# Manually Configuring Your *DataAdapter* Objects

The *DataAdapter* object exposes four properties that contain *Command* objects. You've already learned that the *SelectCommand* property contains the *Command* that the *DataAdapter* uses to fill your *DataTable*. The other three properties—*UpdateCommand*, *InsertCommand*, and *DeleteCommand*—contain the *Command* objects that the *DataAdapter* uses to submit pending changes.

This architecture represents a major change from the ADO object model. There is no magical "black box" technology involved. You control how the *DataAdapter* submits pending changes because you supply the *Command* objects that the *DataAdapter* uses.

The *DataAdapter* object's *Update* method is very flexible. You can supply a *DataSet*, a *DataSet* and a table name, a *DataTable*, or an array of *DataRow* objects. Regardless of how you call the *DataAdapter* object's *Update* method, the *DataAdapter* will attempt to submit the pending changes through the appropriate *Command*. All the work we performed earlier in the *SubmitChangesByHand* procedure can be accomplished using a single call to the *DataAdapter* object's *Update* method.

# Introducing Bound Parameters

The *SubmitChangesByHand* procedure that we created was not terribly complex. The procedure also didn't do much work. Instead, it delegated the nasty work to one of three functions: *SubmitUpdate*, *SubmitInsert*, or *SubmitDelete*. These functions populate the values for the parameters in the appropriate query based on the contents of the modified row.

We'll use the same parameterized queries to submit pending changes using a *DataAdapter*.

```
UPDATE [Order Details]
    SET OrderID = ?, ProductID = ?, Quantity = ?, UnitPrice = ?
    WHERE OrderID = ?  AND ProductID = ? AND
          Quantity = ? AND UnitPrice = ?

INSERT INTO [Order Details] (OrderID, ProductID, Quantity, UnitPrice)
    VALUES (?, ?, ?, ?)

DELETE FROM [Order Details]
    WHERE OrderID = ? AND ProductID = ? AND
          Quantity = ? AND UnitPrice = ?
```

However, when we add *Parameter* objects to the *DataAdapter* object's *Command* objects, we'll use two properties of the ADO.NET *Parameter* object that are designed specifically for updates using the *DataAdapter*: *SourceColumn* and *SourceVersion*.

These properties basically bind a *Parameter* to a *DataColumn* in your *DataTable*. The *DataAdapter* uses these properties to determine how to populate the *Parameter* object's *Value* property before executing the query, similar to how we

accomplished this task in the *SubmitUpdate*, *SubmitInsert*, and *SubmitDelete* functions. Figure 10-2 better illustrates this behavior.

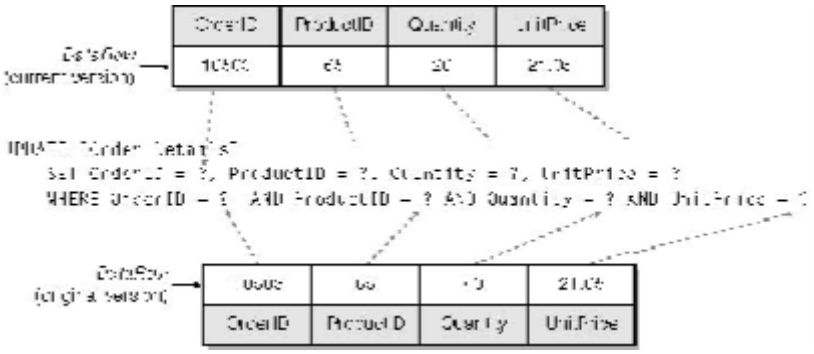

**Figure 10-2**
*Binding Parameter objects to DataColumn objects.*

The following code snippet creates our parameterized *Command* objects but sets the *SourceColumn* and *SourceVersion* properties of the *Parameter* objects. The default value for the *SourceVersion* property is *DataRowVersion.Current*, so we need to set the property only if we want to bind the *Parameter* objects to the original values in the desired column.

## Visual Basic .NET

```
Private Function CreateDataAdapterUpdateCommand() As OleDbCommand
    Dim strSQL As String
    strSQL = "UPDATE [Order Details] " & _
             "    SET OrderID = ?, ProductID = ?, " & _
             "        Quantity = ?, UnitPrice = ? " & _
             "    WHERE OrderID = ?  AND ProductID = ? AND " & _
             "          Quantity = ? AND UnitPrice = ?"
    Dim cmd As New OleDbCommand(strSQL, cn)

    Dim pc As OleDbParameterCollection = cmd.Parameters
    pc.Add("OrderID_New", OleDbType.Integer, 0, "OrderID")
    pc.Add("ProductID_New", OleDbType.Integer, 0, "ProductID")
    pc.Add("Quantity_New", OleDbType.SmallInt, 0, "Quantity")
    pc.Add("UnitPrice_New", OleDbType.Currency, 0, "UnitPrice")

    Dim param As OleDbParameter
    param = pc.Add("OrderID_Orig", OleDbType.Integer, 0, "OrderID")
    param.SourceVersion = DataRowVersion.Original
    param = pc.Add("ProductID_Orig", OleDbType.Integer, 0, _
                   "ProductID")
    param.SourceVersion = DataRowVersion.Original
    param = pc.Add("Quantity_Orig", OleDbType.SmallInt, 0, _
                   "Quantity")
    param.SourceVersion = DataRowVersion.Original
    param = pc.Add("UnitPrice_Orig", OleDbType.Currency, 0, _
                   "UnitPrice")
    param.SourceVersion = DataRowVersion.Original
```

```
    Return cmd
End Function

Private Function CreateDataAdapterInsertCommand() As OleDbCommand
    Dim strSQL As String
    strSQL = "INSERT INTO [Order Details] " & _
             "   (OrderID, ProductID, Quantity, UnitPrice) " & _
             "   VALUES (?, ?, ?, ?)"
    Dim cmd As New OleDbCommand(strSQL, cn)

    Dim pc As OleDbParameterCollection = cmd.Parameters
    pc.Add("OrderID", OleDbType.Integer, 0, "OrderID")
    pc.Add("ProductID", OleDbType.Integer, 0, "ProductID")
    pc.Add("Quantity", OleDbType.SmallInt, 0, "Quantity")
    pc.Add("UnitPrice", OleDbType.Currency, 0, "UnitPrice")

    Return cmd
End Function

Private Function CreateDataAdapterDeleteCommand() As OleDbCommand
    Dim strSQL As String
    strSQL = "DELETE FROM [Order Details] " & _
             "    WHERE OrderID = ? AND ProductID = ? AND " & _
             "          Quantity = ? AND UnitPrice = ?"
    Dim cmd As New OleDbCommand(strSQL, cn)

    Dim pc As OleDbParameterCollection = cmd.Parameters
    Dim param As OleDbParameter
    pc.Add("OrderID", OleDbType.Integer, 0, "OrderID")
    param.SourceVersion = DataRowVersion.Original
    pc.Add("ProductID", OleDbType.Integer, 0, "ProductID")
    param.SourceVersion = DataRowVersion.Original
    pc.Add("Quantity", OleDbType.SmallInt, 0, "Quantity")
    param.SourceVersion = DataRowVersion.Original
    pc.Add("UnitPrice", OleDbType.Currency, 0, "UnitPrice")
    param.SourceVersion = DataRowVersion.Original

    Return cmd
End Function
```

## Visual C# .NET

```
static OleDbCommand CreateDataAdapterUpdateCommand()
{
    string strSQL;
    strSQL = "UPDATE [Order Details] " & _
             "    SET OrderID = ?, ProductID = ?, " +
             "        Quantity = ?, UnitPrice = ? " +
             "    WHERE OrderID = ? AND ProductID = ? AND " +
             "          Quantity = ? AND UnitPrice = ?";
    OleDbCommand cmd = new OleDbCommand(strSQL, cn);
```

```
    OleDbParameterCollection pc = cmd.Parameters;
    pc.Add("OrderID_New", OleDbType.Integer, 0, "OrderID");
    pc.Add("ProductID_New", OleDbType.Integer, 0, "ProductID");
    pc.Add("Quantity_New", OleDbType.SmallInt, 0, "Quantity");
    pc.Add("UnitPrice_New", OleDbType.Currency, 0, "UnitPrice");

    OleDbParameter param;
    param = pc.Add("OrderID_Orig", OleDbType.Integer, 0, "OrderID");
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("ProductID_Orig", OleDbType.Integer, 0,
                   "ProductID");
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("Quantity_Orig", OleDbType.SmallInt, 0,
                   "Quantity");
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("UnitPrice_Orig", OleDbType.Currency, 0,
                   "UnitPrice");
    param.SourceVersion = DataRowVersion.Original;

    return cmd;
}

static OleDbCommand CreateDataAdapterInsertCommand()
{
    string strSQL;
    strSQL = "INSERT INTO [Order Details] " +
             "    (OrderID, ProductID, Quantity, UnitPrice) " +
             "    VALUES (?, ?, ?, ?)";
    OleDbCommand cmd = new OleDbCommand(strSQL, cn);

    OleDbParameterCollection pc = cmd.Parameters;
    pc.Add("OrderID", OleDbType.Integer, 0, "OrderID");
    pc.Add("ProductID", OleDbType.Integer, 0, "ProductID");
    pc.Add("Quantity", OleDbType.SmallInt, 0, "Quantity");
    pc.Add("UnitPrice", OleDbType.Currency, 0, "UnitPrice");

    return cmd;
}

static OleDbCommand CreateDataAdapterDeleteCommand()
{
    string strSQL;
    strSQL = "DELETE FROM [Order Details] " +
             "    WHERE OrderID = ? AND ProductID = ? AND " +
             "          Quantity = ? AND UnitPrice = ?";
    OleDbCommand cmd = new OleDbCommand(strSQL, cn);

    OleDbParameter param;
    OleDbParameterCollection pc = cmd.Parameters;
    param = pc.Add("OrderID", OleDbType.Integer, 0, "OrderID");
```

```
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("ProductID", OleDbType.Integer, 0, "ProductID");
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("Quantity", OleDbType.SmallInt, 0, "Quantity");
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("UnitPrice", OleDbType.Currency, 0, "UnitPrice");
    param.SourceVersion = DataRowVersion.Original;

    return cmd;
}
```

We can now replace the *SubmitChangesByHand*, *SubmitUpdate*, *SubmitInsert*, and *SubmitDelete* procedures with the following code:

## Visual Basic .NET

```
Private Sub SubmitChangesViaDataAdapter()
    da.UpdateCommand = CreateDataAdapterUpdateCommand()
    da.InsertCommand = CreateDataAdapterInsertCommand()
    da.DeleteCommand = CreateDataAdapterDeleteCommand()
    da.Update(tbl)
End Sub
```

## Visual C# .NET

```
static void SubmitChangesViaDataAdapter()
{
    da.UpdateCommand = CreateDataAdapterUpdateCommand();
    da.InsertCommand = CreateDataAdapterInsertCommand();
    da.DeleteCommand = CreateDataAdapterDeleteCommand();
    da.Update(tbl);
}
```

# Using Stored Procedures to Submit Updates

A common complaint of developers who used ADO to retrieve data from their databases was that they couldn't use the *Recordset* object's *UpdateBatch* method to submit updates using stored procedures.

Earlier, I mentioned that the *DataAdapter* lets you define your own updating logic. The previous code snippets showed how you can build your own *Command* objects that the *DataAdapter* can then use to submit pending changes. We can use similar code to submit updates using stored procedures.

First we need to define stored procedures in the Northwind database that can modify, insert, and delete rows from the Order Details table. You can paste and then execute the following code in SQL Query Analyzer to create the stored procedures that we're going to call in our code. If you don't have access to SQL Query Analyzer because you have only MSDE installed, you can call a procedure named *CreateSprocs* (which appears in a later code snippet) to create the desired stored procedures.

```
USE Northwind

GO

CREATE PROCEDURE spUpdateDetail
    (@OrderID_New int, @ProductID_New int,
     @Quantity_New smallint, @UnitPrice_New money,
     @OrderID_Orig int, @ProductID_Orig int,
     @Quantity_Orig smallint, @UnitPrice_Orig money)
AS
UPDATE [Order Details]
    SET OrderID = @OrderID_New, ProductID = @ProductID_New,
        Quantity = @Quantity_New, UnitPrice = @UnitPrice_New
    WHERE OrderID = @OrderID_Orig AND ProductID = @ProductID_Orig AND
          Quantity = @Quantity_Orig AND UnitPrice = @UnitPrice_Orig

GO

CREATE PROCEDURE spInsertDetail
    (@OrderID int, @ProductID int,
     @Quantity smallint, @UnitPrice money)
AS
INSERT INTO [Order Details]
    (OrderID, ProductID, Quantity, UnitPrice)
    VALUES (@OrderID, @ProductID, @Quantity, @UnitPrice)

GO

CREATE PROCEDURE spDeleteDetail
    (@OrderID int, @ProductID int,
     @Quantity smallint, @UnitPrice money)
AS
DELETE FROM [Order Details]
    WHERE OrderID = @OrderID AND ProductID = @ProductID AND
          Quantity = @Quantity AND UnitPrice = @UnitPrice
```

Now that we have stored procedures that we can call to submit changes to the Order Details table, we can write *Command* objects to call those stored procedures automatically when we call the *DataAdapter* object's *Update* method.

The following code snippet contains functions that create *Command* objects that contain calls to the stored procedures I just described. It also contains a procedure you can call to create those stored procedures in your database. All that's left to do to submit updates using stored procedures is to wire up our new *Command* objects to the *DataAdapter*, which we can do in the *SubmitChangesViaStoredProcedures* procedure.

## Visual Basic .NET

```
Private Sub SubmitChangesViaStoredProcedures()
    da.UpdateCommand = CreateUpdateViaSPCommand()
```

```
    da.InsertCommand = CreateInsertViaSPCommand()
    da.DeleteCommand = CreateDeleteViaSPCommand()
    da.Update(tbl)
End Sub

Private Function CreateUpdateViaSPCommand() As OleDbCommand
    Dim cmd As New OleDbCommand("spUpdateDetail", cn)
    cmd.CommandType = CommandType.StoredProcedure

    Dim pc As OleDbParameterCollection = cmd.Parameters
    pc.Add("OrderID_New", OleDbType.Integer, 0, "OrderID")
    pc.Add("ProductID_New", OleDbType.Integer, 0, "ProductID")
    pc.Add("Quantity_New", OleDbType.SmallInt, 0, "Quantity")
    pc.Add("UnitPrice_New", OleDbType.Currency, 0, "UnitPrice")

    Dim param As OleDbParameter
    param = pc.Add("OrderID_Orig", OleDbType.Integer, 0, "OrderID")
    param.SourceVersion = DataRowVersion.Original
    param = pc.Add("ProductID_Orig", OleDbType.Integer, 0, _
                   "ProductID")
    param.SourceVersion = DataRowVersion.Original
    param = pc.Add("Quantity_Orig", OleDbType.SmallInt, 0, _
                   "Quantity")
    param.SourceVersion = DataRowVersion.Original
    param = pc.Add("UnitPrice_Orig", OleDbType.Currency, 0, _
                   "UnitPrice")
    param.SourceVersion = DataRowVersion.Original

    Return cmd
End Function

Private Function CreateInsertViaSPCommand() As OleDbCommand
    Dim cmd As New OleDbCommand("spInsertDetail", cn)
    cmd.CommandType = CommandType.StoredProcedure

    Dim pc As OleDbParameterCollection = cmd.Parameters
    pc.Add("OrderID", OleDbType.Integer, 0, "OrderID")
    pc.Add("ProductID", OleDbType.Integer, 0, "ProductID")
    pc.Add("Quantity", OleDbType.SmallInt, 0, "Quantity")
    pc.Add("UnitPrice", OleDbType.Currency, 0, "UnitPrice")

    Return cmd
End Function

Private Function CreateDeleteViaSPCommand() As OleDbCommand
    Dim cmd As New OleDbCommand("spDeleteDetail", cn)
    cmd.CommandType = CommandType.StoredProcedure

    Dim pc As OleDbParameterCollection = cmd.Parameters
    Dim param As OleDbParameter
    param = pc.Add("OrderID", OleDbType.Integer, 0, "OrderID")
```

```
    param.SourceVersion = DataRowVersion.Original
    param = pc.Add("ProductID", OleDbType.Integer, 0, "ProductID")
    param.SourceVersion = DataRowVersion.Original
    param = pc.Add("Quantity", OleDbType.SmallInt, 0, "Quantity")
    param.SourceVersion = DataRowVersion.Original
    param = pc.Add("UnitPrice", OleDbType.Currency, 0, "UnitPrice")
    param.SourceVersion = DataRowVersion.Original

    Return cmd
End Function

Private Sub CreateSprocs()
    Dim cmd As OleDbCommand = cn.CreateCommand
    Dim strSQL As String

    strSQL = "CREATE PROCEDURE spUpdateDetail " & vbCrLf & _
             "    (@OrderID_New int, @ProductID_New int, " & vbCrLf & _
             "     @Quantity_New smallint, " & vbCrLf & _
             "     @UnitPrice_New money, " & vbCrLf & _
             "     @OrderID_Orig int, " & vbCrLf & _
             "     @ProductID_Orig int, " & vbCrLf & _
             "     @Quantity_Orig smallint, " & vbCrLf & _
             "     @UnitPrice_Orig money) " & vbCrLf & _
             "AS " & vbCrLf & _
             "UPDATE [Order Details] " & vbCrLf & _
             "    SET OrderID = @OrderID_New, " & vbCrLf & _
             "        ProductID = @ProductID_New, " & vbCrLf & _
             "        Quantity = @Quantity_New, " & vbCrLf & _
             "        UnitPrice = @UnitPrice_New " & vbCrLf & _
             "    WHERE OrderID = @OrderID_Orig AND " & vbCrLf &_
             "          ProductID = @ProductID_Orig AND " & vbCrLf & _
             "          Quantity = @Quantity_Orig AND " & vbCrLf & _
             "          UnitPrice = @UnitPrice_Orig"
    cmd.CommandText = strSQL
    cmd.ExecuteNonQuery()

    strSQL = "CREATE PROCEDURE spInsertDetail " & vbCrLf & _
             "    (@OrderID int, @ProductID int, " & vbCrLf & _
             "     @Quantity smallint, @UnitPrice money) " & vbCrLf & _
             "AS " & vbCrLf & _
             "INSERT INTO [Order Details] " & vbCrLf & _
             "    (OrderID, ProductID, Quantity, UnitPrice) " & vbCrLf &
             "    VALUES (@OrderID, @ProductID, @Quantity, @UnitPrice)"
    cmd.CommandText = strSQL
    cmd.ExecuteNonQuery()

    strSQL = "CREATE PROCEDURE spDeleteDetail " & vbCrLf & _
             "    (@OrderID int, @ProductID int, " & vbCrLf & _
             "     @Quantity smallint, @UnitPrice money) " & vbCrLf & _
             "AS " & vbCrLf & _
             "DELETE FROM [Order Details] " & vbCrLf & _
```

```
            "    WHERE OrderID = @OrderID AND " & vbCrLf & _
            "          ProductID = @ProductID AND " & vbCrLf & _
            "          Quantity = @Quantity AND UnitPrice = @UnitPrice"
    cmd.CommandText = strSQL
    cmd.ExecuteNonQuery()
End Sub
```

## Visual C# .NET

```
static void SubmitChangesViaStoredProcedures()
{
    da.UpdateCommand = CreateUpdateViaSPCommand();
    da.InsertCommand = CreateInsertViaSPCommand();
    da.DeleteCommand = CreateDeleteViaSPCommand();
    da.Update(tbl);
}

static OleDbCommand CreateUpdateViaSPCommand()
{
    OleDbCommand cmd = new OleDbCommand("spUpdateDetail", cn);
    cmd.CommandType = CommandType.StoredProcedure;

    OleDbParameterCollection pc = cmd.Parameters;
    pc.Add("OrderID_New", OleDbType.Integer, 0, "OrderID");
    pc.Add("ProductID_New", OleDbType.Integer, 0, "ProductID");
    pc.Add("Quantity_New", OleDbType.SmallInt, 0, "Quantity");
    pc.Add("UnitPrice_New", OleDbType.Currency, 0, "UnitPrice");

    OleDbParameter param;
    param = pc.Add("OrderID_Orig", OleDbType.Integer, 0, "OrderID");
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("ProductID_Orig", OleDbType.Integer, 0, "ProductID");
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("Quantity_Orig", OleDbType.SmallInt, 0, "Quantity");
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("UnitPrice_Orig", OleDbType.Currency, 0, "UnitPrice");
    param.SourceVersion = DataRowVersion.Original;

    return cmd;
}

static OleDbCommand CreateInsertViaSPCommand()
{
    OleDbCommand cmd = new OleDbCommand("spInsertDetail", cn);
    cmd.CommandType = CommandType.StoredProcedure;

    OleDbParameterCollection pc = cmd.Parameters;
    pc.Add("OrderID", OleDbType.Integer, 0, "OrderID");
    pc.Add("ProductID", OleDbType.Integer, 0, "ProductID");
    pc.Add("Quantity", OleDbType.SmallInt, 0, "Quantity");
    pc.Add("UnitPrice", OleDbType.Currency, 0, "UnitPrice");
```

```
    return cmd;
}

static OleDbCommand CreateDeleteViaSPCommand()
{
    OleDbCommand cmd = new OleDbCommand("spDeleteDetail", cn);
    cmd.CommandType = CommandType.StoredProcedure;

    OleDbParameterCollection pc = cmd.Parameters;
    OleDbParameter param;
    param = pc.Add("OrderID", OleDbType.Integer, 0, "OrderID");
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("ProductID", OleDbType.Integer, 0, "ProductID");
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("Quantity", OleDbType.SmallInt, 0, "Quantity");
    param.SourceVersion = DataRowVersion.Original;
    param = pc.Add("UnitPrice", OleDbType.Currency, 0, "UnitPrice");
    param.SourceVersion = DataRowVersion.Original;

    return cmd;
}

static void CreateSprocs()
{
    OleDbCommand cmd = cn.CreateCommand();
    string strSQL;

    strSQL = "CREATE PROCEDURE spUpdateDetail \n\r" +
             "    (@OrderID_New int, @ProductID_New int, \n\r" +
             "     @Quantity_New smallint, @UnitPrice_New money, \n\r" +
             "     @OrderID_Orig int, @ProductID_Orig int, \n\r" +
             "     @Quantity_Orig smallint, @UnitPrice_Orig money) \n\r"
             "AS \n\r" +
             "UPDATE [Order Details] \n\r" +
             "    SET OrderID = @OrderID_New, \n\r" +
             "        ProductID = @ProductID_New, \n\r" +
             "        Quantity = @Quantity_New, \n\r" +
             "        UnitPrice = @UnitPrice_New \n\r" +
             "    WHERE OrderID = @OrderID_Orig AND \n\r" +
             "          ProductID = @ProductID_Orig AND \n\r" +
             "          Quantity = @Quantity_Orig AND \n\r" +
             "          UnitPrice = @UnitPrice_Orig";
    cmd.CommandText = strSQL;
    cmd.ExecuteNonQuery();

    strSQL = "CREATE PROCEDURE spInsertDetail \n\r" +
             "    (@OrderID int, @ProductID int, \n\r" +
             "     @Quantity smallint, @UnitPrice money) \n\r" +
             "AS \n\r" +
             "INSERT INTO [Order Details] \n\r" +
```

```
            "    (OrderID, ProductID, Quantity, UnitPrice) \n\r" +
            "    VALUES (@OrderID, @ProductID, @Quantity, @UnitPrice)";
    cmd.CommandText = strSQL;
    cmd.ExecuteNonQuery();

    strSQL = "CREATE PROCEDURE spDeleteDetail \n\r" +
            "    (@OrderID int, @ProductID int, \n\r" +
            "     @Quantity smallint, @UnitPrice money) \n\r" +
            "AS \n\r" +
            "DELETE FROM [Order Details] \n\r" +
            "    WHERE OrderID = @OrderID AND \n\r" +
            "          ProductID = @ProductID AND \n\r" +
            "          Quantity = @Quantity AND UnitPrice = @UnitPrice";
    cmd.CommandText = strSQL;
    cmd.ExecuteNonQuery();
}
```

# Supplying Your Own Updating Logic

Now let's look at the benefits and drawbacks of supplying your own updating logic in code.

## Benefits

The two biggest benefits of supplying your own updating logic are control and performance. The ADO.NET *DataAdapter* offers you more control over your updating logic than any previous Microsoft data access technology. You're no longer restricted to submitting updates directly against tables; you can finally leverage your stored procedures in a RAD way.

Plus, because you're not relying on the data access technology to determine the origin of your data, you can treat any result set as updateable. With the ADO cursor engine, if the cursor engine cannot gather the metadata necessary to submit changes back to your database, there is no way for you to supply that information programmatically. With ADO.NET, you can fill your *DataSet* with the results of a stored procedure call, a query against a temporary table, or the union of multiple queries—or fill it in any other way you see fit—and still be able to submit changes to your database.

Supplying updating logic in your code improves the performance of your application. The code snippet that used the ADO cursor engine to submit updates contained fewer lines of code, but it required the ADO cursor engine to query the database for the source table name, source column names, and primary key information for the source table. Querying database system tables for metadata and then using that metadata to generate updating logic takes more time than simply loading it from local code.

## Drawbacks

The drawbacks of supplying your own updating logic mirror the benefits of the ADO cursor engine's approach. First, it takes a lot more code to supply your own updating logic. Take a quick peek back, and compare how much code it took to submit updates using an ADO.NET *DataAdapter* with the ADO cursor engine's

approach. Writing that code is time consuming and rather tedious.

The other drawback is that many developers are not comfortable writing their own updating logic. They would rather not have to ponder such questions as: Do I need to delimit the table name in the query? What type of parameter markers should I use? Which columns should appear in the WHERE clause of the *CommandText* for the *UpdateCommand* and *DeleteCommand*? What is the appropriate setting for the *OleDbType* property for a parameter that contains a date/time value?

Thankfully, there are more RAD ways to generate your updating logic, as I'll explain in the upcoming sections.