CHAPTER 9

The GridView Family

In a superhero sort of way, the GridView has been hiding a second life from you. In the previous chapters, you've been using the GridView to simply display data, but it can do much more than that. In Chapter 7, you saw that with a minimum of work (translated as "very little code"), you can also sort and page the results. In this chapter, you'll see that, again with very little work, you can use the GridView to edit the data that it displays.

As you'll see as you work through this chapter, the GridView allows you to edit and delete data. However, it doesn't allow you to add new data. For that task, you'll need to use one of its siblings—the DetailsView or FormView. Both the DetailsView and FormView allow you to add new data, as well as edit and delete data.

Although the GridView, DetailsView, and FormView are what are displayed on the page, without a data source, they would be rather useless—no data means nothing to display. In the previous chapters, we've looked at using a SqlDataSource to provide the data to the GridView, and we've also looked at using code to handle the connection to the database. The SqlDataSource also has the ability to modify the contents of database. In the majority of cases when using the GridView, DetailsView, and FormView to modify the contents of the database, you'll use a SqlDataSource to do it. You could write all the necessary code to do so, but why reinvent the wheel?

This chapter covers the following topics:

- An introduction to the updatable features of the SqlDataSource
- An overview of the GridView and DetailsView and the Field Web controls that they support
- How to edit and delete existing data in a GridView
- How to use a DetailsView with a GridView, as well as use a DetailsView to edit, delete, and add data
- How to use a FormView, which is similar a DetailsView but allows more control over the way that the information is presented to the user
- How to add validation Web controls to prevent the user from entering incorrect data

The Updatable SqlDataSource

So far, when you've used a SqlDataSource, you've specified a SELECT query and, depending on what you were doing, a set of parameters for that query. For example, to return all the Players for a given Manufacturer, you would define the following SqlDataSource:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  ConnectionString="<%$ ConnectionStrings:SqlConnectionString %>"
  SelectCommand="SELECT PlayerID, PlayerName FROM Player
    WHERE Player.PlayerManufacturerID = @ManufacturerID">
  <SelectParameters>
    <asp:ControlParameter ControlID="ManufacturerList"
      Name="ManufacturerID" PropertyName="SelectedValue" />
  </SelectParameters>
</asp:SqlDataSource>
```

The names that you use should give the game away a little. You have a SelectCommand property specifying the query that you want to execute, and a SelectParameters collection that contains the parameters, if any, for the query.

The SqlDataSource also supports INSERT, UPDATE, and DELETE queries. Not surprisingly, these have matching names to the SELECT query:

- The INSERT query is specified in the InsertCommand property, and its parameters are specified in the InsertParameters collection.
- The UPDATE query is specified in the UpdateCommand property, and its parameters are specified in the UpdateParameters collection.
- The DELETE query is specified in the DeleteCommand property, and its parameters are specified in the DeleteParameters collection.

By setting the necessary property and adding the parameters, you enable the SqlDataSource to perform the corresponding action. But that's not the end of the story. The SqlDataSource defines the queries but doesn't execute them. The Web control that's making use of the SqlDataSource actually executes the query.

The SqlDataSource exposes a method that allows the corresponding query to be executed. The SelectCommand is executed by calling the Select() method, the InsertCommand by calling the Insert() method, the UpdateCommand by calling the Update() method, and the DeleteCommand by calling the Delete() method. Before calling one of these methods, the GridView (or DetailsView or FormView) populates the necessary parameters for the query.

Each of the four queries also has two events that you can handle if necessary: a before action event and an after action event. For example, the Update() method first raises the Updating event, then executes the UPDATE query, before raising the Updated event. Similar events exist for the Select(), Insert(), and Delete() methods.

Within the before event (Selecting, Inserting, Updating, or Deleting), you can perform whatever processing is needed. Maybe you need to check the parameter values and block any incorrect actions. The after event (Selected, Inserted, Updated, or Deleted) allows you to check how many rows were affected by the query and inspect any exceptions that were raised by the query.

In the examples in this chapter, you won't use the SqlDataSource events, or even manually call the SqlDataSource to execute a query. The GridView, DetailsView, and FormView provide the required functionality for you.

We're going to spend the majority of this chapter looking at the GridView and DetailsView. We'll cover the FormView as well, but in less detail. Once you know how the DetailsView works, you'll be able to use a FormView without any problems.

The GridView and DetailsView

As you've already seen, the GridView is used to show a set of results in a table. In the table, one row of data is one row from the database. In contrast, the DetailsView is used to show a single row from the database in a table. One row in the table is, generally, one column from the selected row in the database.

The easiest way to think of these two Web controls is that they have a master-detail relationship. The GridView is used to show the list of results, and the DetailsView shows a single row from the results in more detail. That's not to say that you must use them in conjunction; each can be used separately.

The Field Controls

One of the things that the GridView and DetailsView have in common is that they both contain a collection of Field Web controls. The GridView has a Columns collection, and the DetailsView has a Fields collection. The Field Web controls were introduced in Chapter 7 (see Table 7-5). However, three of the Field Web controls are of special interest when editing or updating data: BoundField, CheckBoxField, and TemplateField.

The BoundField appears as a string of text when viewing data. When editing or updating data, it's displayed as a TextBox. The CheckBoxField is shown as a disabled CheckBox when viewing data, and is enabled when editing or inserting data.

As you learned in Chapter 7, the TemplateField is the most customizable of the Field Web controls. Within the TemplateField, you're free to define whatever look and feel you want for the different parts of the display and also what to display, depending on the mode of the GridView or DetailsView. The templates for the GridView that support displaying data were introduced in Chapter 7 (see Table 7-7). The following are the two additional templates for editing data:

- EditItemTemplate: Specifies the content to be displayed when the GridView or DetailsView is editing data. If not specified when editing data, the ItemTemplate (or AlternatingItemTemplate) will be used, effectively making the Field read-only.

- InsertItemTemplate: Specifies the content to be displayed when adding new data in a DetailsView. If not specified, the EditItemTemplate will be used when adding new data. If there is no EditItemTemplate, the ItemTemplate (or AlternatingItemTemplate) will be used instead, making the Field effectively read-only.

The EmptyDataTemplate

What if the SELECT query that is executed returns no results? The GridView and DetailsView won't have any data to display, so nothing will appear on the page. It's true that the GridView and DetailsView can't make things appear when there's no data, but that doesn't help the user. It would be better to show them some sort of message indicating that no results were returned.

In previous versions of ASP.NET, you were left with a "hack" of using a hidden Web control that you made visible manually if no results were present, to inform the user that there had been a problem. The GridView and DetailsView make this process a little easier by allowing you to add an EmptyItemTemplate, which is displayed whenever there is no data to display.

Here is a very simple declaration for an EmptyItemTemplate:

```
<asp:GridView ID="GridView1" runat="server" ...>
  <EmptyItemTemplate>
    <B>There are no manufacturers to display</B>
  </EmptyItemTemplate>
  <Columns>
    ...
  </Columns>
</asp:GridView>
```

As you'll see in the examples in this chapter, you can also make the EmptyItemTemplate quite functional, rather than just serving as a placeholder for messages.

The Eval() and Bind() Methods

In Chapter 7, when we looked at table binding, you saw that ASP.NET 2.0 added a shorthand method of binding data from the data source to the Web control. The Eval() method gives you easy access to the columns from the data source:

```
<%# Eval("[ManufacturerWebsite]") %>
```

And if you need to format the value from the column, you can add a format string to the Eval() method as well:

```
<%# Eval("[ManufacturerEmail]", "mailto:{0}") %>
```

The Eval() method is fine when all you want to do is show the column. But when you want to allow inline editing, the Eval() method doesn't work.

ASP.NET 2.0 introduces a new method, Bind(), that allows data to flow both ways: from the database to populate the Web control, and then back from the Web control when its value is required to send to the database. It has the same two overloads as the Eval() method. If you want to retrieve the column as it is, you specify the column name:

```
<%# Bind("[ManufacturerWebsite]") %>
```

And if you want to format the column, you specify a format string as well:

```
<%# Bind("[ManufacturerEmail]", "mailto:{0}") %>
```

You should use the method that is most appropriate. When the column is read-only, use the Eval() method. For example, in an ItemTemplate or AlternatingItemTemplate you should use Eval(), as the data will be read-only. If the column is read-write, use the Bind() method. For example, in an EditItemTemplate or an InsertItemTemplate for a column that is to be updated, use the Bind() method. If the column is not going to be updated, use the Eval() method.

As you learned in Chapter 7, all table-binding Web controls can use the `Eval()` method. However, a couple of restrictions apply to using the `Bind()` method:

- Only the `GridView`, `DetailsView`, and `FormView` support the `Bind()` method.
- You can use the `Bind()` method only if the data source is specified using the `DataSourceID` property; that is, if you're using a `SqlDataSource` to query the database.

The examples in this chapter follow both these rules, so you'll be able to use the `Bind()` method without any problems.

Editing Data in a GridView

The `GridView` doesn't support the addition of new data itself. It needs to be combined with the `DetailsView` or `FormView` to allow new data to be added. However, the `GridView` does allow you to update and delete existing data. We'll start out by creating a page for updating data.

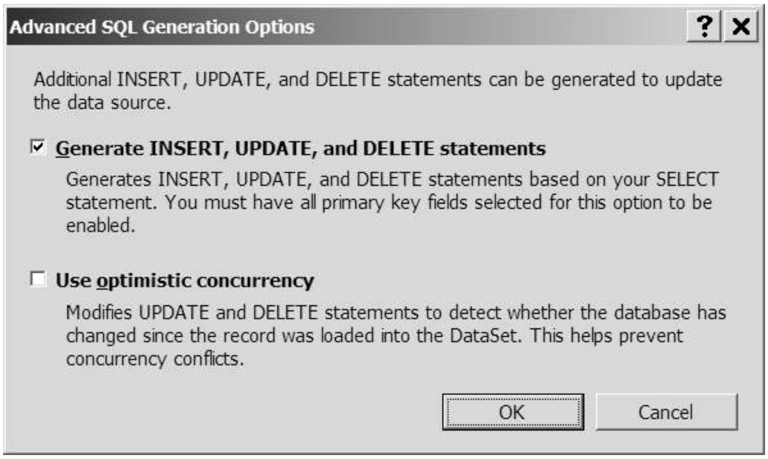
Try It Out: Updating Data in a GridView

In this example, we'll use a `GridView` in a simple page that allows the basic details for a Player to be edited.

1. In Visual Web Developer, create a new Web site at `C:\BAND\Chapter09` and delete the auto-generated `Default.aspx` file.
2. Add a new `Web.config` file to the Web site and add a new setting to the `<connectionStrings />` element:

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

3. Add a new Web Form to the Web site called `Players_Basic.aspx`. Make sure that the Place Code in Separate File check box is unchecked.
4. In the Source view, find the `<title>` tag and change the page title to **Players**.
5. Switch to the Design view and add a `SqlDataSource` to the page. From the Tasks menu, select Configure Data Source.
6. Select `SqlConnectionString` as the data connection, and then click the Next button.
7. Create a query that selects the PlayerID, PlayerName, PlayerManufacturerID, PlayerCost, and PlayerStorage columns from the Player table.
8. Click the Advanced button. In the Advanced SQL Generation Options dialog box, click the Generate INSERT, UPDATE, and DELETE statements check box, as shown in Figure 9-1.


Advanced SQL Generation Options

Additional INSERT, UPDATE, and DELETE statements can be generated to update the data source.

[✓] Generate INSERT, UPDATE, and DELETE statements

Generates INSERT, UPDATE, and DELETE statements based on your SELECT statement. You must have all primary key fields selected for this option to be enabled.

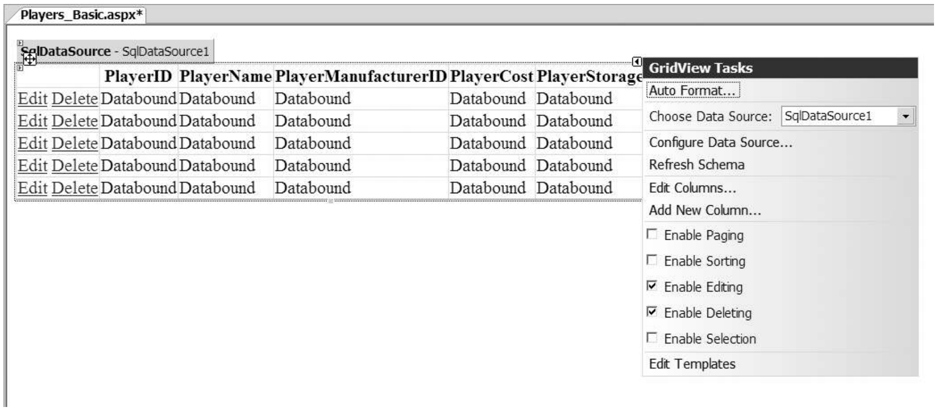
[ ] Use optimistic concurrency

Modifies UPDATE and DELETE statements to detect whether the database has changed since the record was loaded into the DataSet. This helps prevent concurrency conflicts.

Figure 9-1. *Auto-generating INSERT, UPDATE, and DELETE queries*

9. Click the OK button to close the dialog box, and then click the Next button in the Configure Data Source dialog box. You can test the `SELECT` query if you wish on the next step. Click the Finish button to close the Configure Data Source wizard.

10. Add a `GridView` to the page and select `SqlDataSource1` as the data source for the Web control. Select the Enable Editing and Enable Deleting options. This will configure the columns correctly for the `GridView` based on the `SELECT` query and also add Edit and Delete links to each row, as shown in Figure 9-2.


Players_Basic.aspx*

SqlDataSource - SqlDataSource1

	PlayerID	PlayerName	PlayerManufacturerID	PlayerCost	PlayerStorage
Edit Delete	Databound	Databound	Databound	Databound	Databound
Edit Delete	Databound	Databound	Databound	Databound	Databound
Edit Delete	Databound	Databound	Databound	Databound	Databound
Edit Delete	Databound	Databound	Databound	Databound	Databound
Edit Delete	Databound	Databound	Databound	Databound	Databound

GridView Tasks

Auto Format...

Choose Data Source: SqlDataSource1

Configure Data Source...

Refresh Schema

Edit Columns...

Add New Column...

[ ] Enable Paging

[ ] Enable Sorting

[✓] Enable Editing

[✓] Enable Deleting

[ ] Enable Selection

Edit Templates

Figure 9-2. *Enabling editing and deleting for a GridView*

11. On the GridView Tasks menu, click the Edit Templates option. You'll now see that you can enter the details for the EmptyDataTemplate, as shown in Figure 9-3.

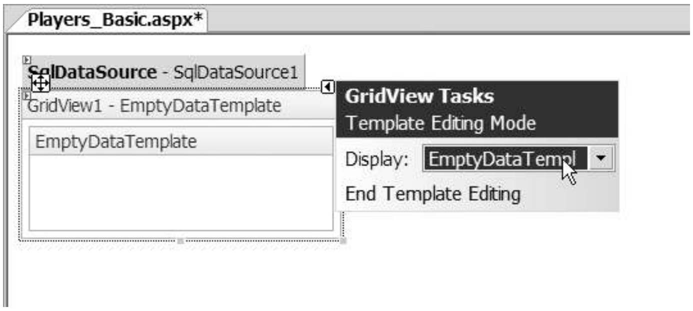
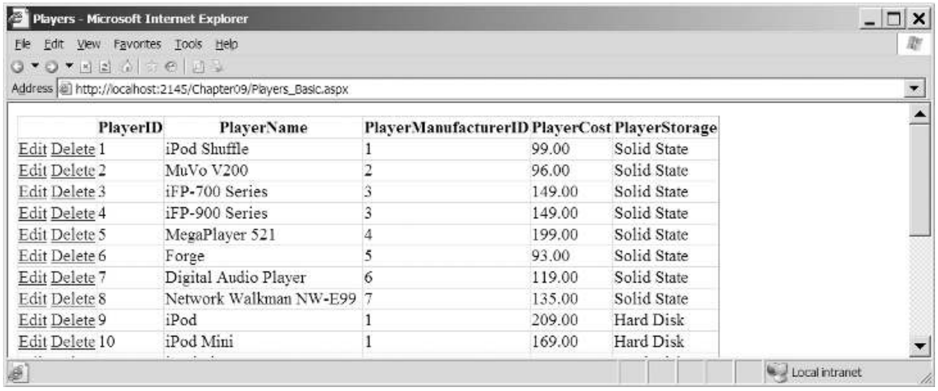


Figure 9-3. *Defining the EmptyDataTemplate*

12. Enter **There are no players to display** for the EmptyDataTemplate, and then click the End Template Editing option on the Tasks menu.

13. Save the page, and then open it in your browser. As shown in Figure 9-4, you'll see the list of all of the Players in the database, along with Edit and Delete links for each row.

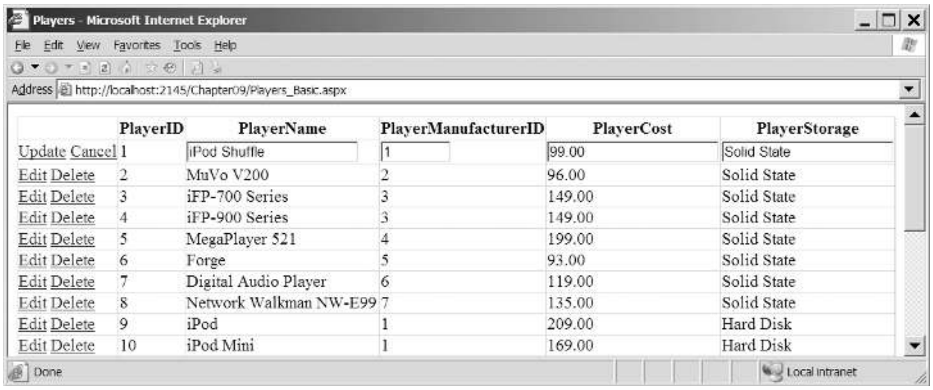

Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Basic.aspx
Local intranet

	PlayerID	PlayerName	PlayerManufacturerID	PlayerCost	PlayerStorage
Edit Delete	1	iPod Shuffle	1	99.00	Solid State
Edit Delete	2	MuVo V200	2	96.00	Solid State
Edit Delete	3	iFP-700 Series	3	149.00	Solid State
Edit Delete	4	iFP-900 Series	3	149.00	Solid State
Edit Delete	5	MegaPlayer 521	4	199.00	Solid State
Edit Delete	6	Forge	5	93.00	Solid State
Edit Delete	7	Digital Audio Player	6	119.00	Solid State
Edit Delete	8	Network Walkman NW-E99	7	135.00	Solid State
Edit Delete	9	iPod	1	209.00	Hard Disk
Edit Delete	10	iPod Mini	1	169.00	Hard Disk

Figure 9-4. *Each row in the GridView has Edit and Delete links.*

14. Click the Edit link for the first row in the table. The page will be posted back to the server and the row placed into Edit mode, as shown in Figure 9-5.

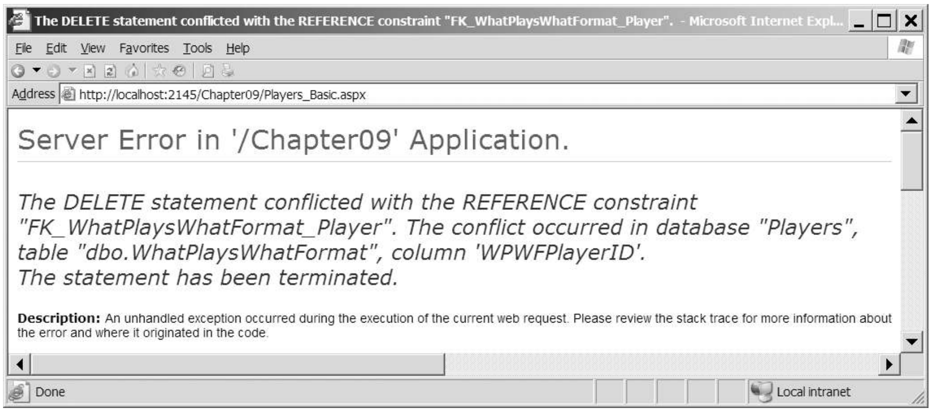
15. Change the data and click the Update link. The page will refresh, and the GridView will be updated to reflect the changes.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Basic.aspx
Done
Local intranet

	PlayerID	PlayerName	PlayerManufacturerID	PlayerCost	PlayerStorage
Update Cancel	1	iPod Shuffle	1	99.00	Solid State
Edit Delete	2	MuVo V200	2	96.00	Solid State
Edit Delete	3	iFP-700 Series	3	149.00	Solid State
Edit Delete	4	iFP-900 Series	3	149.00	Solid State
Edit Delete	5	MegaPlayer 521	4	199.00	Solid State
Edit Delete	6	Forge	5	93.00	Solid State
Edit Delete	7	Digital Audio Player	6	119.00	Solid State
Edit Delete	8	Network Walkman NW-E99	7	135.00	Solid State
Edit Delete	9	iPod	1	209.00	Hard Disk
Edit Delete	10	iPod Mini	1	169.00	Hard Disk

Figure 9-5. *Rows can be edited inline.*

16. Click the Delete link for the first row. Maybe you won't be surprised to see that you get a runtime error, as shown in Figure 9-6.


The DELETE statement conflicted with the REFERENCE constraint "FK_WhatPlaysWhatFormat_Player". - Microsoft Internet Expl...
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Basic.aspx
Done
Local intranet

Server Error in '/Chapter09' Application.

*The DELETE statement conflicted with the REFERENCE constraint "FK_WhatPlaysWhatFormat_Player". The conflict occurred in database "Players", table "dbo.WhatPlaysWhatFormat", column 'WPWFPlayerID'.
The statement has been terminated.*

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.

Figure 9-6. *We can't delete Players from the database.*

17. Switch back to the Design view in Visual Web Developer and add a `Label` between the `SqlDataSource` and `GridView`. Set its `ID` to `lblError`, `Visible` to `False`, `ForeColor` to Red, and remove the `Text` value.

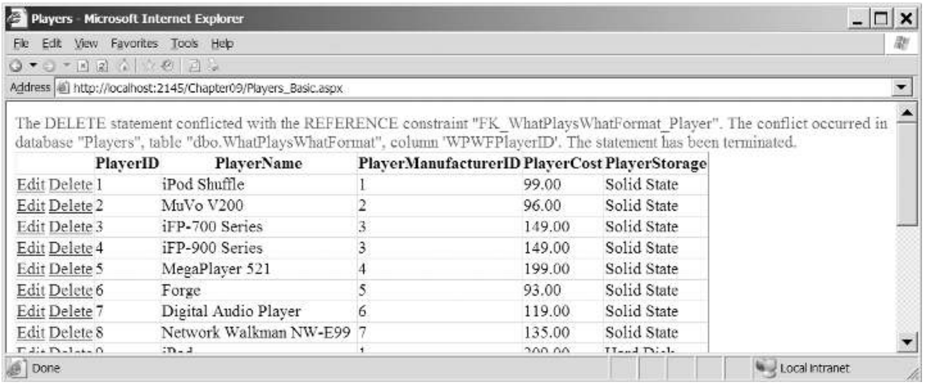
18. Add a `RowDeleted` event to the `GridView` and add the following code to the event handler:

```
protected void GridView1_RowDeleted(object sender,
  GridViewDeletedEventArgs e)
{
  if (e.Exception != null)
  {
    lblError.Visible = true;
    lblError.Text = e.Exception.Message;
    e.ExceptionHandled = true;
  }
}
```

19. Also add a `Page_Load` event, and then add the following code to the event handler:

```
protected void Page_Load(object sender, EventArgs e)
{
  lblError.Visible = false;
}
```

20. Save the page. Now try to delete a Player. Instead of the ASP.NET error message, you'll receive a nicer warning, as shown in Figure 9-7.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Basic.aspx
Done
Local intranet

The DELETE statement conflicted with the REFERENCE constraint "FK_WhatPlaysWhatFormat_Player". The conflict occurred in database "Players", table "dbo.WhatPlaysWhatFormat", column 'WPWFPlayerID'. The statement has been terminated.

	PlayerID	PlayerName	PlayerManufacturerID	PlayerCost	PlayerStorage
Edit Delete	1	iPod Shuffle	1	99.00	Solid State
Edit Delete	2	MuVo V200	2	96.00	Solid State
Edit Delete	3	iFP-700 Series	3	149.00	Solid State
Edit Delete	4	iFP-900 Series	3	149.00	Solid State
Edit Delete	5	MegaPlayer 521	4	199.00	Solid State
Edit Delete	6	Forge	5	93.00	Solid State
Edit Delete	7	Digital Audio Player	6	119.00	Solid State
Edit Delete	8	Network Walkman NW-E99	7	135.00	Solid State

Figure 9-7. *Handling errors that occur during database operations*

How It Works

With only a few lines of code, you've built a page that allows you to edit information for the Players in the database. It's not perfect though, as the error you received when trying to delete a Player from the database shows.

Deleting Players from the database isn't the only problem, however. The page that you've built has at least four problems:

- **The page isn't very user-friendly**. Instead of showing the name of the Manufacturer, you're showing the PlayerManufacturerID. Now you may know that a value of 1 for PlayerManufacturerID corresponds to Apple, but what about the others?
- **You can't delete a Player from the database**. As you learned in Chapter 8, whenever you modify the database, you're bound by the database rules. In the sample database, Players are part of a relationship with the WhatPlaysWhatFormat table. As the error in Figure 9-6 shows, the Player is part of a relationship with the WhatPlaysWhatFormat table, and the constraints in the database prevent you from deleting a Player without deleting the referencing rows from the WhatPlaysWhatFormat table first.
- **You can't add a new Player to the database**. Although the `GridView` allows you to edit and delete data, it doesn't allow you to add new data.
- **You can't deal with the Player's supported Formats**. The `GridView` allows you to handle details from the Player table and, with a little more work, also the relationship to the Manufacturer of the Player, but it doesn't allow you to handle the many-to-many relationship with the Formats that the Player may support.

The first two problems can be resolved by making some modifications to the way that you use the `GridView`, as you'll see in the next two examples. In order to add a new Player to the database, you need to use one of the `GridView`'s siblings: either a `DetailsView` or a `FormView`. We'll look at both of these Web controls later in the chapter and build a new example that allows you to add new Players to the database.

The supported Formats cause the most problems. How do you deal with the Formats that the Player supports? Recall that the Player and Format tables are involved in a many-to-many relationship. This is mapped to two one-to-many relationships via the WhatPlaysWhatFormat table, as shown in Figure 9-8.

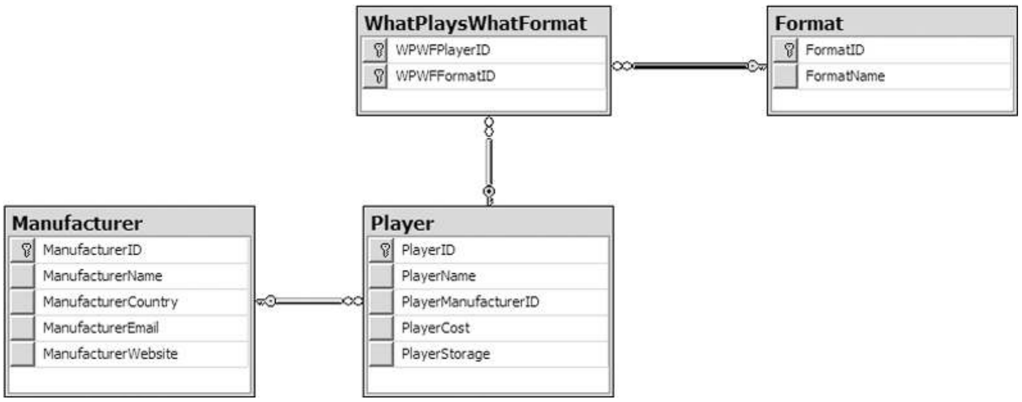


Figure 9-8. *The relationships in the database*

You can handle the Player-to-Manufacturer relationship by allowing the user to change the value of the PlayerManufacturerID value in the Player table. At the moment, the user must change the value manually, but in the next example, you'll see how to make the interface more user-friendly.

It's the Player-to-Format relationship that causes the problem. It's a many-to-many relationship, via the WhatPlaysWhatFormat intermediary table, and you can't model this automatically. You saw in Chapter 8 that you need to write some relatively complex code to model this relationship.

The `GridView` can't handle this relationship easily. When dealing with a complex database relationship such as this, you're better off using an alternative. Indeed, the examples that you saw in Chapter 8 are the best way to handle modifications to the Player table.

■ **Note** In the code download for this chapter, you'll find two pages, `Manufacturers_Details.aspx` and `Formats_Details.aspx`, that allow you to manage the Manufacturer and Format tables. Here, you used the Player table, because that demonstrates more of the `GridView`'s capabilities. The other tables contain only text.

Now that we've established that there are problems with the `GridView` and why it's not suitable in all cases, we'll look at what it actually does.

Auto-Generated SQL Queries

When adding a SqlDataSource that supports modifications to the database, you follow the normal process for adding the Web control. The Configure Data Source wizard allows you to define the SELECT query that you want to execute, and you can also test that this is returning the correct results by testing the query. However, you also need to define INSERT, UPDATE, and DELETE queries in order for the changes to be propagated back to the database.

You can define these queries manually, or you can ask the Configure Data Source wizard to generate these queries automatically by checking the Generate INSERT, UPDATE, and DELETE statements check box in the wizard, as shown earlier in Figure 9-1. If you select this option, you'll see that the wizard has not only added the SelectCommand property to the SqlDataSource, but also InsertCommand, UpdateCommand, and DeleteCommand properties as well:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  ConnectionString="<%$ ConnectionStrings:SqlConnectionString %>"
  DeleteCommand="DELETE FROM Player WHERE PlayerID = @PlayerID"
  InsertCommand="INSERT INTO Player (PlayerName, PlayerManufacturerID,
    PlayerStorage, PlayerCost) VALUES (@PlayerName,
    @PlayerManufacturerID, @PlayerStorage, @PlayerCost)"
  SelectCommand="SELECT PlayerID, PlayerName, PlayerManufacturerID,
    PlayerStorage, PlayerCost FROM Player"
  UpdateCommand="UPDATE Player SET PlayerName = @PlayerName,
    PlayerManufacturerID = @PlayerManufacturerID, PlayerStorage =
    @PlayerStorage, PlayerCost = @PlayerCost WHERE PlayerID = @PlayerID">
```

The wizard has automatically determined what the different queries should be from the information that you've supplied for the SELECT query. It also had to query the structure of the table to work out the primary key. If you look at the DeleteCommand and UpdateCommand properties, you'll see that the queries are constrained by a WHERE clause for the PlayerID column.

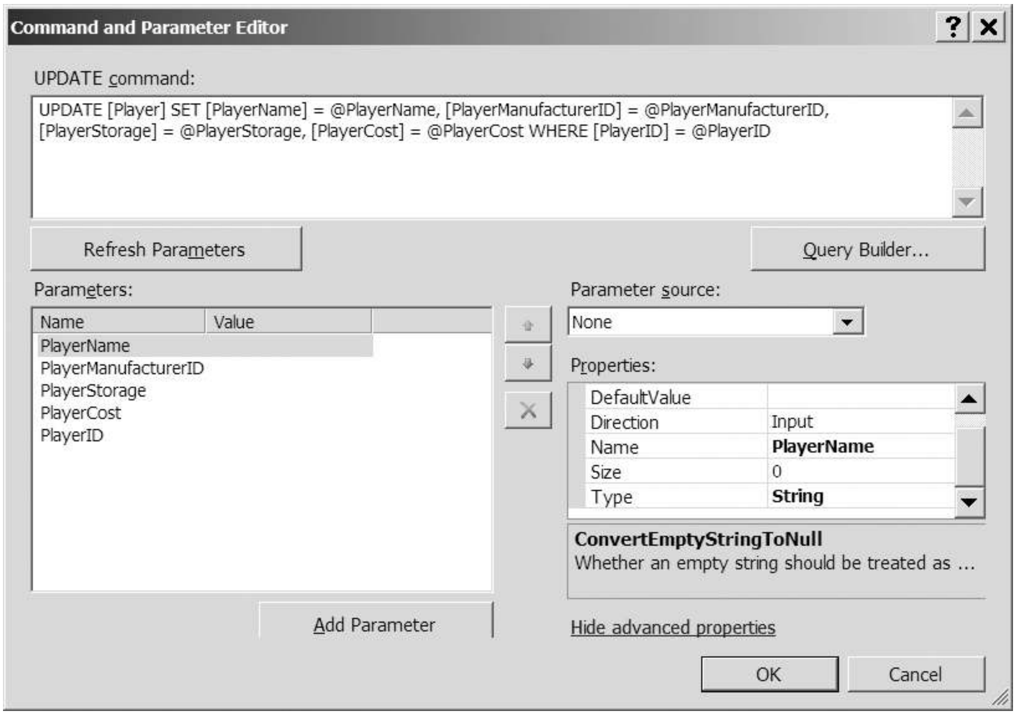
You'll see that the three auto-generated queries are parameterized, and the wizard has also queried the database for the types of these parameters and added the parameters to the SqlDataSource definition:

```
<DeleteParameters>
  <asp:Parameter Name="PlayerID" Type="Int32" />
</DeleteParameters>
<UpdateParameters>
  <asp:Parameter Name="PlayerName" Type="String" />
  <asp:Parameter Name="PlayerManufacturerID" Type="Int32" />
  <asp:Parameter Name="PlayerStorage" Type="String" />
  <asp:Parameter Name="PlayerCost" Type="Decimal" />
  <asp:Parameter Name="PlayerID" Type="Int32" />
</UpdateParameters>
<InsertParameters>
  <asp:Parameter Name="PlayerName" Type="String" />
  <asp:Parameter Name="PlayerManufacturerID" Type="Int32" />
  <asp:Parameter Name="PlayerStorage" Type="String" />
  <asp:Parameter Name="PlayerCost" Type="Decimal" />
</InsertParameters>
```

The wizard has auto-generated the three required queries automatically and added the parameters to the SqlDataSource definition. But there may be instances where you want to define your own INSERT, UPDATE, and DELETE queries. You can do this manually by modifying the relevant Command property and adding the correct parameters for the query. If you use stored procedures (which we'll look at in Chapter 10), you'll usually need to create the queries for the SqlDataSource manually and specify the parameters to be passed to the query.

Another option when specifying the different queries for the SqlDataSource is to use the Command and Parameter Editor. You've already seen this in action for building SELECT queries, but it also handles INSERT, UPDATE, and DELETE queries.

If you look at the properties for the SqlDataSource, you'll see that, as well as the SelectQuery property, you also have DeleteQuery, InsertQuery, and UpdateQuery properties. Clicking the ellipsis for any of these properties launches the Command and Parameter Editor dialog box, as shown in Figure 9-9 for the UpdateQuery property.



```
UPDATE [Player] SET [PlayerName] = @PlayerName, [PlayerManufacturerID] = @PlayerManufacturerID,
[PlayerStorage] = @PlayerStorage, [PlayerCost] = @PlayerCost WHERE [PlayerID] = @PlayerID
```

Name	Value
PlayerName	
PlayerManufacturerID	
PlayerStorage	
PlayerCost	
PlayerID	

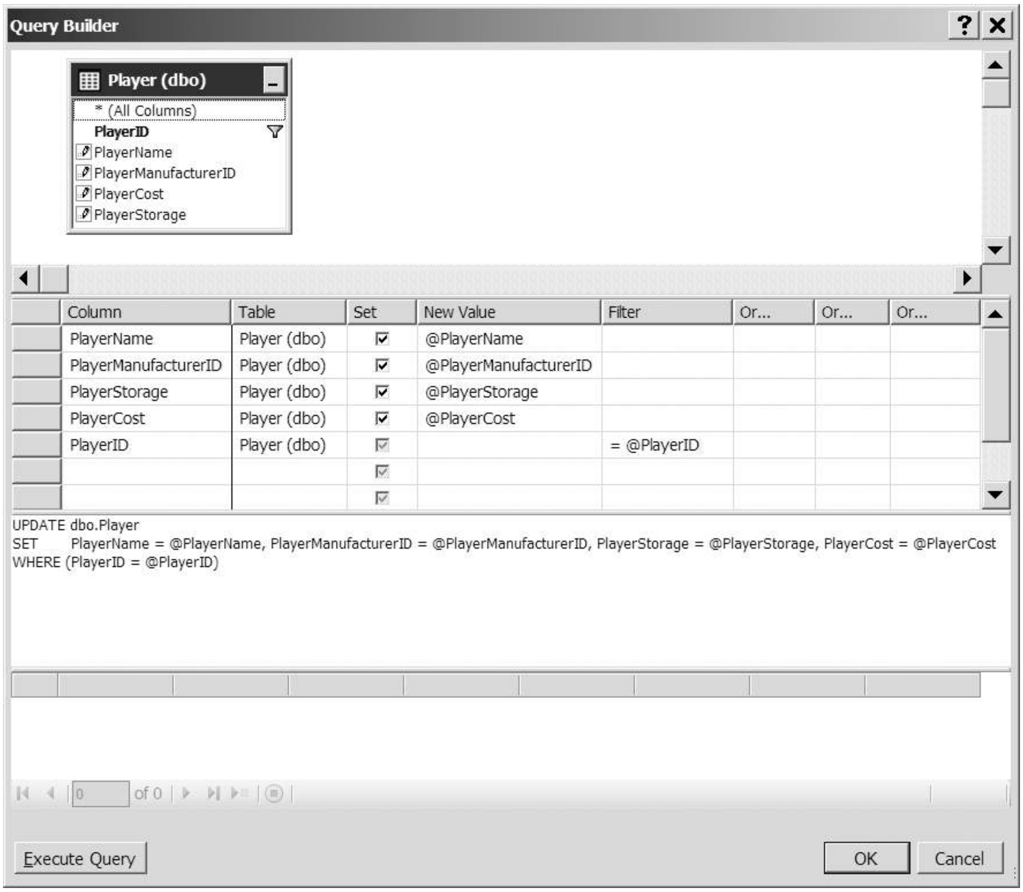
DefaultValue	
Direction	Input
Name	**PlayerName**
Size	0
Type	**String**

ConvertEmptyStringToNull
Whether an empty string should be treated as ...

Figure 9-9. *Parameters can be defined using the Command and Parameter Editor dialog box.*

The Command and Parameter Editor dialog box allows you to define the query that you want to execute, and then define the parameters for it as well. A quick shortcut is to specify the query, and then click the Refresh Parameters button to automatically extract the parameters and populate the Parameters collection. By clicking the Show Advanced Properties link, you can also further modify the parameters. The wizard has defined the types for each of the parameters, as you can see in Figure 9-9.

You can also graphically design the query by clicking the Query Builder button. This will launch the Query Builder dialog box, as shown in Figure 9-10.



Column	Table	Set	New Value	Filter	Or...	Or...	Or...
PlayerName	Player (dbo)	✓	@PlayerName				
PlayerManufacturerID	Player (dbo)	✓	@PlayerManufacturerID				
PlayerStorage	Player (dbo)	✓	@PlayerStorage				
PlayerCost	Player (dbo)	✓	@PlayerCost				
PlayerID	Player (dbo)	✓		= @PlayerID			

```
UPDATE dbo.Player
SET    PlayerName = @PlayerName, PlayerManufacturerID = @PlayerManufacturerID, PlayerStorage = @PlayerStorage, PlayerCost = @PlayerCost
WHERE (PlayerID = @PlayerID)
```

Figure 9-10. *The Query Builder can also handle DELETE, INSERT, and UPDATE queries.*

You can also test the query by clicking the Execute Query button. Be careful when testing `INSERT`, `UPDATE`, and `DELETE` queries, as you're going to be modifying the data in the database, and any changes you make are permanent.

Once you're happy with the query, you can close the Query Builder dialog box, and then use the Refresh Parameters button in the Command and Parameter Editor dialog box to build the correct `Parameters` collection.

The Editable GridView

Once the `SqlDataSource` has been configured correctly, you can use it as the data source for a `GridView`. The easiest way to do this is to select the data source from the Tasks menu for the `GridView`.

We've already looked at binding a `SqlDataSource` to a `GridView`. However, when you have an editable `SqlDataSource`, something else happens:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
  DataKeyNames="PlayerID" DataSourceID="SqlDataSource1">
```

The `DataKeyNames` property has been added and set to the primary key, PlayerID, from the `SELECT` query that is being executed. This property is used by the `GridView` (and the `DetailsView` and `FormView`) to enable the `INSERT`, `UPDATE`, and `DELETE` queries to function. If you don't set this property correctly, the `INSERT`, `UPDATE`, and `DELETE` queries will not work.

A couple of properties are applied to the PlayerID `BoundField` as well:

```
<asp:BoundField DataField="PlayerID" HeaderText="PlayerID"
  InsertVisible="False" ReadOnly="True" SortExpression="PlayerID" />
```

The two highlighted properties are related to the fact that the PlayerID column is the primary key for the table that you're modifying:

- `ReadOnly`: You can't modify the primary key column for a row of data. Setting the `ReadOnly` property to `True` prevents the `GridView` from making this column editable.
- `InsertVisible`: When entering new data, you don't normally specify the primary key. By setting the `InsertVisible` property to `False`, you hide the column when you're adding new data to the database. With the `GridView`, this property doesn't affect anything, as you can't add new data using the `GridView`. However, when we look at using the `DetailsView` later in this chapter, you'll see this property in action.

Setting the `GridView` to allow editing and deleting has also added a new column, a `CommandField`, to the `Columns` collection. This column specifies which of the automatic actions you want the `GridView` to perform. In this case, you want Edit and Delete links:

```
<asp:CommandField ShowDeleteButton="True" ShowEditButton="True" />
```

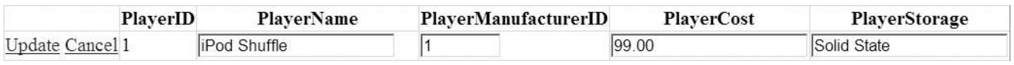
You'll notice that the two properties you've set refer to buttons, rather than links. This isn't a mistake. By default, the `GridView` displays a `LinkButton`, rather than a `Button`. So, they appear as links on the pages you're building.

The `CommandField` column has several different properties that you can set. Not only can you add a couple other buttons, but you can also specify the text that the buttons show (the `DeleteText` and `UpdateText` properties) and change the type of button (using the `ButtonType` property). You'll see some of the other properties as we progress through this chapter.

■ **Note** For a full discussion of the `CommandField`, refer to `http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.commandfield.aspx`.

Editing a Row

Clicking the Edit link for a row of data posts the page back to the server and switches the `GridView` into `Edit` mode. Figure 9-11 shows the first row in `Edit` mode.



	PlayerID	PlayerName	PlayerManufacturerID	PlayerCost	PlayerStorage
Update Cancel	1	iPod Shuffle	1	99.00	Solid State

Figure 9-11. *A row in the GridView in Edit mode*

As you'll recall, the PlayerID column is specified as a ReadOnly column, so you can't modify this column. However, the remaining four columns have been turned into text boxes that can be modified.

A BoundField, when in Edit mode, is shown as a TextBox. There is no way to change this. If you need to use a different Web control, you'll need to change the column to a TemplateField and create the necessary templates. You'll do this in the next example.

The other change when in Edit mode is that the CommandField column has changed to show Update and Cancel links. If you decide you don't want to make any changes, click Cancel. If you're happy with the changes that you've made, click Update. Either way, the page is posted back to the server. If you clicked Update, the changes are made to the database, and the GridView is switched back into View mode. Any changes you've made will be visible immediately.

So how does the Update link work? The GridView populates the UpdateParameters collection in the bound SqlDataSource, and then calls the Update() method.

Handling Errors

As you saw, if you click the Delete link, an error is thrown by the database. Showing the ASP.NET error page to the user is not advisable and, if there is an error, you need to handle this gracefully and let the user know that an error has occurred without breaking the Web site.

The GridView (and the DetailsView and FormView) make this very simple. The after action events (Inserted, Updated, and Deleted) allow you to check to see if any errors have occurred and handle those errors. You can then tell the GridView that you've handled the error so that it isn't rethrown and the ASP.NET error page shown.

In this example, you just show the error message that is generated in a Label. In a real Web site, you may want to log the error to a file or by e-mail.

The error that you're dealing with arises when the DELETE query is called, so you need to add the Deleted event handler to the GridView (the same principal applies if you're executing an INSERT or UPDATE query and using the Inserted or Updated event handlers).

The argument to the Deleted event handler, a GridViewDeletedEventArgs object, has an Exception property that contains the exception that was raised, or null if no exception was raised. If the Exception property is not null, you know an error has occurred, and you can show this to the user:

```
if (e.Exception != null)
{
  lblError.Visible = true;
  lblError.Text = e.Exception.Message;
  e.ExceptionHandled = true;
}
```

You first make sure the Label is visible, and then show the exact Message to the user. You also set the ExceptionHandled property of the GridViewDeletedEventArgs object to true to indicate that you've handled the error.

By saying that you've handled the error, you're telling the GridView not to worry about it, and the users won't see the ASP.NET error page. Instead, they will see your nice, red error message.

■**Note** You'll notice that you haven't added error handling for the `UPDATE` query to this example. Nor do you add any new error handling to any of the examples from this point forward. The process for handling errors is the same for `INSERT`, `UPDATE`, and `DELETE`, so once you've seen it, adding it again (and again and again) to the examples will just make them longer for no added benefit.

Try It Out: Changing Controls Used for Editing

In this example, you'll expand on the previous example by changing the Web control that is used for setting the Manufacturer for a Player. Rather than using a `TextBox`, you'll populate a `DropDownList` with the list of Manufacturers and let the users select which Manufacturer they want.

1. Open `Players_Basic.aspx` from the previous example.
2. Add a second `SqlDataSource` to the page (this will be given an ID of `SqlDataSource2`). Use the `SqlConnectionString` to connect to the correct database, and set the `SelectCommand` to the following:

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

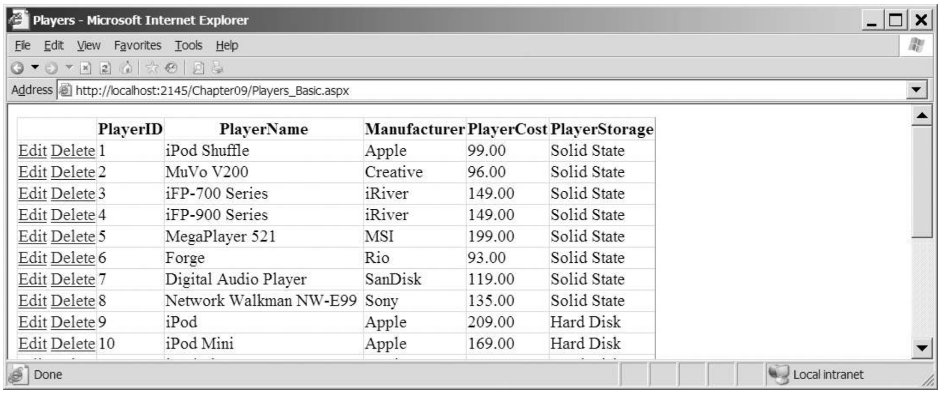
3. Switch to the Source view and change the `SelectCommand` for `SqlDataSource1` to the following:

```
SELECT PlayerID, PlayerName, ManufacturerName,
  PlayerManufacturerID, PlayerStorage, PlayerCost
FROM Player INNER JOIN Manufacturer
  ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
```

4. Replace the PlayerManufacturerID `BoundField` with the following `TemplateField`:

```
<asp:TemplateField HeaderText="Manufacturer"
  SortExpression="ManufacturerName">
  <ItemTemplate>
    <asp:Literal ID="litManufacturer" runat="server"
      Text='<%# Eval("ManufacturerName") %>'>
    </asp:Literal>
  </ItemTemplate>
  <EditItemTemplate>
    <asp:DropDownList id="lstManufacturer" runat="server"
      DataSourceID="SqlDataSource2"
      DataTextField="ManufacturerName" DataValueField="ManufacturerID"
      SelectedValue='<%# Bind("PlayerManufacturerID") %>'>
    </asp:DropDownList>
  </EditItemTemplate>
</asp:TemplateField>
```

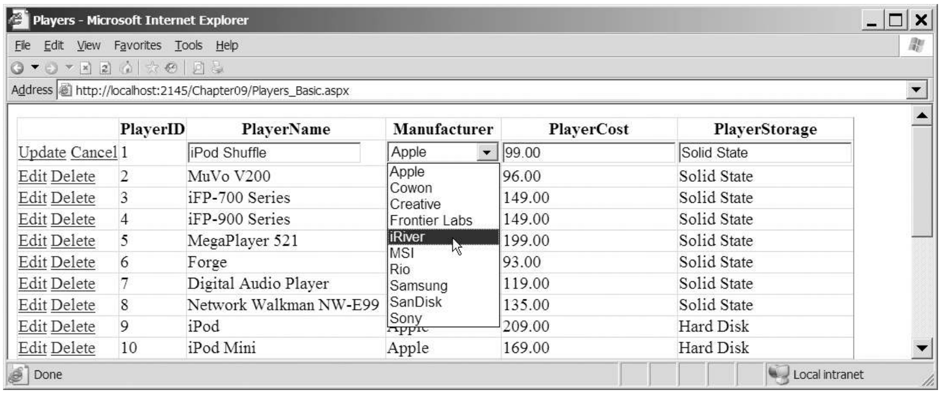
5. Save the page and view it in your browser. You'll see that the Manufacturer column now appears with the name of the Manufacturer, rather than the ID value, as shown in Figure 9-12.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Basic.aspx
Done
Local intranet

	PlayerID	PlayerName	Manufacturer	PlayerCost	PlayerStorage
Edit Delete	1	iPod Shuffle	Apple	99.00	Solid State
Edit Delete	2	MuVo V200	Creative	96.00	Solid State
Edit Delete	3	iFP-700 Series	iRiver	149.00	Solid State
Edit Delete	4	iFP-900 Series	iRiver	149.00	Solid State
Edit Delete	5	MegaPlayer 521	MSI	199.00	Solid State
Edit Delete	6	Forge	Rio	93.00	Solid State
Edit Delete	7	Digital Audio Player	SanDisk	119.00	Solid State
Edit Delete	8	Network Walkman NW-E99	Sony	135.00	Solid State
Edit Delete	9	iPod	Apple	209.00	Hard Disk
Edit Delete	10	iPod Mini	Apple	169.00	Hard Disk

Figure 9-12. *The Manufacturer is now shown in a meaningful fashion.*

6. Click the Edit option. You'll see that the Manufacturer can now be selected from a `DropDownList`, as shown in Figure 9-13.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Basic.aspx
Apple
Cowon
Creative
Frontier Labs
iRiver
MSI
Rio
Samsung
SanDisk
Sony
Done
Local intranet

	PlayerID	PlayerName	Manufacturer	PlayerCost	PlayerStorage
Update Cancel	1	iPod Shuffle	Apple	99.00	Solid State
Edit Delete	2	MuVo V200		96.00	Solid State
Edit Delete	3	iFP-700 Series		149.00	Solid State
Edit Delete	4	iFP-900 Series		149.00	Solid State
Edit Delete	5	MegaPlayer 521		199.00	Solid State
Edit Delete	6	Forge		93.00	Solid State
Edit Delete	7	Digital Audio Player		119.00	Solid State
Edit Delete	8	Network Walkman NW-E99		135.00	Solid State
Edit Delete	9	iPod	Apple	209.00	Hard Disk
Edit Delete	10	iPod Mini	Apple	169.00	Hard Disk

Figure 9-13. *You can now select the Manufacturer from a DropDownList.*

How It Works

Rather than forcing the user to remember all of the different values for the Manufacturers, you're allowing the user to select the Manufacturer from a `DropDownList`. You need to populate this `DropDownList` with a list of Manufacturers, so you use a second `SqlDataSource` to return the Manufacturers with a simple `SELECT` query:

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

You'll use this shortly to populate the DropDownList, but you also change the SelectCommand query to populate the GridView with all of the Players. Although you have the PlayerManufacturerID and can use this to set the DropDownList to the correct value, the value is not meaningful to the user. So, you modify the existing SELECT query to return the ManufacturerName, as well as the PlayerManufacturerID:

```
SELECT PlayerID, PlayerName, ManufacturerName,
  PlayerManufacturerID, PlayerStorage, PlayerCost
FROM Player INNER JOIN Manufacturer
  ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
```

You've changed only the SELECT query and not modified the INSERT, UPDATE, or DELETE queries. You're going to modify only the Player table, and the fact that you're returning the ManufacturerName is irrelevant as far as modifying the Player table is concerned. You return this only for display to the user.

In order to display a DropDownList, you need to replace the BoundField with a TemplateField:

```
<asp:TemplateField HeaderText="Manufacturer"
  SortExpression="ManufacturerName">
```

You set the HeaderText property to something meaningful and set the SortExpression to the values that you're going to show when the GridView is in View mode. Although you haven't enabled sorting or paging in this example, you'll notice that all of the BoundField columns have a SortExpression, and you add one here for completeness.

As you've seen in previous examples, within the TemplateField, you can define several different templates. The first one that you define here is the ItemTemplate displayed when the GridView is in View mode:

```
<ItemTemplate>
  <asp:Literal ID="litManufacturer" runat="server"
    Text='<%# Eval("ManufacturerName") %>'>
  </asp:Literal>
</ItemTemplate>
```

Within the ItemTemplate, you want to show the ManufacturerName, and you use a Literal to do this. You set the Text property to the ManufacturerName using the Eval() method. You don't need to edit the ManufacturerName, so you can use the Eval() method, as this allows one-way binding.

You also define an EditItemTemplate that will be displayed when the GridView is in Edit mode:

```
<EditItemTemplate>
  <asp:DropDownList id="lstManufacturer" runat="server"
    DataSourceID="SqlDataSource2"
    DataTextField="ManufacturerName" DataValueField="ManufacturerID"
    SelectedValue='<%# Bind("PlayerManufacturerID") %>'>
  </asp:DropDownList>
</EditItemTemplate>
```

We looked at data binding the DropDownList in some depth in Chapter 7. You're using SqlDataSource2 as the data source, and then specifying the DataTextField and the DataValueField correctly. The property that we're really interested in here is SelectedValue.

You're going to set the SelectedValue to the value returned as the PlayerManufacturerID. By using the Bind() method, you can perform two-way data binding. When an update occurs, the currently selected value in the DropDownList will be passed to the PlayerManufacturerID parameter.

Try It Out: Deleting Data in a GridView

As you saw earlier, the Player table is part of a relationship with the WhatPlaysWhatFormat table, and the constraints in the database prevent you from performing deletions from the Player without deleting the referencing rows from WhatPlaysWhatFormat table. Now you will update the earlier example to allow Players to be deleted from the database.

1. Open Players_Basic.aspx from the previous example.
2. Add the required Import statement to the top of the page:

```
<%@ Import Namespace="System.Data.SqlClient" %>
```

3. In the Design view, select the GridView. From the Properties window, add the RowDeleting event. Within the RowDeleting event handler, add the following code:

```
protected void GridView1_RowDeleting(object sender,
  GridViewDeleteEventArgs e)
{
  // create the connection
  string strConnectionString = ConfigurationManager.
    ConnectionStrings["SqlConnectionString"].ConnectionString;
  SqlConnection myConnection = new SqlConnection(strConnectionString);

  try
  {
    // query to execute
    string strQuery = "DELETE FROM WhatPlaysWhatFormat ➥
      WHERE WPWFPlayerID = @PlayerID;";

    // create the command
    SqlCommand myCommand = new SqlCommand(strQuery, myConnection);

    // add the parameter
    myCommand.Parameters.AddWithValue("@PlayerID", e.Keys["PlayerID"]);

    // open the connection
    myConnection.Open();
```

```
        // execute the command
        myCommand.ExecuteNonQuery();
      }
      catch (Exception ex)
      {
        lblError.Visible = true;
        lblError.Text = ex.Message;
        e.Cancel = true;
      }
      finally
      {
        // close the connection
        myConnection.Close();
      }
    }
```

4. Save the page, and then view it in your browser. If you now delete one of the Players, you'll see that the page no longer reports an error. The Player is deleted from the database.

How It Works

As you've learned, the main constraint when modifying the contents of the database is that you must work to the database rules. You can't delete from the Player table if there is related data in the WhatPlaysWhatFormat table. You must delete that data before you delete the Player itself.

When you click the Delete link, there is no more interaction from the user, and the `GridView` takes full control. It populates the `DeleteParameters` collection of the bound `SqlDataSource`, and then calls the `Delete()` method. A couple of events are raised, as well as the `DELETE` itself being performed:

- The `RowDeleting` event is raised.
- The `DELETE` query (via the `SqlDataSource`) is executed.
- The `RowDeleted` event is raised.

As you must make changes to the database before you allow the Player itself to be deleted, you need to use the `RowDeleting` event to delete the related information.

Within the `RowDeleting` event handler, you perform a simple `DELETE` from the WhatPlaysWhatFormat table:

```
DELETE FROM WhatPlaysWhatFormat
WHERE WPWFPlayerID = @PlayerID
```

You have a parameterized query that requires the PlayerID. You don't have direct access to this value, but you can retrieve it from the `GridViewDeleteEventArgs` parameter. The `Keys` collection contains all of the keys for the row in question, as specified by the `DataKeyNames` property of the `GridView`.

You can retrieve the current value for the specified key by indexing on the name of the column:

```
myCommand.Parameters.AddWithValue("@PlayerID", e.Keys["PlayerID"]);
```

You then use the `ExecuteNonQuery()` method to execute the `DELETE` query. This deletes the related data from the WhatPlaysWhatFormat table, and you can then let the `GridView` delete the data from the Player table.

■**Note** It's fine to simply delete the related data within the database when you delete a Player. The Formats that the Player supports are owned by the Player, so deleting the related data doesn't affect any data that it shouldn't. The same isn't true for the Manufacturer and Format tables: you can't delete a Manufacturer or a Format if they're in use. In the code download, you'll find that `Manufacturers_Details.aspx` and `Formats_Details.aspx` prevent any deletions from being made if the Manufacturer or Format is in use.

You'll notice that you're also catching any errors that may occur during the `DELETE` against the WhatPlaysWhatFormat table:

```
catch (Exception ex)
{
  lblError.Visible = true;
  lblError.Text = ex.Message;
  e.Cancel = true;
}
```

As with the earlier example, whenever there is an error, you display the error to the user using the `Label`. You're also setting the `Cancel` property of the `GridViewDeleteEventArgs` object to `true`. This tells the `GridView` that you don't want the `DELETE` operation to take place—as there has been an error, you don't want another error to potentially occur when you try to delete the data from the Player table.

■**Note** As I've already pointed out, we're not going to add error handling to all of the examples that you're going to build. The techniques for dealing with errors are exactly the same for the `GridView`, `DetailsView`, and `FormView`. You should be able to add error handling to the different Web controls easily, now that you know how it is accomplished.

Using the DetailsView

Although you now have an editable `GridView`, you don't have a way to add new Players to the database. You could build a new page to do this, as you saw in Chapter 8. Alternatively, you could use a `DetailsView` in conjunction with the `GridView` to allow new Players to be added. However, the `DetailsView` actually allows you to edit and delete data, as well as add it.

Try It Out: Showing Data in a DetailsView

In this first example, you'll build a new page that uses a `GridView` and a `DetailsView` to show the Players in the database. The following examples will build on this to allow the user to edit, delete, and add Players.

1. Add a new Web Form to the Web site called `Players_Details.aspx`. Make sure that the Place Code in Separate File check box is unchecked.
2. In the Source view, find the `<title>` tag within the HTML at the bottom of the page and change the page title to **Players**.
3. Switch to the Design view and add a `SqlDataSource` to the top of the page. Use the `SqlConnectionString` to connect to the database. On the Configure the Select Statement step, click the Specify a Custom SQL Statement or Stored Procedure option, and then click the Next button.
4. Enter the following SQL query:

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

5. Click Next, and then click Finish to close the Configure Data Source wizard.
6. Switch to the Source view and add the following table definition to the page after the `SqlDataSource` markup:

```
<table>
  <tr>
    <td valign="top"></td>
    <td valign="top"></td>
  </tr>
</table>
```

7. Switch back to the Design view and add a `GridView` to the page in the first cell of the table. From the Tasks menu, set the data source to `SqlDataSource1` and check the Enable Selection check box.
8. From the Tasks menu, select the Auto Format option and select a format, such as Colorful.
9. Switch to the Source view and add a `DataKeyNames` property to the `GridView`:

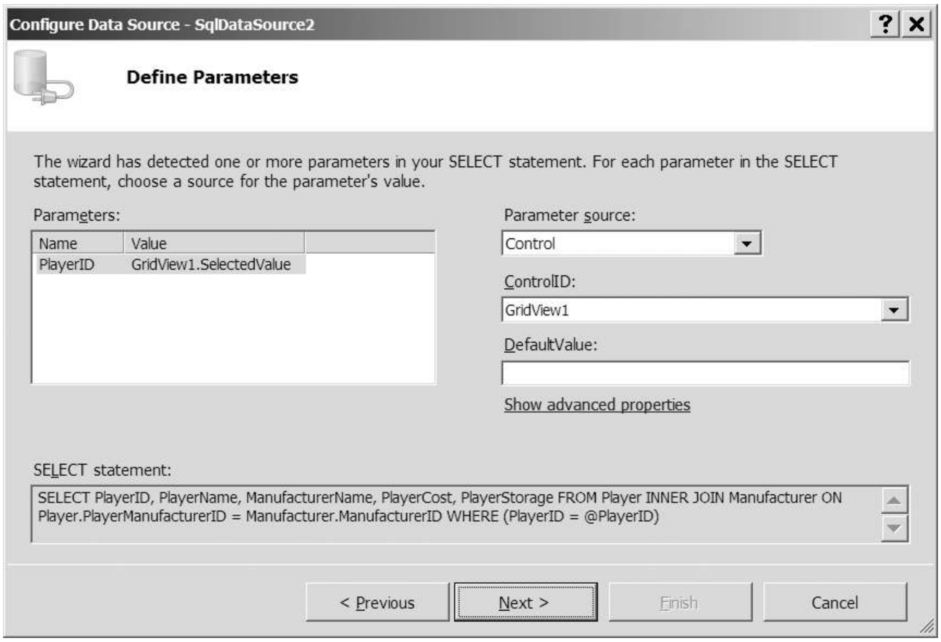
```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
  DataSourceID="SqlDataSource1" DataKeyNames="PlayerID">
```

10. Add a second `SqlDataSource` to the page (which will be called `SqlDataSource2` automatically). Again, use the `SqlConnectionString` to connect to the database. On the Configure the Select Statement step, click the Specify a Custom SQL Statement or Stored Procedure option, and then click the Next button.

```
Enter the following SQL query:

SELECT PlayerID, PlayerName, ManufacturerName, PlayerCost, PlayerStorage
FROM Player INNER JOIN Manufacturer
  ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
WHERE (PlayerID = @PlayerID)
```

11. Click Next. On the Define Parameters step, set the Parameter Source to Control and the ControlID to `GridView1`, as shown in Figure 9-14.



The wizard has detected one or more parameters in your SELECT statement. For each parameter in the SELECT statement, choose a source for the parameter's value.

Parameters:

Name	Value
PlayerID	GridView1.SelectedValue

Parameter source: Control

ControlID: GridView1

DefaultValue:

SELECT statement:

SELECT PlayerID, PlayerName, ManufacturerName, PlayerCost, PlayerStorage FROM Player INNER JOIN Manufacturer ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID WHERE (PlayerID = @PlayerID)

Figure 9-14. *Setting the parameters for the SELECT query*

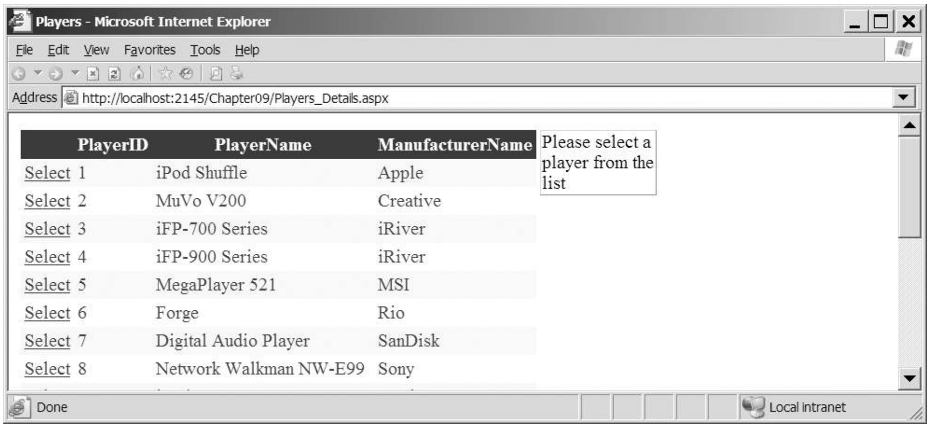
12. Click Next, and then click Finish to close the Configure Data Source wizard.

13. Add a `DetailsView` to the page in the second cell of the table. Set its data source to `SqlDataSource2`.

14. Switch to the Source view and add an `EmptyDataTemplate` to the `DetailsView`:

```
<EmptyDataTemplate>
  Please select a player from the list
</EmptyDataTemplate>
```

15. Save the page, and then view it in your browser. Initially, no Player will be selected, so the `DetailsView` will show the `EmptyDataTemplate`, as shown in Figure 9-15.

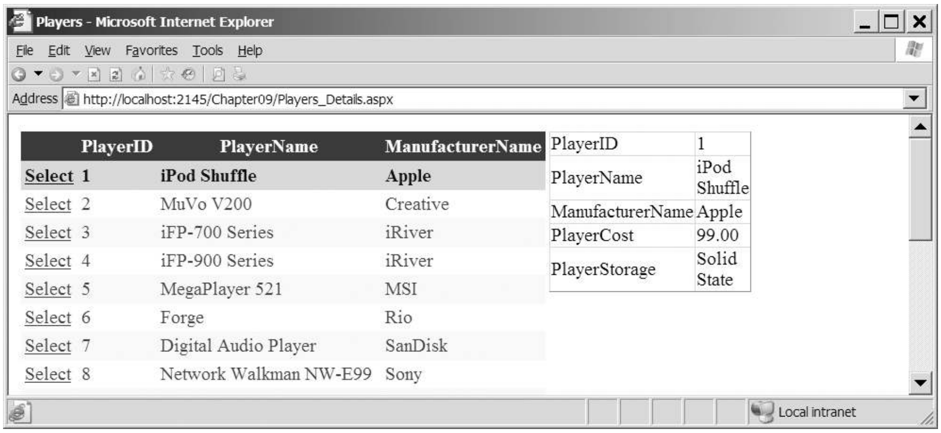

Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Details.aspx
Done
Local intranet

	PlayerID	PlayerName	ManufacturerName
Select	1	iPod Shuffle	Apple
Select	2	MuVo V200	Creative
Select	3	iFP-700 Series	iRiver
Select	4	iFP-900 Series	iRiver
Select	5	MegaPlayer 521	MSI
Select	6	Forge	Rio
Select	7	Digital Audio Player	SanDisk
Select	8	Network Walkman NW-E99	Sony

Please select a player from the list

Figure 9-15. *If no Player is selected, the EmptyDataTemplate is displayed.*

16. Click the Select link for a row, and you will see the details for the Player in the `DetailsView`, as shown in Figure 9-16.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Details.aspx
Local intranet

	PlayerID	PlayerName	ManufacturerName
Select	**1**	**iPod Shuffle**	**Apple**
Select	2	MuVo V200	Creative
Select	3	iFP-700 Series	iRiver
Select	4	iFP-900 Series	iRiver
Select	5	MegaPlayer 521	MSI
Select	6	Forge	Rio
Select	7	Digital Audio Player	SanDisk
Select	8	Network Walkman NW-E99	Sony

PlayerID	1
PlayerName	iPod Shuffle
ManufacturerName	Apple
PlayerCost	99.00
PlayerStorage	Solid State

Figure 9-16. *Viewing the details for the Player*

How It Works

In this example, you use a `GridView` to show the list of Players and then a `DetailsView` to show the details for the Player. If you haven't selected a Player from the `GridView`, the `DetailsView` shows the `EmptyDataTemplate`. When you do select a Player, the `DetailsView` shows the details for that Player.

Rather than show all of the details for the Players in the `GridView`, you show only a subset, using the following `SELECT` query:

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

This is a simple query, and you use this as the data source for the `GridView`. However, this time, you need to manually set the `DataKeyNames` property. Because it's not a query from a single table, the `GridView` is unable to work out what the primary key is for the query (is it the PlayerID from the Player table or the ManufacturerID from the Manufacturer table?), so you must manually specify that PlayerID is the primary key:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
  DataSourceID="SqlDataSource1" DataKeyNames="PlayerID">
```

The `DetailsView` also uses a `SqlDataSource` as its data source. As you're selecting only a single Player from the table, you need to use a parameterized query:

```
SELECT PlayerID, PlayerName, ManufacturerName, PlayerCost, PlayerStorage
FROM Player INNER JOIN Manufacturer
  ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
WHERE (PlayerID = @PlayerID)
```

Using the Configure Data Source wizard, you also specified the details for the parameter, and this added a parameter to the `SelectParameters` collection:

```
<SelectParameters>
  <asp:ControlParameter Name="PlayerID" ControlID="GridView1"
    PropertyName="SelectedValue" />
</SelectParameters>
```

You're using the `SelectedValue` property of the `GridView` as the value of the parameter. And this is where the Select link in the `GridView` rows is used.

By clicking the Select link, you're causing a postback to occur, and the `GridView` to select the row. As you have a `SelectedRowStyle` defined (by virtue of selecting a format from the Auto Format dialog box), you can visually see that the row in the `GridView` has been selected.

Selecting a row in the `GridView` also sets the `SelectedValue` property. As the `GridView` has the PlayerID value set for the `DataKeyNames` property, the PlayerID for the selected row is returned as the `SelectedValue`, and this is used as the parameter to the filtered `SqlDataSource`.

■**Note** By default, the `SelectedValue` of a `GridView` is null, and when the page is first loaded, this value is passed to the filtered query when the `DetailsView` is data-bound. As none of the Players have a null value as a PlayerID, no data is returned, and the `DetailsView` displays the `EmptyDataTemplate`. However, the query to the database is still executed.

If you look at the `Fields` collection for the `DetailsView`, you'll see that it is remarkably similar to the `Columns` collection for the `GridView` in the previous example. As you've learned, the Fields are interchangeable between the two Web controls:

```
<Fields>
  <asp:BoundField DataField="PlayerID"
    HeaderText="PlayerID" SortExpression="PlayerID"
    InsertVisible="False" ReadOnly="True" />
  <asp:BoundField DataField="PlayerName"
    HeaderText="PlayerName" SortExpression="PlayerName" />
  <asp:BoundField DataField="ManufacturerName"
    HeaderText="ManufacturerName" SortExpression="ManufacturerName" />
  <asp:BoundField DataField="PlayerCost"
    HeaderText="PlayerCost" SortExpression="PlayerCost" />
  <asp:BoundField DataField="PlayerStorage"
    HeaderText="PlayerStorage" SortExpression="PlayerStorage" />
</Fields>
```

Now that you've seen how easy it is to view data in a `DetailsView`, let's turn our attention to editing the details. As you're using the same Web Field controls in both the `GridView` and `DetailsView`, you've probably guessed that editing in a `DetailsView` is very similar to editing in a `GridView`.

Try It Out: Editing Data in a DetailsView

You'll expand the previous example by adding the ability to edit the selected Player.

1. Open `Players_Details.aspx` from the previous example.
2. Switch to the Design view and select Configure Data Source from the Tasks menu for `SqlDataSource2`.
3. Click Next twice to skip to the Define Custom Statements or Stored Procedures step in the wizard.
4. Modify the `SELECT` query as follows:

   ```
   SELECT PlayerID, PlayerName, ManufacturerName, PlayerManufacturerID,
     PlayerCost, PlayerStorage
   FROM Player INNER JOIN Manufacturer
     ON Player.PlayerManufacturerID = Manufacturer.ManufacturerID
   WHERE (PlayerID = @PlayerID)
   ```

5. Click the UPDATE tab and enter the following query:

   ```
   UPDATE Player SET PlayerName = @PlayerName,
       PlayerManufacturerID = @PlayerManufacturerID,
       PlayerStorage = @PlayerStorage, PlayerCost = @PlayerCost
   WHERE PlayerID = @PlayerID
   ```

6. Click the Next button, and then click the Finish button to close the Configure Data Source wizard. Click No if prompted to Refresh Fields and Keys for `DetailsView1`.

7. From the Tasks menu for the `DetailsView`, select the Enable Editing option. You'll see that there is now an Edit link at the bottom of the `DetailsView`.

8. From the Properties window, add an event handler for the `ItemUpdated` event and add the following code to the event handler:

```
protected void DetailsView1_ItemUpdated(object sender,
  DetailsViewUpdatedEventArgs e)
{
  GridView1.DataBind();
}
```

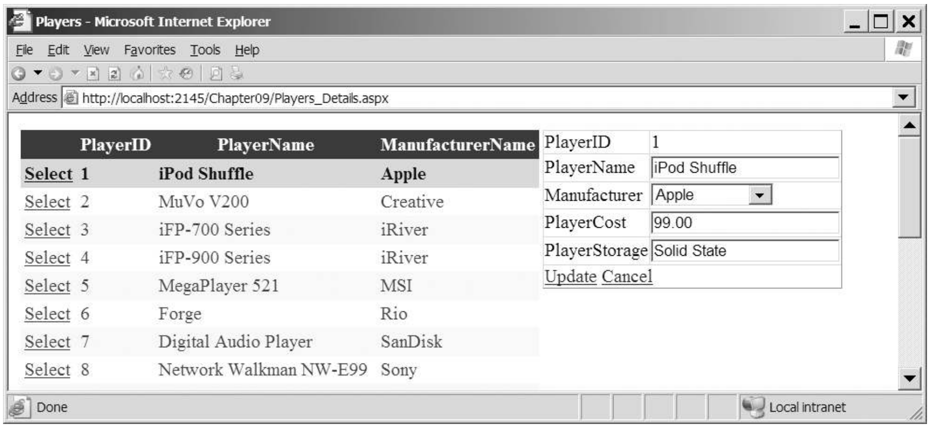
9. Add a third `SqlDataSource` to the page (which will be called `SqlDataSource3` automatically) and use `SqlConnectionString` to connect to the correct database. Use the following query to return the list of Manufacturers from the database:

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

10. Switch to the Source view and, in the `DetailsView`, replace the `BoundField` for the ManufacturerName with the following `TemplateField`:

```
<asp:TemplateField HeaderText="Manufacturer"
  SortExpression="ManufacturerName">
  <ItemTemplate>
    <asp:Literal ID="litManufacturer" runat="server"
      Text='<%# Eval("ManufacturerName") %>'>
    </asp:Literal>
  </ItemTemplate>
  <EditItemTemplate>
    <asp:DropDownList id="lstManufacturer" runat="server"
      DataSourceID="SqlDataSource3"
      DataTextField="ManufacturerName" DataValueField="ManufacturerID"
      SelectedValue='<%# Bind("PlayerManufacturerID") %>'>
    </asp:DropDownList>
  </EditItemTemplate>
</asp:TemplateField>
```

11. Save the page, and then view it in your browser. Clicking the Select link for a row will show the details for the Player in the `DetailsView`, as you saw in the previous example. Clicking the Edit link will allow you to edit the Player, as shown in Figure 9-17.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Details.aspx
Done
Local intranet

	PlayerID	PlayerName	ManufacturerName
Select	1	iPod Shuffle	Apple
Select	2	MuVo V200	Creative
Select	3	iFP-700 Series	iRiver
Select	4	iFP-900 Series	iRiver
Select	5	MegaPlayer 521	MSI
Select	6	Forge	Rio
Select	7	Digital Audio Player	SanDisk
Select	8	Network Walkman NW-E99	Sony

PlayerID	1
PlayerName	iPod Shuffle
Manufacturer	Apple
PlayerCost	99.00
PlayerStorage	Solid State
Update Cancel	

Figure 9-17. *Editing the details for the Player*

12. Change the details and click the Update link. You'll see that the changes are made to the database, and the `GridView` is updated to show the changes.

How It Works

All the changes you've made here should be familiar to you by now. The `SELECT` query that you're executing to populate the `DetailsView` is the same query that you used in the `GridView` example previously. Also, the `UPDATE` query is the same (and it probably won't come as a shock that the next example uses the same `DELETE` query).

The one thing that is slightly different is the `UpdateParameters` collection. Although the collection has been created with the correct parameters, none of them have a `Type` specified:

```
<UpdateParameters>
  <asp:Parameter Name="PlayerName" />
  <asp:Parameter Name="PlayerManufacturerID" />
  <asp:Parameter Name="PlayerStorage" />
  <asp:Parameter Name="PlayerCost" />
  <asp:Parameter Name="PlayerID" />
</UpdateParameters>
```

Because you've manually specified the `UPDATE` query that you want to execute, rather than letting the wizard auto-generate the query, the wizard has no idea of the types of the columns. It has parsed the query correctly and identified the parameter names, but not their types. Thankfully, you don't actually need the types, but if you want to add a `Type` property for each of the parameters, you can.

You'll also notice that the `TemplateField` definition is the same one that you used earlier to show the Manufacturer details in a more user-friendly manner. The `SqlDataSource` that the `DropDownList` binds to should also be very familiar, since it's the same query you used earlier to return a list of Manufacturers.

That's not to say that you've seen everything here before. There are a couple of new things to consider!

The first is the configuration of the `DetailsView`. Once you've added an `UPDATE` query to the `SqlDataSource`, the `DetailsView` acquires a new option on its Tasks menu: Enable Editing. Selecting this option adds a `CommandField` to the `Fields` collection:

```
<asp:CommandField ShowEditButton="True" />
```

Not surprisingly, this has the same effect in the `DetailsView` as it does in the `GridView`: it switches the `DetailsView` into `Edit` mode. You can then modify the data as you require and click either the Update or Cancel link to return to view mode, or as the `DetailsView` likes to think of it, `ReadOnly` mode.

Clicking the Update link saves the changes to the database, and you then need to ensure that the `GridView` is showing the updated data. In order to do this, you use the `ItemUpdated` event of the `DetailsView`:

```
protected void DetailsView1_ItemUpdated(object sender,
  DetailsViewUpdatedEventArgs e)
{
  GridView1.DataBind();
}
```

This will ensure that the `GridView` binds to the updated data in the database. If you don't call the `DataBind()` method, the `GridView` will still show the old data. The `GridView` will request data from the database only on the initial page load, and it will still show the old data unless you tell it that it needs to retrieve new data. Manually calling the `DataBind()` method forces the `GridView` to retrieve new data from the database.

Try It Out: Deleting Data in a DetailsView

Now that you know how to edit data, let's look at how to delete data.

1. Open `Players_Details.aspx` from the previous example.
2. Add the required `Import` statement to the top of the page:

   ```
   <%@ Import Namespace="System.Data.SqlClient" %>
   ```

3. Switch to the Design view and select Configure Data Source from the Tasks menu for `SqlDataSource2`.
4. Click Next twice to skip to the Define Custom Statements or Stored Procedures step in the wizard.
5. Select the DELETE tab and enter the following query:

   ```
   DELETE FROM Player
   WHERE PlayerID = @PlayerID
   ```

6. Click the Next button, and then click the Finish button to close the Configure Data Source wizard.
7. From the Tasks menu for the `DetailsView`, select the Enable Deleting option. You'll see that there is now a Delete link at the bottom of the `DetailsView`.

8. In the Properties window, add an ItemDeleting event for the DetailsView. Add the following code to the event handler:

```
protected void DetailsView1_ItemDeleting(object sender,
  DetailsViewDeleteEventArgs e)
{
  // create the connection
  string strConnectionString = ConfigurationManager.
    ConnectionStrings["SqlConnectionString"].ConnectionString;
  SqlConnection myConnection = new SqlConnection(strConnectionString);

  try
  {
    // query to execute
    string strQuery = "DELETE FROM WhatPlaysWhatFormat ➥
      WHERE WPWFPlayerID = @PlayerID;";

    // create the command
    SqlCommand myCommand = new SqlCommand(strQuery, myConnection);

    // add the parameter
    myCommand.Parameters.AddWithValue("@PlayerID", e.Keys["PlayerID"]);

    // open the connection
    myConnection.Open();

    // execute the command
    myCommand.ExecuteNonQuery();
  }
  finally
  {
    // close the connection
    myConnection.Close();
  }
}
```

9. Add an ItemDeleted event handler for the DetailsView and add the following code to the event handler:

```
protected void DetailsView1_ItemDeleted(object sender,
  DetailsViewDeletedEventArgs e)
{
  GridView1.SelectedIndex = -1;
  GridView1.DataBind();
}
```

10. Save the page, and then view it in your browser. Clicking the Select link for a row will show the details for the Player in the `DetailsView`, as you saw in the previous example. Clicking the Delete link will delete the Player from the database and update the list of Players displayed by the `GridView`.

How It Works

You've added a new button to the `CommandField`, and you're now displaying a Delete link as well as an Edit link:

```
<asp:CommandField ShowEditButton="True" ShowDeleteButton="True" />
```

As you're still working to the database rules, you need to make sure that you're deleting any data in the WhatPlaysWhatFormat table before you delete the data from the Player table. The `GridView` has a `RowDeleting` event, and the `DetailsView` has a corresponding `ItemDeleting` event. In this event, you use the same code as you saw in the `RowDeleting` event earlier to delete any related data that is in the WhatPlaysWhatFormat table.

When you click the Delete link and delete the data from the database, you also need to make sure that the `GridView` is binding to the latest version of the data in the database, and you use the `ItemDeleted` event of the `DetailsView` to call the `DataBind()` method of the `GridView`. You also set the `SelectedIndex` of the `GridView` to `-1`, so that your previous selection is removed. If it isn't, the "next" Player in the `GridView` will be selected, as it now has the row index of the deleted Player.

Try It Out: Adding Data in a DetailsView

Now that you can edit and delete data from the database, it's time to look at the final piece in the puzzle: adding a new Player to the database.

1. Open `Players_Details.aspx` from the previous example.
2. Switch to the Design view and select Configure Data Source from the Tasks menu for `SqlDataSource2`.
3. Click Next twice to skip to the Define Custom Statements or Stored Procedures step in the wizard.
4. Select the INSERT tab and enter the following query:

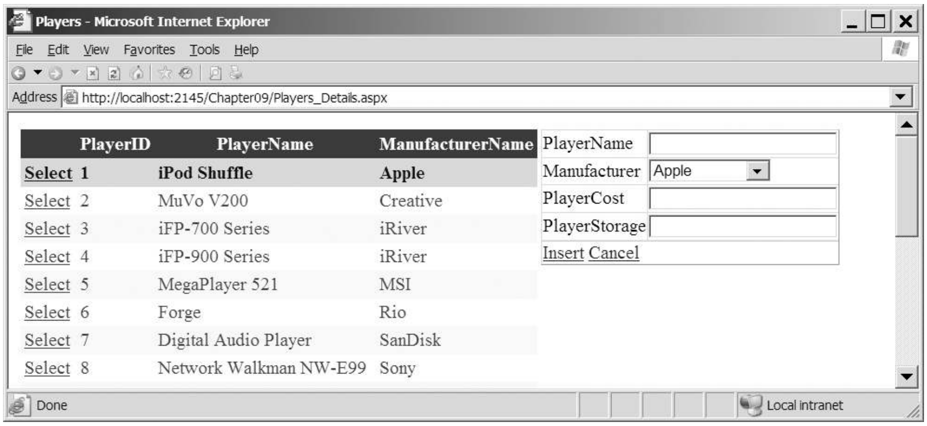
   ```
   INSERT INTO Player
     (PlayerName, PlayerManufacturerID, PlayerCost, PlayerStorage)
   VALUES (@PlayerName, @PlayerManufacturerID, @PlayerCost, @PlayerStorage)
   ```

5. Click the Next button, and then click the Finish button to close the Configure Data Source wizard. Click No if prompted to Refresh Fields and Keys for `DetailsView1`.
6. From the Tasks menu for the `DetailsView`, select the Enable Inserting option. You'll see that there is now a New link at the bottom of the `DetailsView`.

7. In the Properties window, add an `ItemInserted` event handler for the `DetailsView` and add the following code to the event handler:

```
protected void DetailsView1_ItemInserted(object sender,
  DetailsViewInsertedEventArgs e)
{
  GridView1.SelectedIndex = -1;
  GridView1.DataBind();
}
```

8. Save the page, and then view it in your browser. Clicking the Select link for a row will show the details for the Player in the `DetailsView` as you saw in the previous example. If you then click the New link, you'll be able to add a new Player to the database, as shown in Figure 9-18.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Details.aspx
Done
Local intranet

	PlayerID	PlayerName	ManufacturerName
Select	**1**	**iPod Shuffle**	**Apple**
Select	2	MuVo V200	Creative
Select	3	iFP-700 Series	iRiver
Select	4	iFP-900 Series	iRiver
Select	5	MegaPlayer 521	MSI
Select	6	Forge	Rio
Select	7	Digital Audio Player	SanDisk
Select	8	Network Walkman NW-E99	Sony

PlayerName	
Manufacturer	Apple
PlayerCost	
PlayerStorage	
Insert Cancel	

Figure 9-18. *Adding a new Player to the database*

9. Click Insert to add a new Player to the database and update the list of Players shown in the `GridView`.

How It Works

In this example, you've added the ability to add a new Player to the database with very little extra effort. You've added an `INSERT` query to the `SqlDataSource` and checked an extra option for the `DetailsView`, much as you did for the `UPDATE` and `DELETE` queries in the two previous examples.

When using a `BoundField` to interact with the database, the same Web control is used for the `SELECT`, `UPDATE`, and `INSERT` actions. For the `SELECT` query, you'll recall that a column is returned from the database that matches the `DataField` property of the `BoundField`. For the `UPDATE` and `INSERT` queries, the `UpdateParameters` and `InsertParameters` collections have a parameter that has the same name. You saw this earlier with the `UpdateParameters` collection, and here you have a similar `InsertParameters` collection:

```
<InsertParameters>
  <asp:Parameter Name="PlayerName" />
  <asp:Parameter Name="PlayerManufacturerID" />
  <asp:Parameter Name="PlayerCost" />
  <asp:Parameter Name="PlayerStorage" />
</InsertParameters>
```

You'll notice that the `InsertParameters` collection is missing the PlayerID parameter. This shouldn't come as a great surprise, as it's the auto-generated primary key and isn't present in the `INSERT` query that you're executing. When adding a new Player to the database, the PlayerID `BoundField` isn't shown because its `InsertVisible` property is set to `False`:

```
<asp:BoundField DataField="PlayerID" HeaderText="PlayerID"
  InsertVisible="False" ReadOnly="True" SortExpression="PlayerID" />
```

The other thing to notice is that you're not using the `InsertItemTemplate` within the `TemplateField` row. If you recall from the earlier discussion of templates for editing, the `InsertItemTemplate` is available when you're adding new data using a `DetailsView`. When an `InsertItemTemplate` isn't specified, the `DetailsView` will use the `EditItemTemplate` when adding new data.

All you're doing in the `TemplateField` is allowing the user to select the Manufacturer of the Player, and the process is the same for both editing and adding a Player. If you don't need to do anything different, then adding an `InsertItemTemplate` is just another place for errors to creep into your pages. Stick to the one template and use only an `EditItemTemplate`.

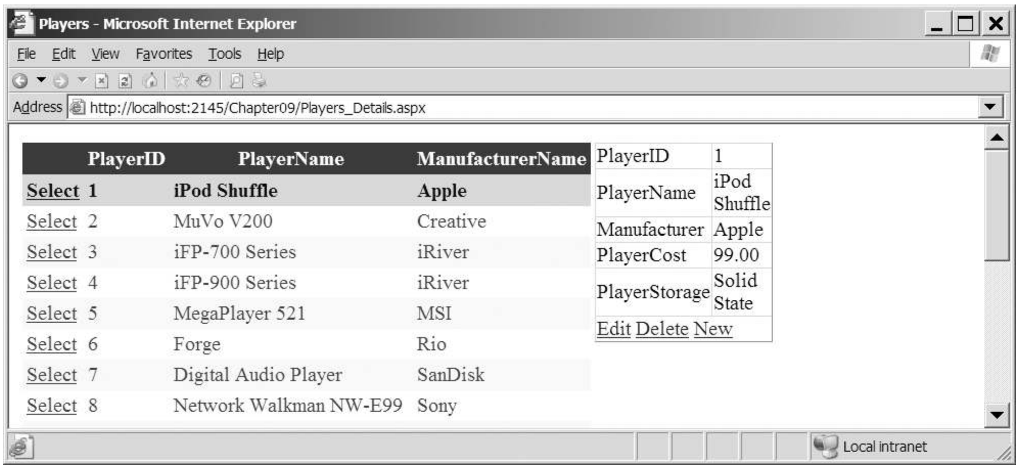
■ **Note** Although the `EditItemTemplate` is used when inserting if an `InsertItemTemplate` hasn't been defined, the same isn't true if you have an `InsertItemTemplate` and no `EditItemTemplate`. If you don't specify an `EditItemTemplate`, when you switch the Web control into `Edit` mode, the `TemplateField` will still appear in `ReadOnly` mode. If no `EditItemTemplate` is defined, the `ItemTemplate` is used in its place.

The one thing that you can't do in this example is specify the Formats that the Player supports. In reality, adding a new Player to the database is a more complex task than shown here. You saw a much better way of handling the addition of Players in Chapter 8. Although the `DetailsView` allows you to build quite complex pages very simply, this is one example of its limitations.

Tidying Up the User Interface

If you've been working through the examples so far, you may have noticed a problem related to adding a new Player to the database. If you haven't spotted the problem, open `Players_Details.aspx` in your browser. The page you'll see is shown earlier in Figure 9-15.

Still can't see the problem? How exactly do you add a new Player to the database? There's no Add button anywhere. It's only when you actually view an existing Player that you're presented with the ability to add a new Player, as shown in Figure 9-19.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Details.aspx
Local intranet

	PlayerID	PlayerName	ManufacturerName
Select	**1**	**iPod Shuffle**	**Apple**
Select	2	MuVo V200	Creative
Select	3	iFP-700 Series	iRiver
Select	4	iFP-900 Series	iRiver
Select	5	MegaPlayer 521	MSI
Select	6	Forge	Rio
Select	7	Digital Audio Player	SanDisk
Select	8	Network Walkman NW-E99	Sony

PlayerID	1
PlayerName	iPod Shuffle
Manufacturer	Apple
PlayerCost	99.00
PlayerStorage	Solid State
Edit Delete New	

Figure 9-19. *Do you really want to have to view an existing Player to add a new one?*

Although the `GridView` and `DetailsView` interact, it's hardly an ideal setup. What you really want is an Add New Player link that allows you to add a new Player to the database. It's fine for the Edit and Delete links to be shown when you've selected a Player in the `GridView`, but you don't want the New link to appear.

You can accomplish this with a few little changes to the way that the Web controls interact.

Try It Out: Manually Adding an Add New Player Link

In this example, you'll modify the previous example to make it possible for the user to add a new Player to the database without having to select an existing Player first.

1. Open `Players_Details.aspx` from the previous example.

2. Select the `DetailsView` and from its Tasks menu, unselect the Enable Inserting option.

3. Add a new `LinkButton` (above the table containing the `GridView` and `DetailsView`) and change its `Text` property to **Add New Player**.

4. Double-click the `LinkButton` to add the `Click` event handler and add the following code:

```
protected void LinkButton1_Click(object sender, EventArgs e)
{
  DetailsView1.ChangeMode(DetailsViewMode.Insert);
}
```

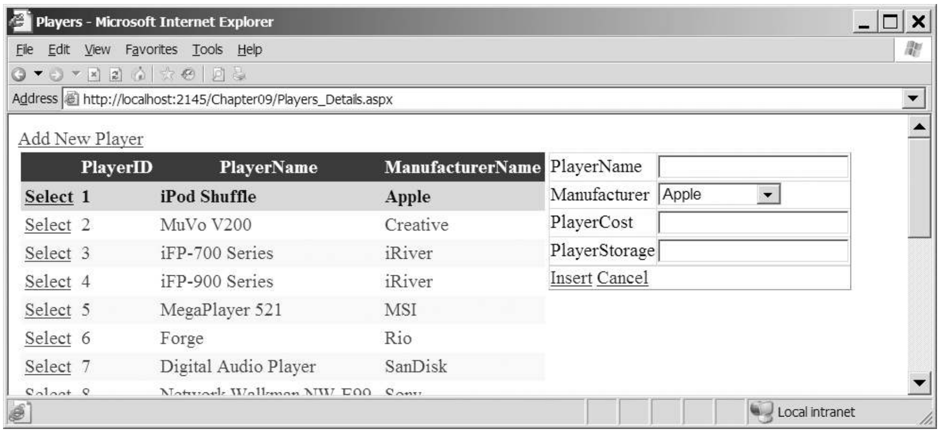
5. Switch back to the Design view and add the `SelectedIndexChanged` event for the `GridView`. Add the following code to the event handler:

```
protected void GridView1_SelectedIndexChanged(object sender, EventArgs e)
{
  DetailsView1.ChangeMode(DetailsViewMode.ReadOnly);
}
```

6. Switch back to the Design view and add the `PreRender` event for the `DetailsView`. Add the following code to the event handler:

```
protected void DetailsView1_PreRender(object sender, EventArgs e)
{
  if (DetailsView1.CurrentMode == DetailsViewMode.Insert)
  {
    DetailsView1.AutoGenerateInsertButton = true;
  }
  else
  {
    DetailsView1.AutoGenerateInsertButton = false;
  }
}
```

7. Save the page, and then view it in the browser. You'll see that clicking the Add New Player link allows you to add a new Player to the database without viewing an existing Player first, as shown in Figure 9-20.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Local intranet

http://localhost:2145/Chapter09/Players_Details.aspx

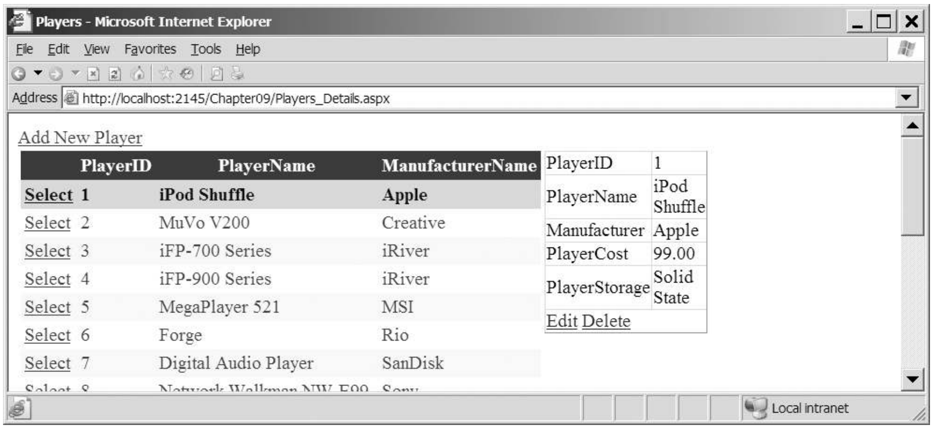
Add New Player

	PlayerID	PlayerName	ManufacturerName
Select	1	iPod Shuffle	Apple
Select	2	MuVo V200	Creative
Select	3	iFP-700 Series	iRiver
Select	4	iFP-900 Series	iRiver
Select	5	MegaPlayer 521	MSI
Select	6	Forge	Rio
Select	7	Digital Audio Player	SanDisk

PlayerName	
Manufacturer	Apple
PlayerCost	
PlayerStorage	
Insert Cancel	

Figure 9-20. *Adding a new Player without viewing an existing Player*

8. Choose to view an existing Player. As shown in Figure 9-21, you'll see that when you do this, you no longer have the New link.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address
Local intranet

http://localhost:2145/Chapter09/Players_Details.aspx

Add New Player

	PlayerID	PlayerName	ManufacturerName
Select	1	iPod Shuffle	Apple
Select	2	MuVo V200	Creative
Select	3	iFP-700 Series	iRiver
Select	4	iFP-900 Series	iRiver
Select	5	MegaPlayer 521	MSI
Select	6	Forge	Rio
Select	7	Digital Audio Player	SanDisk

PlayerID	1
PlayerName	iPod Shuffle
Manufacturer	Apple
PlayerCost	99.00
PlayerStorage	Solid State
Edit Delete	

Figure 9-21. *Viewing an existing Player no longer shows the New link.*

How It Works

In order to tidy up the user interface when using the `GridView` and `DetailsView`, you've resorted to a little bit of trickery. You no longer let the `DetailsView` decide when to show a link to add a new Player. Instead, you add your own link.

In order to do this, you need to manually tell the `DetailsView` that it is inserting a new Player. In previous examples, the `DetailsView` switched itself into `Insert` mode when you clicked the New link. As you don't have this link here, you need to do this manually:

```
protected void LinkButton1_Click(object sender, EventArgs e)
{
  DetailsView1.ChangeMode(DetailsViewMode.Insert);
}
```

So when the user clicks the Add New Player link, the `Click` event handler is executed, and the `ChangeMode` method of the `DetailsView` is used to switch into `Insert` mode.

Once you switch into `Insert` mode, the `DetailsView` will remain in that mode until it's told to be in a different mode. If you clicked a Select link, the `DetailsView` would still be in `Insert` mode, not in `ReadOnly` mode, as you would expect. So, when a Select link is clicked, you need to switch the `DetailsView` back into `ReadOnly` mode.

When a row is selected in the `GridView`, not only does the `SelectedValue` property get set correctly, but also the `SelectedIndexChanged` event is fired. You can use this event handler to switch back to `ReadOnly` mode:

```
protected void GridView1_SelectedIndexChanged(object sender, EventArgs e)
{
  DetailsView1.ChangeMode(DetailsViewMode.ReadOnly);
}
```

You can change the mode of the Web control in the event handlers, as the `GridView` and `DetailsView` will be automatically data-bound after the `OnPreRender` event has fired. User-requested events are fired before you get to this stage, so you're free to change the status of the

DetailsView without creating an extra load on the database due to any data binding occurring before you change the mode.

All this talk of the page life cycle leads us nicely into the last change you made to the page. You actually added an event handler for the OnPreRender event to the DetailsView:

```
protected void DetailsView1_PreRender(object sender, EventArgs e)
{
  if (DetailsView1.CurrentMode == DetailsViewMode.Insert)
  {
    DetailsView1.AutoGenerateInsertButton = true;
  }
  else
  {
    DetailsView1.AutoGenerateInsertButton = false;
  }
}
```

The first change that you made to the DetailsView was to turn off the automatic creation of the New link. Indeed, if you look at the definition for the CommandField, you'll see that you no longer have a ShowInsertButton property. However, you still need to add the New link when you're in Insert mode. You do this as late as you can—immediately before the Web control is rendered—so that the mode has been set correctly by either of the event handlers.

The AutoGenerateInsertButton property, when set to true, automatically adds a CommandField with a New link to the DetailsView. If you're not in Insert mode (you're in Edit or ReadOnly mode), then you don't want to show the New link, so you set the AutoGenerateInsertButton to false. But you never see a New link—or do you?

In the previous examples, you've seen that the Edit link automatically turns into Update and Cancel links when the DetailsView is in Edit mode. The same is also true for the New link. It turns into Insert and Cancel links when the DetailsView is in Insert mode.

As you've switched the DetailsView into Insert mode and told it to show the New link, you get the Insert and Cancel links. Being in Insert mode also hides the Edit and Delete links, as they're only shown when the Web control is in Edit or ReadOnly mode.

Using the FormView

As you've just seen, the DetailsView is quite powerful and allows you to build a page that allows editing, deleting, and adding data quite easily with a minimal amount of code.

However, the DetailsView does have one major problem: it is tabular. You're limited to adding Field Web controls (BoundField, CommandField, and so on), and they appear as a table of data. Granted, you can use a TemplateField to achieve quite a lot of customization, but in some cases, you may not want to be restricted to a tabular layout. Enter the FormView.

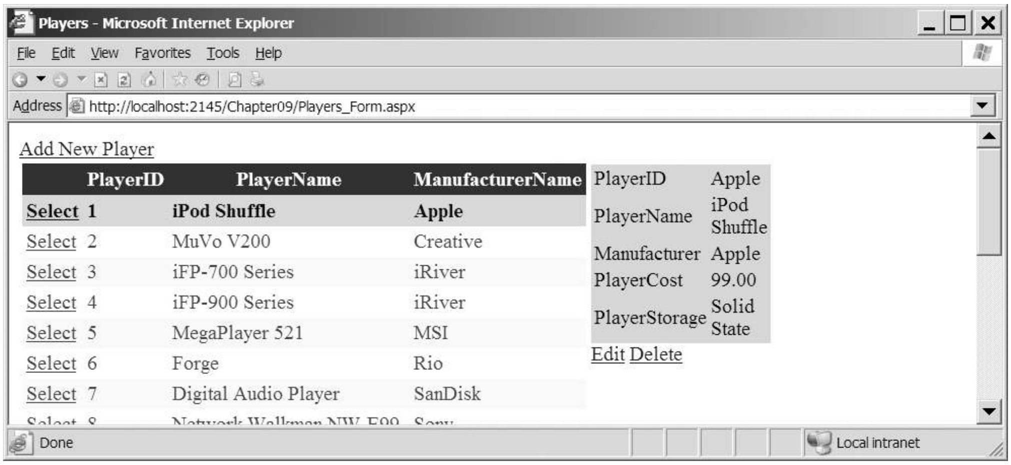
The FormView, like the DetailsView, allows you to edit, delete, and add new data. Unlike the DetailsView, however, the FormView allows you complete control over its layout. This comes with a downside though, as you need to do a little more work when you're setting up the Web control. But after you've completed the setup, you'll see that the FormView is remarkably similar to the DetailsView in the way that it works. Indeed, the SqlDataSource that you use as the data source for the DetailsView can be used with the FormView with no changes whatsoever.

In the code download for this chapter, you'll find another page, `Players_Form.aspx`, that uses a `FormView` to perform the same functions as you saw in the `DetailsView` examples.

Using Templates with the FormView

The `FormView` is completely controlled by templates. Think of it as being similar to one big `TemplateField`. You define an `ItemTemplate` that you want to use when the Web control is in `ReadOnly` mode, an `EditItemTemplate` when in `Edit` mode, and an `InsertItemTemplate` when in `Insert` mode.

Within each of the templates, you must then define what you want to be displayed on the screen. If you look at `Players_Form.aspx`, you'll see that all three templates have been defined. For the `ItemTemplate`, I've chosen to use a table with a light blue background, which distinguishes it from the Player list, as shown in Figure 9-22.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Form.aspx
Done
Local intranet

Add New Player

	PlayerID	PlayerName	ManufacturerName
Select	**1**	**iPod Shuffle**	**Apple**
Select	2	MuVo V200	Creative
Select	3	iFP-700 Series	iRiver
Select	4	iFP-900 Series	iRiver
Select	5	MegaPlayer 521	MSI
Select	6	Forge	Rio
Select	7	Digital Audio Player	SanDisk

PlayerID	Apple
PlayerName	iPod Shuffle
Manufacturer	Apple
PlayerCost	99.00
PlayerStorage	Solid State

Edit Delete

Figure 9-22. *Using a FormView to view an existing Player*

The `ItemTemplate` here is defined as follows:

```
<ItemTemplate>
  <table bgcolor="LightBlue">
    <tr>
      <td>PlayerID</td>
      <td><asp:Literal ID="litPlayerID" runat="server"
        Text='<%# Eval("ManufacturerName") %>'></asp:Literal></td>
    </tr>
    <tr>
      <td>PlayerName</td>
      <td><asp:Literal ID="litPlayerName" runat="server"
        Text='<%# Eval("PlayerName") %>'></asp:Literal></td>
    </tr>
    <tr>
      <td>Manufacturer</td>
      <td><asp:Literal ID="litManufacturer" runat="server"
        Text='<%# Eval("ManufacturerName") %>'></asp:Literal></td>
    </tr>
    <tr>
      <td>PlayerCost</td>
      <td><asp:Literal ID="litPlayerCost" runat="server"
        Text='<%# Eval("PlayerCost") %>'></asp:Literal></td>
    </tr>
    <tr>
      <td>PlayerStorage</td>
      <td><asp:Literal ID="litPlayerStorage" runat="server"
        Text='<%# Eval("PlayerStorage") %>'></asp:Literal></td>
    </tr>
  </table>
  <asp:LinkButton ID="btnEdit" runat="server"
    CommandName="Edit" Text="Edit" />
  <asp:LinkButton ID="btnDelete" runat="server"
    CommandName="Delete" Text="Delete" />
</ItemTemplate>
```

As you can see, you no longer have available the niceties of the BoundField to automatically show the data retrieved from the SqlDataSource. Instead, you need to manually output the columns you require. The ItemTemplate shows data only in ReadOnly mode, so you can use the Eval() method to do this.

The one thing to notice is that you also must manually add whatever buttons you need. The FormView is template-based and has no way to automatically generate the buttons to add the Edit and Delete links (nor any other links that are required). You therefore need to add a Button, in this case a LinkButton, which takes the CommandName of the action you require. The pertinent actions are shown in Table 9-1.

Table 9-1. *The FormView Command Buttons*

CommandName	Action
Cancel	Causes the Insert or Update operation to be canceled and returns the FormView to ReadOnly mode.
Delete	Causes the current item to be deleted from the underlying database. Raises the ItemDeleting and ItemDeleted events.
Edit	Switches the FormView into Edit mode, displaying the EditItemTemplate.
Insert	Causes the new item to be added from the underlying data source. Raises the ItemInserting and ItemInserted events.
New	Switches the FormView into Insert mode, displaying the InsertItemTemplate.
Update	Causes the current item to be updated in the underlying data source. Raises the ItemUpdating and ItemUpdated events.

As you can see from Table 9-1, it is possible to add buttons to perform whatever action you require. For the ItemTemplate, you want the user to be able to edit or delete the selected item, so you add buttons for those two actions.

You'll also see that this example follows the same pattern for the InsertItemTemplate and the EditItemTemplate. You must define the layout that you require for the template, and then provide the necessary buttons to enable the user to complete the action.

The InsertItemTemplate has Insert and Cancel buttons:

```
<asp:LinkButton ID="btnInsert" runat="server"
  CommandName="Insert" Text="Insert" />
<asp:LinkButton ID="btnCancel" runat="server"
  CommandName="Cancel" Text="Cancel" />
```

And the EditItemTemplate has Update and Cancel buttons:

```
<asp:LinkButton ID="btnUpdate" runat="server"
  CommandName="Update" Text="Update" />
<asp:LinkButton ID="btnCancel" runat="server"
  CommandName="Cancel" Text="Cancel" />
```

Switching Modes

You saw in the DetailsView example earlier that you can manually change the mode for the Web control by calling the ChangeMode() method. This also works for the FormView.

You have a LinkButton on the page that allows the user to add a new Player to the database. In its Click event handler, you change the mode to Insert:

```
protected void LinkButton1_Click(object sender, EventArgs e)
{
  FormView1.ChangeMode(FormViewMode.Insert);
}
```

And you switch the FormView back into ReadOnly mode if the user selects a different Player:

```
protected void GridView1_SelectedIndexChanged(object sender, EventArgs e)
{
  FormView1.ChangeMode(FormViewMode.ReadOnly);
}
```

This is exactly the same as you saw for the DetailsView. What you don't need to do when using the FormView is turn the New link on and off. Since you have no way of automatically adding buttons to a FormView, you explicitly specify the actions that you allow the user to take within the definition of the InsertItemTemplate and the EditItemTemplate.

Validating User Responses

The one thing missing from all of the examples that you've seen so far is validation. Nothing stops you, at the moment, from entering invalid data. You saw in Chapter 8 that you can use various validation Web controls to force the user to enter valid data before any changes to the database are attempted.

You can use the same validation Web controls when you're using a FormView without any work other than adding the Web controls to the page and configuring them. With the GridView and the DetailsView, however, things are a little more complex, because the validation Web controls won't work in conjunction with a BoundField or a CheckBoxField. The solution is to convert the BoundField or CheckBoxField into an equivalent TemplateField.

The DetailsView example includes several BoundField controls. One of these is used to display the PlayerName column:

```
<asp:BoundField DataField="PlayerName"
  HeaderText="PlayerName" SortExpression="PlayerName" />
```

Converting this to a TemplateField is very simple. You need to add an ItemTemplate that displays the PlayerName using a Literal and an EditItemTemplate that has a TextBox that allows the PlayerName to be edited:

```
<asp:TemplateField HeaderText="PlayerName"
  SortExpression="PlayerName">
  <ItemTemplate>
    <asp:Literal ID="litPlayerName" runat="server"
      Text='<%# Eval("PlayerName") %>'></asp:Literal>
  </ItemTemplate>
  <EditItemTemplate>
    <asp:TextBox id="txtPlayerName" runat="server"
      Text='<%# Bind("PlayerName") %>'></asp:TextBox>
  </EditItemTemplate>
</asp:TemplateField>
```

With a DetailsView, you could also add an InsertItemTemplate to use when inserting a new record. If you don't specify an InsertItemTemplate, the EditItemTemplate will be used when the DetailsView is in Insert mode. For that reason, you may still need to use the InsertVisible property on the TemplateField to indicate that the Field is not to be displayed when adding a new record.

The one BoundField property that isn't supported is ReadOnly. If you need to have a ReadOnly Field, then simply leave the BoundField as it is. It will always be displayed, and the user will never be able to modify it. You could also mimic its functionality by defining only an ItemTemplate. If the DetailsView or GridView is in Edit mode and no EditItemTemplate is defined, the ItemTemplate is used. If it's in Insert mode and no InsertItemTemplate or EditItemTemplate is defined, then the ItemTemplate is used.

Once you've converted all of your BoundField controls to TemplateField controls. you can then add the required validation to the page. When you try to edit or add data to the database, the validation will occur. If it fails, the action will be canceled and the validation results shown.

We'll now look at how you can change the DetailsView from earlier to add validation to the page.

■ **Note** Adding validation Web controls to the GridView and FormView is similar to adding validation to the DetailsView. In the code download, you'll find two extra pages, named Players_Basic_Validation.aspx and Players_Form_Validation.aspx, which have validation added to them.

Try It Out: Adding Validation to the DetailsView

In this example, you'll modify the existing `DetailsView` example to add validation Web controls to the page. You'll also see that Visual Web Developer provides a handy means of editing templates through the graphical designer.

1. Copy `Players_Details.aspx` and rename the copy `Players_Details_Validation.aspx`.

2. In the Source view, find the HTML markup for the `DetailsView` and replace the PlayerName, PlayerCost, and PlayerStorage `BoundField` controls with `TemplateField` controls.

3. Replace the PlayerName `BoundField` with the following:

```
<asp:TemplateField HeaderText="PlayerName"
  SortExpression="PlayerName">
  <ItemTemplate>
    <asp:Literal ID="litPlayerName" runat="server"
      Text='<%# Eval("PlayerName") %>'></asp:Literal>
  </ItemTemplate>
  <EditItemTemplate>
    <asp:TextBox id="txtPlayerName" runat="server"
      Text='<%# Bind("PlayerName") %>'></asp:TextBox>
  </EditItemTemplate>
</asp:TemplateField>
```

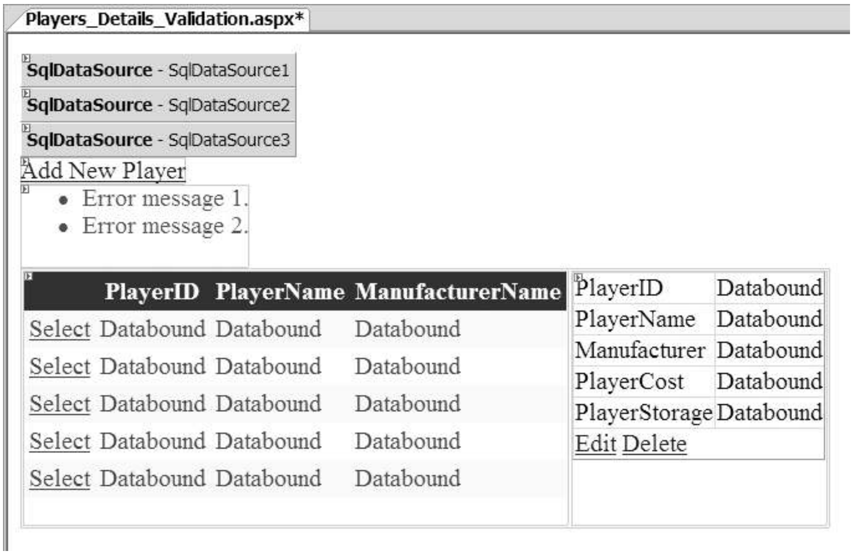
4. Replace the PlayerCost `BoundField` with the following:

```
<asp:TemplateField HeaderText="PlayerCost"
  SortExpression="PlayerCost">
  <ItemTemplate>
    <asp:Literal ID="litPlayerCost" runat="server"
      Text='<%# Eval("PlayerCost") %>'></asp:Literal>
  </ItemTemplate>
  <EditItemTemplate>
    <asp:TextBox id="txtPlayerCost" runat="server"
      Text='<%# Bind("PlayerCost") %>'></asp:TextBox>
  </EditItemTemplate>
</asp:TemplateField>
```

5. Replace the PlayerStorage `BoundField` with the following:

```
<asp:TemplateField HeaderText="PlayerStorage"
  SortExpression="PlayerStorage">
  <ItemTemplate>
    <asp:Literal ID="litPlayerStorage" runat="server"
      Text='<%# Eval("PlayerStorage") %>'></asp:Literal>
  </ItemTemplate>
  <EditItemTemplate>
    <asp:TextBox id="txtPlayerStorage" runat="server"
      Text='<%# Bind("PlayerStorage") %>'></asp:TextBox>
  </EditItemTemplate>
</asp:TemplateField>
```

6. Switch to the Design view and add a `ValidationSummary` after the Add New Player link, as shown in Figure 9-23.


Players_Details_Validation.aspx*
SqlDataSource - SqlDataSource1
SqlDataSource - SqlDataSource2
SqlDataSource - SqlDataSource3
Add New Player
• Error message 1.
• Error message 2.

	PlayerID	PlayerName	ManufacturerName
Select	Databound	Databound	Databound
Select	Databound	Databound	Databound
Select	Databound	Databound	Databound
Select	Databound	Databound	Databound
Select	Databound	Databound	Databound

PlayerID	Databound
PlayerName	Databound
Manufacturer	Databound
PlayerCost	Databound
PlayerStorage	Databound
Edit Delete	

Figure 9-23. *Adding a ValidationSummary to the page*

7. Select the `DetailsView` and open its Tasks menu. Select the Edit Templates option. The `DetailsView` will change its appearance. If you expand the Display drop-down list in the Tasks menu, you'll see that you can switch to the different `TemplateField` controls and the different Web controls within each `TemplateField`, as shown in Figure 9-24.

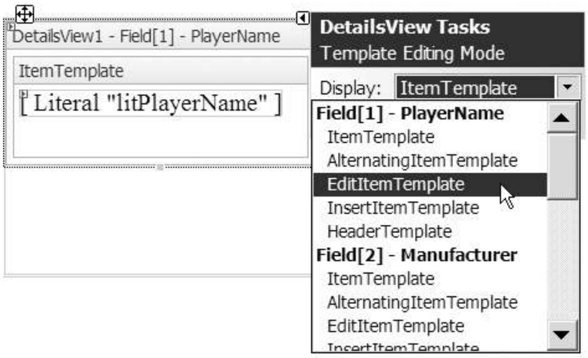


Figure 9-24. *Selecting the correct TemplateField and template to edit*

8. Select `EditItemTemplate` from beneath the PlayerName `TemplateField`, and you'll see the `EditItemTemplate` showing a `TextBox`.

9. Add a `RequiredFieldValidator` to the `EditItemTemplate`. Set its `Display` property to `Dynamic`, `Text` property to *, and `ErrorMessage` to **You must enter a name**. Finally, set the `ControlToValidate` property to `txtPlayerName`. The `EditItemTemplate` should now look similar to Figure 9-25.

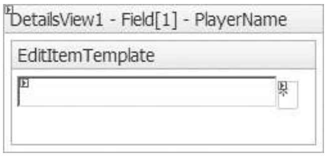


Figure 9-25. *Adding a RequiredFieldValidator to the EditItemTemplate*

10. Switch to the PlayerCost `EditItemTemplate` and add a `RequiredFieldValidator` to the template. Set its properties as follows:

    - `Display`: Dynamic
    - `Text`: *
    - `ErrorMessage`: You must enter a cost
    - `ControlToValidate`: `txtPlayerCost`

11. Add a `CompareValidator` to the template and set its properties as follows:

    - `Display`: Dynamic
    - `Text`: *
    - `ErrorMessage`: You must specify the cost as a decimal
    - `ControlToValidate: txtPlayerCost`
    - `Operator`: `DataTypeCheck`
    - `Type`: Currency

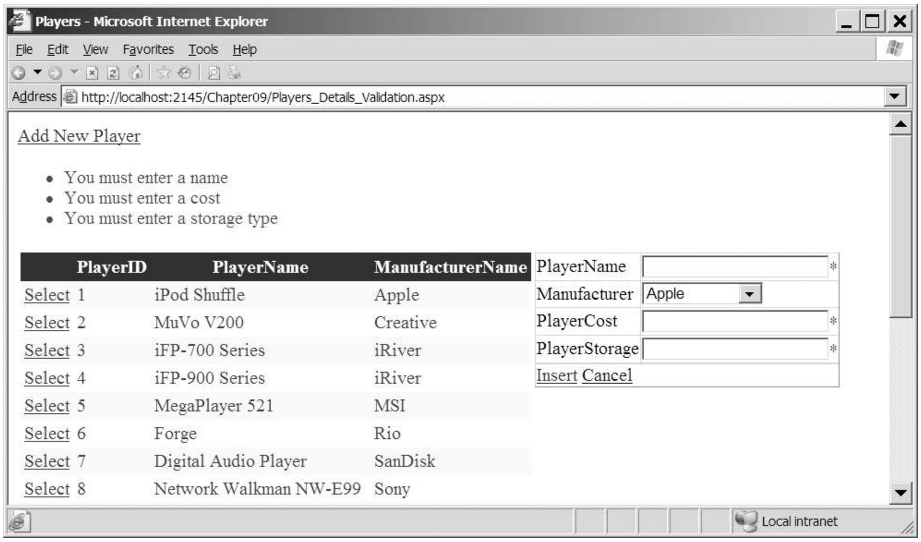
12. Switch to the PlayerStorage `EditItemTemplate` and add a `RequiredFieldValidator` to the template. Set its properties as follows:

    - `Display`: Dynamic
    - `Text`: *
    - `ErrorMessage`: You must enter a storage type
    - `ControlToValidate`: `txtPlayerStorage`

13. From the `DetailsView` Tasks menu, select the End Template Editing option to switch the `DetailsView` back to its normal view.

14. Set the `CausesValidation` property of the Add New Player `LinkButton` to `false`.

15. Save the page, and then view it in the browser. You'll find that it is now impossible to edit a Player or add a new Player if you haven't specified all of the required values. If you try to do this, you'll see that the validators report the errors, as shown in Figure 9-26.


Players - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:2145/Chapter09/Players_Details_Validation.aspx
Local intranet

Add New Player

- You must enter a name
- You must enter a cost
- You must enter a storage type

	PlayerID	PlayerName	ManufacturerName
Select	1	iPod Shuffle	Apple
Select	2	MuVo V200	Creative
Select	3	iFP-700 Series	iRiver
Select	4	iFP-900 Series	iRiver
Select	5	MegaPlayer 521	MSI
Select	6	Forge	Rio
Select	7	Digital Audio Player	SanDisk
Select	8	Network Walkman NW-E99	Sony

PlayerName	*
Manufacturer	Apple
PlayerCost	*
PlayerStorage	*
Insert Cancel	

Figure 9-26. *The validators fire if the data is incorrect.*

How It Works

The validators you've added to the different templates are the same as the validators that you added to the examples in Chapter 8. You add three `RequiredFieldValidator` controls to make sure that the user enters all of the required data. You've used a `CompareValidator` to ensure the PlayerCost is entered as a valid currency.

As we've already discussed, you need to turn any `BoundField` controls that you want to validate into `TemplateField` controls with the correct templates defined. If you don't want to edit a `BoundField` or have no reason to validate the user's entry, then you can leave it as a `BoundField` (with the `ReadOnly` and `InsertVisible` properties set correctly).

In this example, you can leave the PlayerID `BoundField` as it is. It's already set as `ReadOnly`, so you can't edit it, and it's also set so that it doesn't appear when you're inserting a Player. The other three `BoundField` controls are converted to `TemplateField` controls quite easily (although it can quickly become quite tedious).

Once the required templates are added, you can use the design-time features of the `DetailsView` to configure the validators. By selecting the Edit Templates option, you can use the normal design-time tools to edit the individual templates within the Web control. (If you've ever had to deal with templates in earlier version of the development tools, you'll appreciate how much of an improvement this actually is.)

■ **Note** The `GridView` works exactly the same as the `DetailsView` when editing templates: you need to pick the Field you require, and then choose the template within that Field. The `FormView` is slightly different as it has no concept of Fields, so you can select only the template.

Selecting a template within a `TemplateField` turns that template into its own little design area, showing only that template and none of the others. You can use the normal tools available in Visual Web Developer within this template.

The important thing to remember is that when editing a template, any Web controls within the template can see only other Web controls within that template. If you look at any of the validation Web controls and expand the `ControlToValidate` drop-down list, you'll see that only the `TextBox` in the template is available. The `TextBox` controls defined in other templates aren't visible.

The validators run at both the client and the server. The client-side validators completely stop the page from being posted back to the server if it has any errors. If the client-side validators accept the page, it is posted back to the server, and the server-side validators are processed before any other events. If the server-side validation fails, the `IsValid` property of the `Page` is set to `false`. As the `Page` is now invalid, the insert or update action is canceled. The `ItemInserting`/`ItemInserted` or `ItemUpdating`/`ItemUpdated` events are no longer fired, and the `INSERT` or `UPDATE` isn't attempted.

Summary

In this chapter, we've looked at using the `GridView`, `DetailsView`, and `FormView` for modifying data in the database.

From working through the chapter, you may have the impression that the `GridView`, `DetailsView`, and `FormView` are the ideal answer when you need to interact with the database. However, you saw two examples of when this isn't the case. The design of our database includes a many-to-many relationship between the Player table and the Format table, and this causes problems.

When deleting a Player from the database, you had to add a second `DELETE` query to delete any related data in the WhatPlaysWhatFormat table. This can't be handled automatically by any of the Web controls that you've used. The `SqlDataSource` can handle only a single query in the `DeleteStatement` property.

The WhatPlaysWhatFormat table also causes problems when you want to add and edit data in the database. Because it's a many-to-many relationship, you must handle it slightly differently. As you saw in Chapter 8, you need to write extra code to retrieve the available Formats from the database and then determine which of those is supported by the Player. You then had to write extra code to massage the user's selection back into the database. The `SqlDataSource` just can't deal with this many-to-many relationship.

That's not to say that the Web controls we've looked at in this chapter aren't worthwhile. The Manufacturer and Format tables are ideal candidates for these Web controls, and the code download includes two pages, named `Manufacturers_Details.aspx` and `Formats_Details.aspx`, that show that the Web controls are suitable in some situations.

As with all design choices you make, you need to look carefully at what you're trying to accomplish and choose the right paradigm for the job at hand.

You're nearly ready to start building your own Web sites. At the moment, you can build your own pages that use various Web controls and data-binding techniques to display data to the user, as well as allow adding, editing, and deleting data in the database.

In the next chapter, I'll introduce you to stored procedures. Until now, you've always included the SQL query that you want to execute within the page. If another page wanted to execute the same query, you would need to repeat the SQL for the query on the other page as well. If a third page required the same query, you would have yet another copy of the same SQL query. What if you now discover that the query is actually incorrect? You would need to change it in all three places. Stored procedures solve this problem.