufacturerID order, as shown in Figure 3-12.



PlayerName	PlayerManufacturerID
iPod Shuffle	1
iPod	1
iPod Mini	1
iPod Photo	1
MuVo V200	2

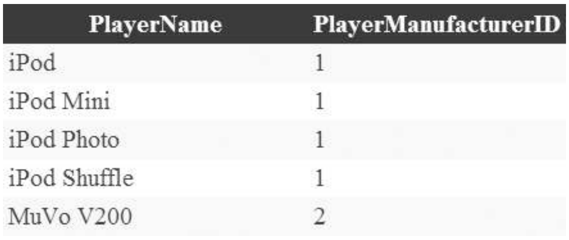
Figure 3-12. *Sorting by an integer column, PlayerManufacturerID*

You could also return the results in descending PlayerManufacturerID order, and the entries that have a PlayerManufacturerID of 10 would be at the beginning of the results.

The `ORDER BY` clause also allows you to sort the results on multiple columns, as you saw offered in the Add ORDER BY Clause dialog box (Figure 3-9). If you look at the results in Figure 3-12, you'll see that, although you've sorted by PlayerManufacturerID, the PlayerName column is unsorted. What you really want is the results sorted by PlayerManufacturerID, and then by PlayerName. You can accomplish this relatively easily by specifying the two columns you want to order, separated by commas after the `ORDER BY` clause, like so:

```
SELECT PlayerName, PlayerManufacturerID
FROM Player
ORDER BY PlayerManufacturerID, PlayerName
```

This will order the Players by PlayerManufacturerID and PlayerName. Both of the sorts will be in ascending order, as you can see from the results returned in Figure 3-13.



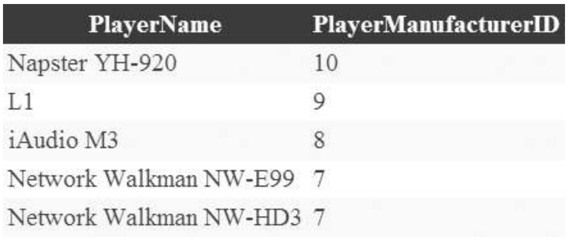
PlayerName	PlayerManufacturerID
iPod	1
iPod Mini	1
iPod Photo	1
iPod Shuffle	1
MuVo V200	2

Figure 3-13. *Sorting on multiple columns*

It's also possible to have different sort orders on different columns by specifying the sort order you want for each column, like so:

```
SELECT PlayerName, PlayerManufacturerID
FROM Player
ORDER BY PlayerManufacturerID DESC, PlayerName
```

You want the results by PlayerManufacturerID in descending order and then by PlayerName in ascending order. As shown in Figure 3-14, this is exactly the order in which the results are returned.



PlayerName	PlayerManufacturerID
Napster YH-920	10
L1	9
iAudio M3	8
Network Walkman NW-E99	7
Network Walkman NW-HD3	7

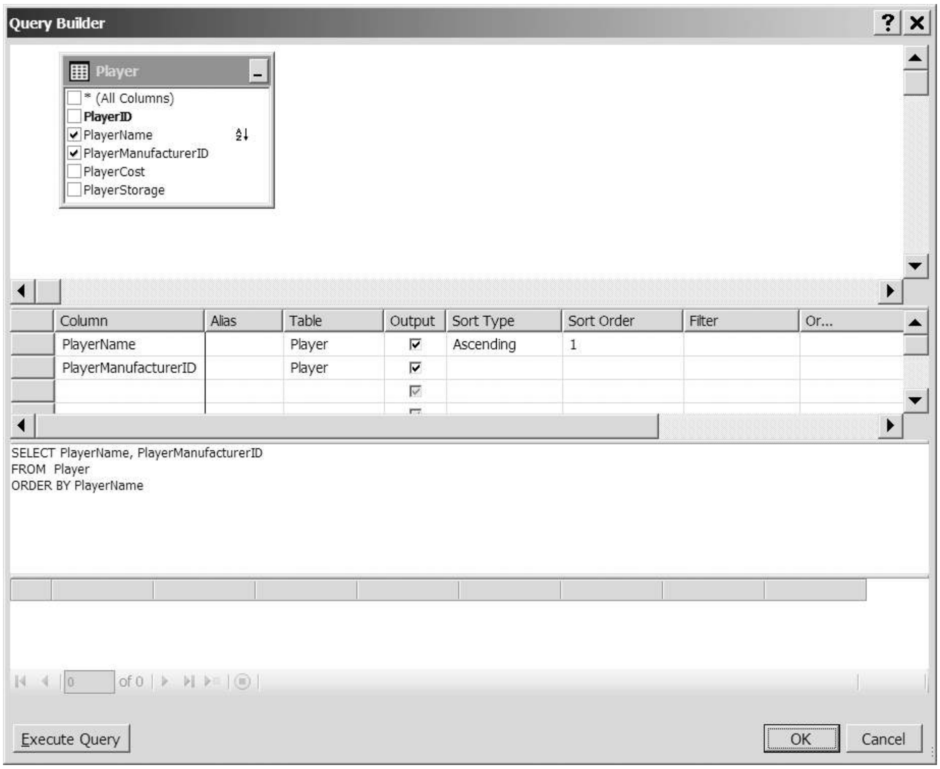
Figure 3-14. *Sorting on multiple columns with different sort orders*

Try It Out: Querying Multiple Tables

Looking at the results you received from the previous example (Figure 3-10), you'll see that although you return the title of the Player correctly, the Manufacturer is simply an integer. Recall from Chapter 2 that the PlayerManufacturerID column in the Player table corresponds to the ManufacturerID column in the Manufacturer table. Unless you're capable of remembering vast quantities of data, the integer on its own won't be a lot of use. You need some way of joining the Player and Manufacturer tables so that, instead of returning the PlayerManufacturerID, you return the name of the Manufacturer.

SQL allows you to return data from multiple tables by using the `JOIN` clause when you specify from which tables you want to return data. You'll now see a `JOIN` in action by building on the previous example and returning the name of the Manufacturer instead of the ID integer. Follow these steps:

1. If you've closed `Select.aspx` from the previous example, reopen it.
2. Select `SqlDataSource1`, and from the Properties window, click the ellipsis for the `SelectQuery` property.
3. In the Command and Parameter Editor dialog box, click the Query Builder button. The Query Builder dialog box isn't easy to use in its default size, so expand the window until it's at a reasonable size, as shown in Figure 3-15.
4. Click the top part of the dialog box next to the Player table and select Add Table from the context menu.



	Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...
	PlayerName		Player	✓	Ascending	1		
	PlayerManufacturerID		Player	✓				

```
SELECT PlayerName, PlayerManufacturerID
FROM  Player
ORDER BY PlayerName
```

Figure 3-15. *The Query Builder dialog box*

5. From the Add Table dialog box, select the Manufacturer table, and then click OK. This adds the Manufacturer table and shows the relationship it has with the Player table, as shown in Figure 3-16.

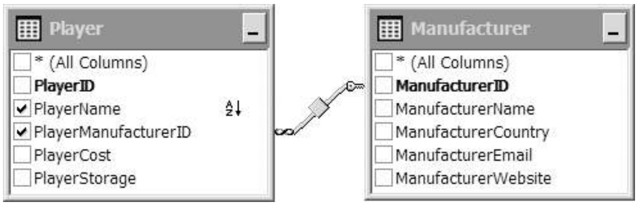
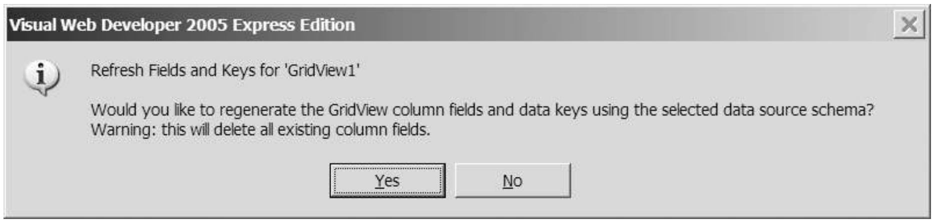


Figure 3-16. *Relationships are shown in the Query Builder dialog box*

6. Uncheck the PlayerManufacturerID check box in the Player table, and check the ManufacturerName check box in the Manufacturer table. This updates both the column list in the second part of the Query Builder dialog box and the SQL view in the third part.

7. Click OK to close the Query Builder dialog box, and then click OK to close the Command and Parameter Editor dialog box.

8. Open the Tasks menu for `GridView1` and select the Refresh Schema option. You're presented with a confirmation dialog box, as shown in Figure 3-17.



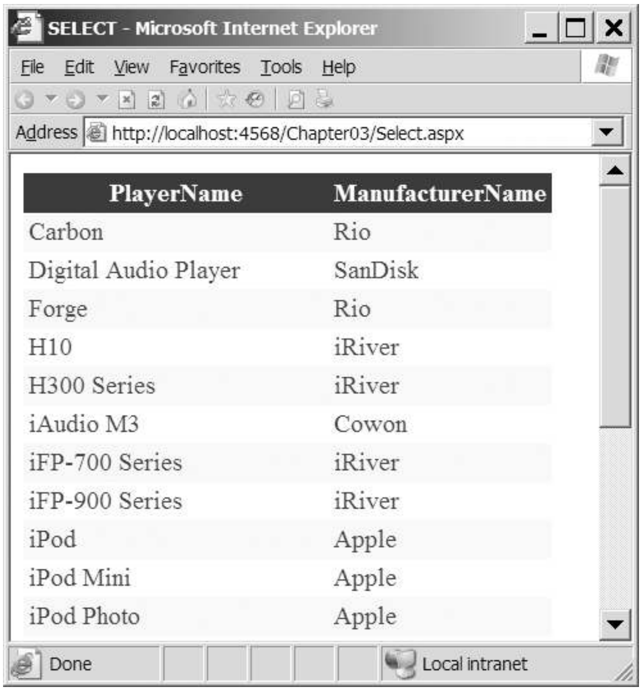
Refresh Fields and Keys for 'GridView1'

Would you like to regenerate the GridView column fields and data keys using the selected data source schema?
Warning: this will delete all existing column fields.

Figure 3-17. *You must confirm that you want to change the schema for a GridView.*

9. Click Yes to confirm that you want to refresh the `GridView`. You'll see that the `GridView` has been updated correctly.

10. Open the page. The new query runs against the database, and the results are returned, as shown in Figure 3-18.


SELECT - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:4568/Chapter03/Select.aspx
Done
Local intranet

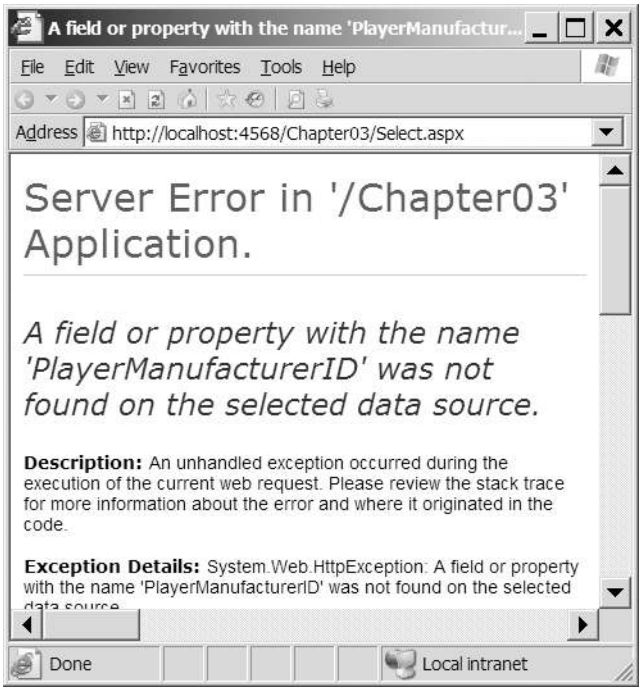
PlayerName	ManufacturerName
Carbon	Rio
Digital Audio Player	SanDisk
Forge	Rio
H10	iRiver
H300 Series	iRiver
iAudio M3	Cowon
iFP-700 Series	iRiver
iFP-900 Series	iRiver
iPod	Apple
iPod Mini	Apple
iPod Photo	Apple

Figure 3-18. *Results of a SELECT query against the Player and Manufacturer table*

How It Works

Before we look at what actually changed in the query, it's worth taking a quick detour into the interaction of the `SqlDataSource` and `GridView`.

The first couple of steps of the example were concerned with changing the query that you are executing. When you closed the Command and Parameter Editor dialog box for the `SelectQuery` property, `SqlDataSource1` was updated with the new `SELECT` query—with PlayerManufacturerID replaced by ManufacturerName—but `GridView1` was still looking for the old column. If you had run the page at this point, you would have seen an error message instead of any results, as shown in Figure 3-19.


A field or property with the name 'PlayerManufactur...
File Edit View Favorites Tools Help
Address http://localhost:4568/Chapter03/Select.aspx
Done
Local intranet

Server Error in '/Chapter03' Application.

A field or property with the name 'PlayerManufacturerID' was not found on the selected data source.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Web.HttpException: A field or property with the name 'PlayerManufacturerID' was not found on the selected data source

Figure 3-19. *A BoundField must show a column from the data source.*

If you had been using automatically generated columns, this wouldn't have been a problem, as the ManufacturerName would have been returned. However, as you saw in the first example in this chapter, you're explicitly saying which columns you want to show, so you would have gotten an error.

To avoid this error, you refreshed the schema for `GridView1` before running the page. When the schema is refreshed, Visual Web Developer looks at the specified data source and replaces the `<Columns>` collection with a new collection for the data source as it stands at that point in time.

Now, let's look at the changes to the `SELECT` query that you're running. The query has been changed to the following:

```
SELECT Player.PlayerName, Manufacturer.ManufacturerName
FROM Player INNER JOIN Manufacturer
  ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
ORDER BY Player.PlayerName
```

First, notice that you're now using the full name syntax to refer to the columns. Although the database doesn't have any column names that could clash, it's good practice when returning data from multiple tables to use the full name, even if you don't have to do so. Not only does it make it easier to see which table a column comes from, it should also make the query a little faster, because the database won't need to work out which table contains the column since you've already specified it.

The following `FROM` clause, and in particular the `INNER JOIN` you added, is the important part of the query:

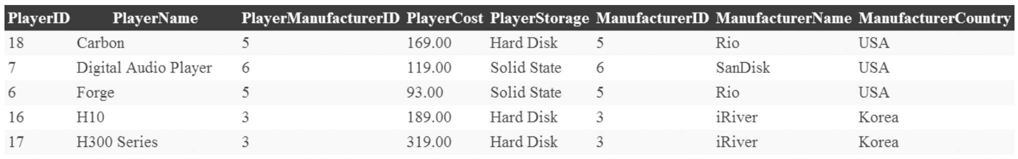
```
FROM Player INNER JOIN Manufacturer
  ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
```

When joining tables using an `INNER JOIN`, the database combines both tables into one "supertable" and returns only records that exist on both sides of the join. It does this by using the table specified after the `FROM` clause as the master table and by appending matching records in the table specified after the `INNER JOIN` based on the `ON` criteria.

■ **Note** Using `INNER JOIN` rather than simply `JOIN` should give you a clue that this isn't the only type of join. SQL also defines `RIGHT JOIN`, `LEFT JOIN`, and `CROSS JOIN`, which allow you to change how the tables are actually joined. The `INNER JOIN` is by far the most commonly used type of join.

The `ON` criterion specifies how the two tables you're joining are being joined. In this case, you specify that the PlayerManufacturerID column in the Player table is joined to the ManufacturerID column in the Manufacturer table. This tells the database that the two columns are equal and to combine the tables based on this.

If you could view the constructed supertable for this join of the Player and Manufacturer table, you would initially see all of the columns in both tables, as shown in truncated form in Figure 3-20.

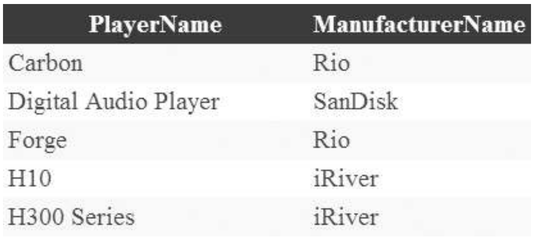


PlayerID	PlayerName	PlayerManufacturerID	PlayerCost	PlayerStorage	ManufacturerID	ManufacturerName	ManufacturerCountry
18	Carbon	5	169.00	Hard Disk	5	Rio	USA
7	Digital Audio Player	6	119.00	Solid State	6	SanDisk	USA
6	Forge	5	93.00	Solid State	5	Rio	USA
16	H10	3	189.00	Hard Disk	3	iRiver	Korea
17	H300 Series	3	319.00	Hard Disk	3	iRiver	Korea

Figure 3-20. *A "supertable" from the join of the Player and Manufacturer tables*

As you can see, you have the PlayerID, PlayerName, PlayerManufacturerID, PlayerCost, and PlayerStorage columns from the Player table, and the ManufacturerID, ManufacturerName, and ManufacturerCountry columns from the Manufacturer table. The supertable contains two more columns that aren't visible in Figure 3-20: ManufacturerEmail and ManufacturerWebsite. In addition, both the PlayerManufacturerID and ManufacturerID columns have the same value, as these are the columns that you specified in the `INNER JOIN` statement.

The query specifies that only the Player.PlayerName and Manufacturer.ManufacturerName columns should be returned, and the remaining columns are ignored, as shown in Figure 3-21.



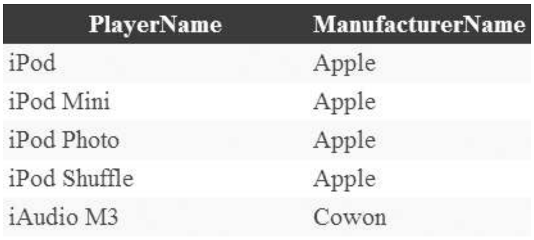
PlayerName	ManufacturerName
Carbon	Rio
Digital Audio Player	SanDisk
Forge	Rio
H10	iRiver
H300 Series	iRiver

Figure 3-21. *Returning a subset of the columns from the supertable*

When you looked at sorting columns in the previous example, you learned that you can sort a set of results by more than one column. The same is also true if you have multiple tables, and you can sort across any of the columns that are in the supertable. So to sort by ManufacturerName and then by PlayerName, you would add both of these to the `ORDER BY` clause in the order you wanted the sorting to take place, like so:

```
SELECT Player.PlayerName, Manufacturer.ManufacturerName
FROM Player INNER JOIN Manufacturer
  ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
ORDER BY Manufacturer.ManufacturerName, Player.PlayerName
```

So, rather than having a list of Players sorted by name, you would now have a list of Players sorted by the name of the Manufacturer and then the name of the Player, as shown in Figure 3-22.



PlayerName	ManufacturerName
iPod	Apple
iPod Mini	Apple
iPod Photo	Apple
iPod Shuffle	Apple
iAudio M3	Cowon

Figure 3-22. *You can also sort using columns from different tables.*

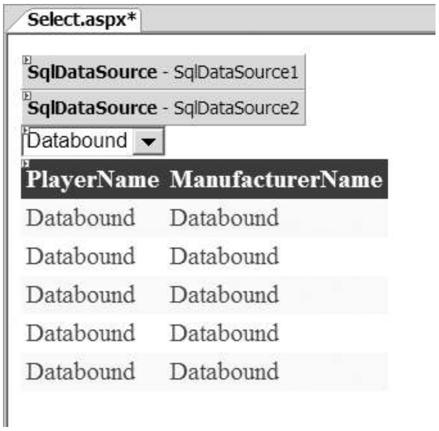
■ **Note** It's possible to have multiple joins in the same SQL query and also to mix different join types within the same SQL query. Joins can quickly become hideously complex. You can find more information about joins and SQL in general in *The Programmer's Guide to SQL* by Christian Darie and Karli Watson (1-59059-218-2; Apress, 2003).

Try It Out: Filtering the Results

In the previous three examples, you've come from writing a simple query that takes data from one table to being able to order the results you get and join two tables to retrieve user-friendly data. Now, let's look at how to filter the results of your queries to return only the results that meet the criteria you specify.

You filter the results of a query by using a `WHERE` clause to constrain the records that are returned. In this next example, you'll again build on the previous example and allow users to select the Manufacturer in which they're interested. Once a selection has been made, only the Players for that Manufacturer will be returned. Follow these steps:

1. If you've closed `Select.aspx` from the previous example, reopen it.
2. In the Design view, add a second `SqlDataSource` to the page. Then add a `DropDownList` from the Standard section of the Toolbox to the page above the `GridView`. Your page should look something like Figure 3-23.


Select.aspx*
SqlDataSource - SqlDataSource1
SqlDataSource - SqlDataSource2
Databound

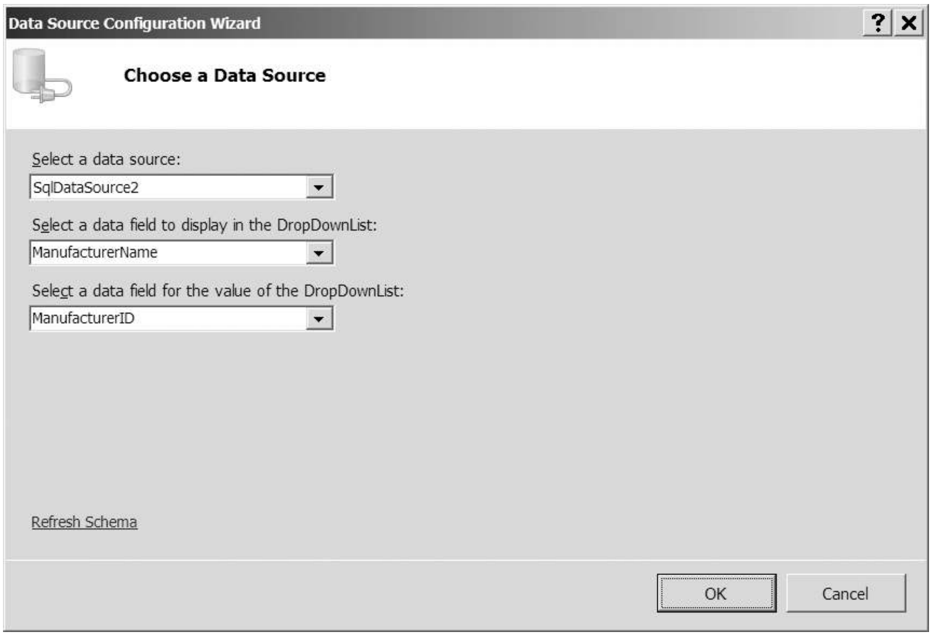
PlayerName	ManufacturerName
Databound	Databound
Databound	Databound
Databound	Databound
Databound	Databound
Databound	Databound

Figure 3-23. *Adding the second SqlDataSource and a DropDownList*

3. Click `SqlDataSource2`, the second `SqlDataSource`, and select Configure Data Source from its Tasks menu.
4. On the Choose Your Data Connection step, select `SqlConnectionString` from the drop-down list, and then click Next.
5. On the Configure the Select Statement step, select Manufacturer from the Name drop-down list, and then select the ManufacturerID and ManufacturerName columns.
6. Click the ORDER BY button. In the Add ORDER BY Clause dialog box, select ManufacturerName in the first Sort By drop-down list and leave the sort order as Ascending. Then click OK. Click Next to continue with the wizard.
7. Test that the Manufacturers are being returned correctly by clicking the Test Query button on the Test Query step. Once you're satisfied that the query works as expected, click Finish to close the wizard.

8. Select the drop-down list, `DropDownList1`, and open the Tasks menu. Check the Enable AutoPostBack check box. Select the Choose Data Source option.

9. On the Choose a Data Source step, select `SqlDataSource2` as the data source and ManufacturerName as the data field to display, as shown in Figure 3-24. Click OK to close the wizard.



Select a data source: SqlDataSource2

Select a data field to display in the DropDownList: ManufacturerName

Select a data field for the value of the DropDownList: ManufacturerID

Figure 3-24. *The DropDownList Data Source Configuration Wizard*

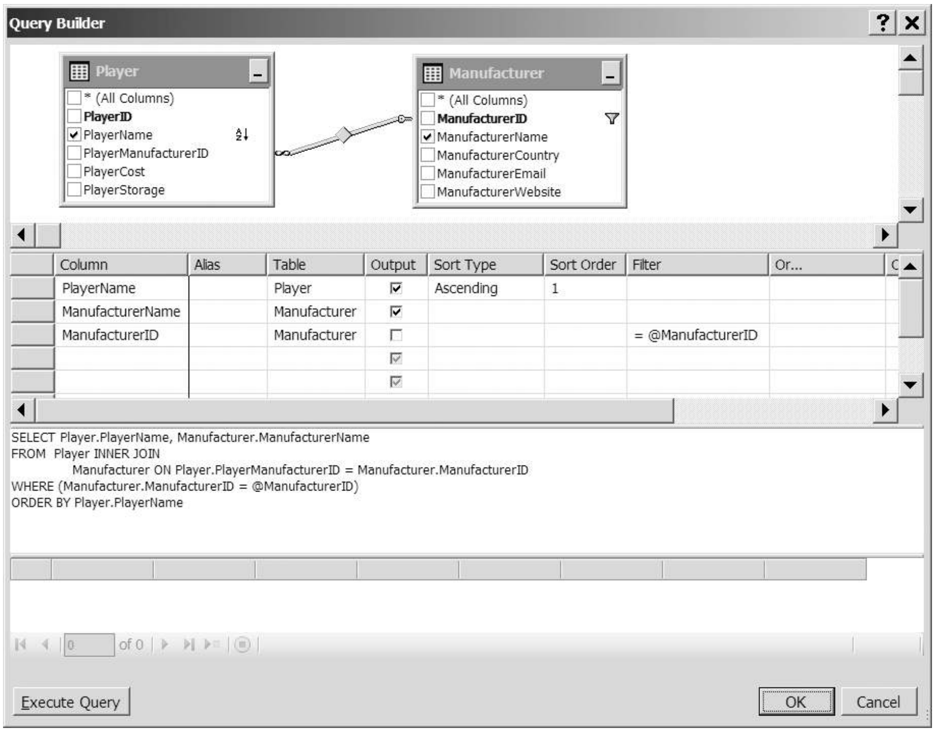
10. Select `SqlDataSource1` and click the ellipsis in the `SelectQuery` property to launch the Command and Parameter Editor dialog box.

11. Click the Add Parameter button and give the parameter a name of `ManufacturerID`. Then select Control as the Parameter source and `DropDownList1` as the ControlID.

12. Click the Query Builder button, and then expand the Query Builder dialog box to a usable size.

13. In the top of the Query Builder dialog box, click the check box next to the ManufacturerID column in the Manufacturer table. In the list of columns in the query, uncheck the Output column next to ManufacturerID.

14. In the Filter column, enter `@ManufacturerID`. Make sure that the filtered column is set correctly. In the table view at the top, you should see that the ManufacturerID column is highlighted with an icon to show that it is filtered. The SQL query also should have changed to show that the ManufacturerID column is being filtered using `@ManufacturerID`, as shown in Figure 3-25.



Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...
PlayerName		Player	✓	Ascending	1		
ManufacturerName		Manufacturer	✓				
ManufacturerID		Manufacturer	☐			= @ManufacturerID	
			✓				
			✓				

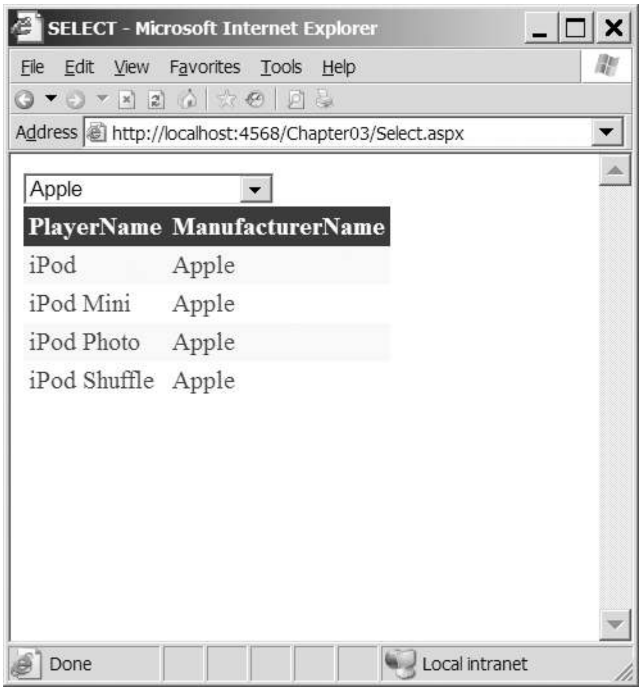
```
SELECT Player.PlayerName, Manufacturer.ManufacturerName
FROM  Player INNER JOIN
         Manufacturer ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
WHERE (Manufacturer.ManufacturerID = @ManufacturerID)
ORDER BY Player.PlayerName
```

Figure 3-25. *Modifying the SELECT query to add filtering*

15. Click OK to close the Query Builder dialog box, and then click OK again to close the Command and Parameter Editor dialog box.

16. Open the page. You'll see that the drop-down list is populated with the available Manufacturers in the database and the grid shows only the Players manufactured by Apple, as shown in Figure 3-26.

17. Select any of the other Manufacturers from the drop-down list to see that the list of Players is indeed modified to display only the correct list of manufactured Players.


SELECT - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:4568/Chapter03/Select.aspx
Apple
Done
Local intranet

PlayerName	ManufacturerName
iPod	Apple
iPod Mini	Apple
iPod Photo	Apple
iPod Shuffle	Apple

Figure 3-26. *Players can be filtered by Manufacturer*

How It Works

Although you've looked at several examples so far in the book, this is the first page you've seen that starts to give you an idea of what's possible with data-driven pages—you've responded to a user selection to modify the results that are returned as part of the query.

We'll look at this example in two parts: how to populate the drop-down list with the list of Manufacturers in the database, and then the changes necessary to make the `GridView`'s data source handle the filtering.

Populating the DropDownList Web Control

You used a new `SqlDataSource`, `SqlDataSource2`, which returns all of the Manufacturers in the database in alphabetical order by executing a simple `SELECT` query:

```
SELECT ManufacturerID, ManufacturerName
FROM Manufacturer
ORDER BY ManufacturerName
```

Using the Configure Data Source wizard, you specified `SqlDataSource2` as the data source for the drop-down list, as well as the text to display and the value to use from the data source. Looking at the markup generated from this wizard, you can start to see what is actually happening:

```
<asp:DropDownList ID="DropDownList1" runat="server"
  DataSourceID="SqlDataSource2"
  DataTextField="ManufacturerName"
  DataValueField="ManufacturerID"
  AutoPostBack="True">
</asp:DropDownList>
```

As you can see, the DataSourceID property is set to the name of the data source that you want to use.

The DataTextField and DataValueField properties are used to control what is shown on the page to the user and what is used as the value of the user's selection. So, by setting ManufacturerName as the DataTextField property value, you're telling ASP.NET to show the Manufacturer names—Apple, Cowon, Creative, and so on. By setting the DataValueField property to ManufacturerID, you're saying that you want the ID of the Manufacturer—1, 2, 3, and so on—to be returned when you request the SelectedValue of the drop-down list.

The AutoPostBack property is set to True to cause any selection changes in the drop-down list to automatically cause a postback to the server to update the grid that is displayed. If you left AutoPostBack set to the default value of False, you would need to add a button (or some other Web control) to the page to force the postback. By having an automatic postback, the page will react immediately to the user's selection.

So you have a drop-down list that contains the different values that you want to filter. Now, let's look at how the results are filtered for the GridView.

Filtering the GridView's Data Source

The GridView simply displays the data that it's told to display from the data source that it's bound to; in this case, SqlDataSource1. To filter the results, you change what the data source is returning from the database by modifying the SELECT query to take into account the value that you've selected in DropDownList1.

First, let's look at the SELECT query that you now use to interrogate the database. This is the same query as you used in the previous example, but with the addition of the WHERE clause (shown in bold):

```
SELECT Player.PlayerName, Manufacturer.ManufacturerName
FROM Player INNER JOIN Manufacturer
  ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
WHERE (Manufacturer.ManufacturerID = @ManufacturerID)
ORDER BY Player.PlayerName
```

The query still returns the Player and Manufacturer names ordered alphabetically on the Player's name. Here, you use the WHERE clause to constrain the results that are returned. We'll look at the WHERE clause in a lot more detail in the next section, but the simple one that you used here should be quite easy to understand.

The WHERE clause compares the ManufacturerID column in the Manufacturer table to a parameter called @ManufacturerID. If the ManufacturerID matches the value of @ManufacturerID, the row will be returned. So, if you select Apple as the Manufacturer, which has a value of 1, only the Players that have a ManufacturerID of 1 will be returned.

But what's `@ManufacturerID`? It's a parameter to the query, as indicated by `@` at the front of its name. *Parameters* in SQL are the means whereby you can pass information into a query at runtime without making any changes to the query. You could have written (quite a lot of) code to extract the value of the Manufacturer that you want to filter for and changed the query that you're running at runtime, but using parameters allows a lot of the hard work to be done for you. You can use the same query every time and simply vary the value of the parameter that you pass into the query.

■**Note** Only when using the `SqlClient` data provider to connect to SQL Server 2005 can you use named parameters. Neither the `Odbc` data provider when connecting to MySQL 5.0 nor the `OleDb` data provider when connecting to any database (such as Microsoft Access) support named parameters.Therefore, the queries for those databases are written a little differently. See the "Connecting to MySQL 5.0 and Microsoft Access" section later in this chapter for more information.

The `@ManufacturerID` parameter is defined for `SqlDataSource1` using the following markup:

```
<SelectParameters>
  <asp:ControlParameter ControlID="DropDownList1"
    Name="ManufacturerID" PropertyName="SelectedValue" />
</SelectParameters>
```

The `<SelectParameters>` element indicates that all the parameters that you're adding here are to be applied to the `SelectCommand` of the data source. You add one parameter here, a `ControlParameter`, which takes its value from another Web control on the page. Table 3-2 lists the various parameters you can use. We'll come back to these in more detail in later chapters, where you'll see a lot more examples of using parameters, but for now we'll concentrate on the `ControlParameter`, the one you're using here.

Table 3-2. *Query Parameter Types*

Name	Description
`ControlParameter`	Takes its value from another Web control on the same page. The Web control to use is specified by `ControlID`, and the property on the Web control to retrieve the value from is indicated by `PropertyName`.
`CookieParameter`	Uses a value specified in a cookie, indicated by the `CookieName` attribute, to set the value of the parameter.
`FormParameter`	Uses a value specified in a form variable, indicated by the `FormName` attribute, to set the value of the parameter.
`QueryStringParameter`	Uses a value in the page's query string, indicated by the `QueryStringField` attribute, to set the value of the parameter.
`SessionParameter`	Uses a value in the user's session, indicated by the `SessionField` attribute, to set the value of the parameter.

All parameters, irrespective of their type, must specify a `Name` property, which is the name used to refer to the parameter within the query. You set a name of `ManufacturerID` here, and this will be passed to the query as `@ManufacturerID`. The leading `@` is added automatically by the `SqlClient` data provider; if you were to add it to the name in the parameter definition, the result would be `@@ManufacturerID`, which isn't what you want.

■ **Note** Several other properties are common across all the parameter types. In most instances, you won't need to use them. However, one that can be quite important is the `DefaultValue` parameter, which sets the value that is used if the parameter that you've defined cannot be evaluated. Examples of all of these properties are available in the MSDN documentation.

For the rest of the parameter definition, you use the `ControlID` and `PropertyName` attributes to specify the Web control to interrogate and the property you want from the Web control. You're filtering the Players based on a Manufacturer selected in the drop-down list, and so you use the name of the Web control, `DropDownList1`, as the `ControlID`. You set the `PropertyName` to `SelectedValue`, to specify from which of the Web control's properties you want to retrieve the parameter's actual value.

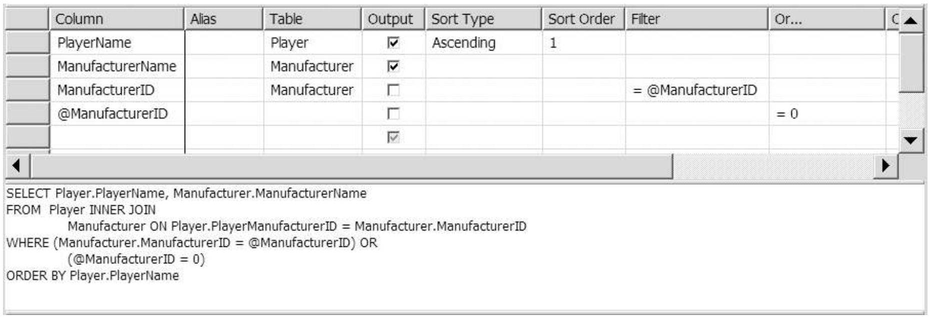
Try It Out: Filtering Results and Showing All Results

In the previous example, you saw how the results could be filtered, but there was no way to show the Players for all of the Manufacturers at the same time. It is possible to show both filtered and nonfiltered results using the same Web controls, as you'll see in this example. Follow these steps:

1. If you've closed `Select.aspx` from the previous example, reopen it.
2. Switch to Design view. Select the drop-down list, `DropDownList1`.
3. In the Properties window, switch to the Events view (by clicking the lightning bolt icon). Under the Data section, double-click in the `DataBound` event to create the `DropDownList1_DataBound` event handler. Enter the following code within the event:

```
DropDownList1.Items.Insert(0,
new ListItem("-- All Manufacturers --", "0"));
```

4. Switch back to Design view and select the `SqlDataSource1`. Click the ellipsis in the `SelectQuery` property to launch the Command and Parameter Editor dialog box.
5. Click the Query Builder button and expand the Query Builder dialog box to a usable size.
6. In the list of columns in the middle of the screen, add a new column called @ManufacturerID, uncheck the Output column, and in the first Or... column, enter a value of 0. You should have the column list and SQL query shown in Figure 3-27.



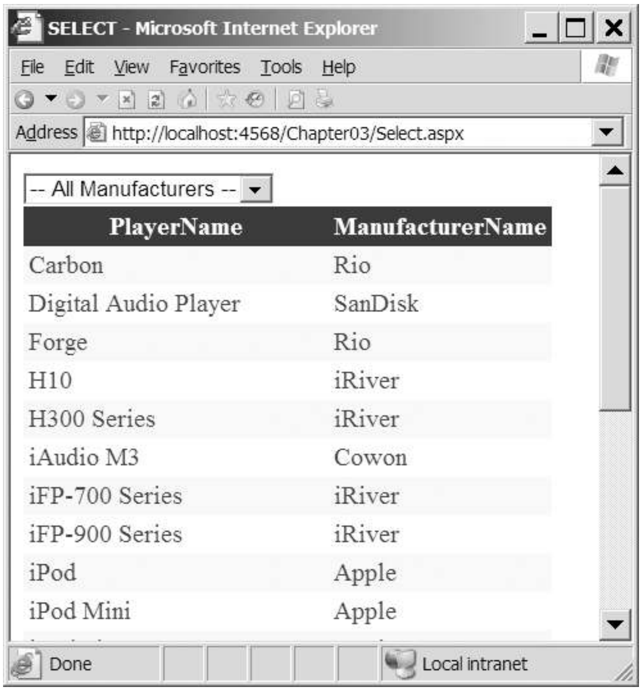
Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...
PlayerName		Player	✓	Ascending	1		
ManufacturerName		Manufacturer	✓				
ManufacturerID		Manufacturer	☐			= @ManufacturerID	
@ManufacturerID			☐				= 0
			✓				

```
SELECT Player.PlayerName, Manufacturer.ManufacturerName
FROM  Player INNER JOIN
          Manufacturer ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
WHERE (Manufacturer.ManufacturerID = @ManufacturerID) OR
          (@ManufacturerID = 0)
ORDER BY Player.PlayerName
```

Figure 3-27. *Adding a second column to the WHERE clause*

7. Close OK to close the Query Builder, and then click OK again to close the Command and Parameter Editor dialog box.

8. Open the page. You'll see that the drop-down list has a new first entry of -- All Manufacturers --, and the grid itself is populated with the Players for all the Manufacturers, as shown in Figure 3-28.


SELECT - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:4568/Chapter03/Select.aspx
-- All Manufacturers --
Done
Local intranet

PlayerName	ManufacturerName
Carbon	Rio
Digital Audio Player	SanDisk
Forge	Rio
H10	iRiver
H300 Series	iRiver
iAudio M3	Cowon
iFP-700 Series	iRiver
iFP-900 Series	iRiver
iPod	Apple
iPod Mini	Apple

Figure 3-28. *The Players for all Manufacturers can now be viewed.*

9. Select any of the Manufacturers from the drop-down list, and you'll see that the list of Players is still modified, depending on your selection.

How It Works

With a little more work (and our first line of code!), you've modified the page so that you can view all of the Players in the database or a list filtered by Manufacturer.

First, you changed the query to return the Players from the database by adding an `OR` to the `WHERE` clause:

```
SELECT Player.PlayerName, Manufacturer.ManufacturerName
FROM Player INNER JOIN Manufacturer
  ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
WHERE (Manufacturer.ManufacturerID = @ManufacturerID)
  OR (@ManufacturerID = 0)
ORDER BY Player.PlayerName
```

There are now two parts to the clause, and these are `OR`'d together; if either of the parts is true, then the row that you're dealing with will be returned.

The first part of the `OR` statement works as it did in the previous example: it filters the results based on the value of the `@ManufacturerID` parameter. The second part of the `OR` statement will return true if the value of the `@ManufacturerID` parameter is equal to 0. This isn't a valid Manufacturer in the database, and as you'll soon see, it's the value used to indicate -- All Manufacturers --.

Because this is an `OR` statement, the parts of the `WHERE` clause work together to filter the Players. If you have a `@ManufacturerID` that isn't 0, the Players will be filtered for that Manufacturer. If you have a `@ManufacturerID` of 0, all of the Players will be returned.

By populating the drop-down list with the Manufacturers from the database, it will be possible to filter the results based on the Manufacturer, but it removes the ability to not filter the results. The `SELECT` query on the database returns the list of Manufacturers, but it doesn't return a "no manufacturer" entry. In order to allow the user to select to view all of the Players in the database, you need to add an -- All Manufacturers -- entry to the drop-down list.

The `DataBound` event fires for the drop-down list after any data binding has occurred, and the `OnDataBound` property of the `GridView` is used to point to the event handler that you want to execute. In this case, `DropDownList1_DataBound` is called after the query to populate the list of Manufacturers has completed, and at this point, you can add the -- All Manufacturers -- entry:

```
DropDownList1.Items.Insert(0,
  new ListItem("-- All Manufacturers --", "0"));
```

You're inserting a new `ListItem` at position 0 to the `Items` collection of the drop-down list, ensuring that it's the first item in the list and that when the page first loads, all of the Players in the database (the nonfiltered results) are displayed.

In this example, you've seen that you can use the `WHERE` clause to restrict the records that you've returned based on the criteria you specify. Now, let's take a closer look at what else you can do with the `WHERE` clause.

Introducing the WHERE Clause

Not only is the `WHERE` clause used with the `SELECT` query, it can also be used with the `UPDATE` and `DELETE` queries to restrict your actions within the database. You'll look at `UPDATE` and `DELETE` queries starting in Chapter 8, but for now, you'll concentrate on what you can do with the `WHERE` clause. In the previous filtering examples, you've looked at comparing only two values to see if they're equal. Now, let's look at the other operators that are available.

Using Comparison Operators

Table 3-3 describes the standard comparison operators. You should be familiar with most of the comparison operators; they work in the way that you would expect.

Table 3-3. *The SQL Comparison Operators*

Operator	Definition	Example
`=`	Equality	`PlayerManufacturerID = 1`
`<>`	Inequality	`PlayerManufacturerID <> 1`
`<`	Less than	`PlayerManufacturerID < 3`
`>`	Greater than	`PlayerManufacturerID > 4`
`<=`	Less than or equal to	`PlayerManufacturerID <= 3`
`>=`	Greater than or equal to	`PlayerManufacturerID >= 4`
`IS NULL`	Test for null values	`PlayerManufacturerID IS NULL`

The only comparison operator that you may not be unfamiliar with is the `IS NULL` operator. Null values in the database can't be compared to any other value and won't appear in any comparison. So, if you execute the following query, you would expect to return every Player that had a `PlayerManufacturerID` of any value other than 1:

```
SELECT PlayerName
FROM Player
WHERE PlayerManufacturerID <> 1
```

However, this isn't the case, and if null values are allowed for PlayerManufacturerID, you would need to test for this condition explicitly, like so:

```
SELECT PlayerName
FROM Player
WHERE PlayerManufacturerID <> 1 OR PlayerManufacturerID IS NULL
```

This will return the correct results: Players that have a PlayerManufacturerID that isn't equal to 1 and Players that have a null value for PlayerManufacturerID.

Using Logical Operators

You've already seen the `OR` logical operator in action when you filtered the results in the previous example, and I sneaked it in again at the end of the discussion of comparison operators. Table 3-4 lists all of the SQL logical operators.

Table 3-4. *The SQL Logical Operators*

Operator	Definition	Usage
`AND`	Returns true if both conditions are true	`a AND b`
`OR`	Returns true if either condition is true	`a OR b`
`NOT`	Returns true if the condition is false	`NOT a`

Using the IN and BETWEEN Operators

SQL also defines two other handy operators that you can use within the `WHERE` clause to make some of the clauses that you need to execute a little simpler.

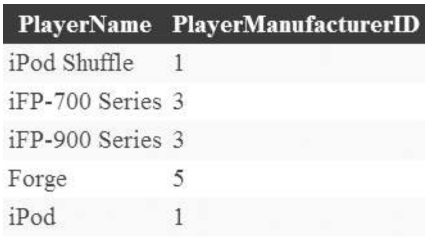
The `IN` operator allows you to specify that you're looking within a series of noncontiguous values and is equivalent to using the `OR` operator to chain together several different equality comparisons. For example, to return all the Players that have a PlayerManufacturerID of 1, 3, or 5, you could execute the following query:

```
SELECT PlayerName, PlayerManufacturerID
FROM Player
WHERE PlayerManufacturerID = 1
  OR PlayerManufacturerID = 3
  OR PlayerManufacturerID = 5
```

The `IN` operator allows you to do this more simply:

```
SELECT PlayerName, PlayerManufacturerID
FROM Player
WHERE PlayerManufacturerID IN (1,3,5)
```

Both queries will return the same results, as shown in Figure 3-29.



PlayerName	PlayerManufacturerID
iPod Shuffle	1
iFP-700 Series	3
iFP-900 Series	3
Forge	5
iPod	1

Figure 3-29. *Selecting from a series of values using the IN operator*

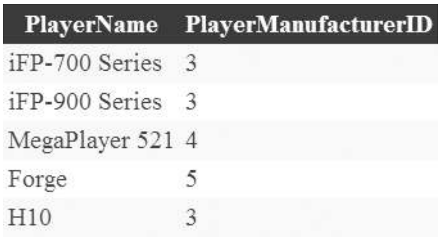
Similarly, you can use the `BETWEEN` operator to specify that you're looking within a contiguous range of values. If you wanted to return all the Players that have a PlayerManufacturerID of 3, 4, or 5, you could use the `AND`, `<=`, and `>=` operators in conjunction to retrieve the correct Players:

```
SELECT PlayerName, PlayerManufacturerID
FROM Player
WHERE PlayerManufacturerID >= 3 AND PlayerManufacturerID <= 5
```

The BETWEEN operator allows you to simplify this, like so:

```
SELECT PlayerName, PlayerManufacturerID
FROM Player
WHERE PlayerManufacturerID BETWEEN 3 AND 5
```

Both of these queries will return the results shown in Figure 3-30.



PlayerName	PlayerManufacturerID
iFP-700 Series	3
iFP-900 Series	3
MegaPlayer 521	4
Forge	5
H10	3

Figure 3-30. *Selecting from a range of values using the BETWEEN operator*

One word of warning when using the BETWEEN operator: You'll notice that you use AND to specify the upper and lower values of the range. This isn't the same as using the AND operator as a logical comparison, so don't get the two confused.

Connecting to MySQL 5.0 and Microsoft Access

Now that we've looked at the SQL Server 2005 pages, it's time to see what the differences are when you use MySQL 5.0 and Microsoft Access pages. As I've noted earlier, when you use the SqlDataSource, you're shielded quite a lot from the intricacies of data access, so the pages to connect to the three databases are the same apart from one area: parameters to queries.

■ **Note** In the code download for this chapter, and indeed for every chapter (available from the Downloads section of the Apress Web site at http://www.apress.com), you'll find two folders: odbc and oledb. The odbc folder contains a copy of the SQL Server 2005-based pages modified to connect to MySQL 5.0, and the oledb folder contains modified pages to connect to Microsoft Access.

Let's first look at the connection strings for the database types, and then review how MySQL and Access handle query parameters.

Connection Strings

So, what sort of things does a connection string contain? This depends on the data provider you're using, and in the case of OleDb and Odbc data providers, also on the underlying provider/driver you're using.

As you've already seen, the connection strings should be stored in the `<connectionStrings>` section of `Web.config`. In the code download for all of the chapters, you'll see that there are three different connection strings defined in `Web.config`:

```
<connectionStrings>
  <add name="SqlConnectionString"
    connectionString="Data Source=localhost\band;Initial Catalog=Players;
      Persist Security Info=True;User ID=band;Password=letmein"
    providerName="System.Data.SqlClient"/>
  <add name="OdbcConnectionString"
    connectionString="Driver={MySQL ODBC 3.51 Driver};
      server=localhost;database=players;uid=band;pwd=letmein;"
    providerName="System.Data.Odbc" />
  <add name="OleDbConnectionString"
    connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
      Data Source=C:\band\players.mdb"
    providerName="System.Data.OleDb" />
</connectionStrings>
```

We've already looked at `SqlConnectionString` earlier in this chapter.

For the `OdbcConnectionString`, the first thing you specify is the ODBC `Driver` being used. Each driver has its own connection string specification, and if you don't specify the driver, the rest of the connection string cannot be parsed correctly.

For MySQL 5.0, the connection string must specify the `server`, `database`, `uid`, and `pwd` that are to be used to connect to the database.

The `OleDbConnectionString` specifies the OLE DB `Provider` being used, and then the connection string details specific to the provider. When using the `JET` provider to connect to Microsoft Access, you specify the `Data Source` as the filename, including the path, to the `.mdb` file.

■ **Note** You have a multitude of options for connecting to data sources. For a comprehensive list, see `http://www.connectionstrings.com`.

Parameters and Queries

The only difference when using MySQL 5.0 or Access for the examples in this chapter involves the parameters for queries.

With named parameters, the `SqlClient` data provider can determine which parameter is required. In the examples, you used a named parameter called `@ManufacturerID`, and even though the parameter was used twice in the query, only one parameter was defined.

The `Odbc` data provider when connecting to MySQL and the `OleDb` data provider when connecting to any database rely on the order that the parameters are added to the `SqlDataSource` to determine how they're inserted into the query. This means that they require a slightly different query than the `SqlClient` data provider. Rather than the `@` syntax to refer to a parameter, you use a question mark (`?`) instead, like so:

```
SELECT Player.PlayerName, Manufacturer.ManufacturerName
FROM Player INNER JOIN Manufacturer
  ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
WHERE Player.PlayerManufacturerID = ?
  OR ? = 0
```

As there are two parameters (indicated by the two question marks), you must specify two parameters, even if, as in this case, the two parameters take the same value. As these aren't named parameters, you don't need to specify a name (if you do, it will simply be ignored):

```
<SelectParameters>
  <asp:ControlParameter ControlID="DropDownList1"
    PropertyName="SelectedValue" />
  <asp:ControlParameter ControlID="DropDownList1"
    PropertyName="SelectedValue" />
</SelectParameters>
```

If you look at the two modified `Select.aspx` files in the `odbc` and `oledb` folders, you'll see that apart from these two changes—the change to the query and the addition of the duplicate parameter—the pages are the same.

Summary

This chapter covered quite a lot of ground. You've learned about the following:

- You were introduced to the `GridView` and `SqlDataSource`, and through the course of the examples, saw how easy it is to build data-driven pages with a minimum of code.
- You were introduced to the `SELECT` query by writing a simple query for one table.
- You expanded your understanding of the `SELECT` query by looking at ordering results and joining tables to retrieve information from both tables in the same query.
- You looked at filtering the results that you returned by using parameters with values taken from other Web controls on the page.
- You looked at the comparison and logical operators you can use with the `WHERE` clause and saw the `IN` and `BETWEEN` operators you can use to simplify the constraints that you add to queries.
- You looked at the connection strings that are required for the SQL Server 2005, MySQL 5.0, and Microsoft Access databases, and saw how to store all three connection strings in the `<connectionStrings>` element of `Web.config`.
- You saw that SQL Server 2005 supports the concept of named parameters where the same parameter can be used several times in the same query. With MySQL 5.0 and Microsoft Access, you need to provide a separate value for each parameter within the query, even if you want to reuse the same parameter.

Now that you've had a taste of how data-driven pages work, we'll move on to look at how you can connect to the database using code, rather than relying on the `SqlDataSource` to do all of the work for you.