Using ADO.NET *Command* Objects to Submit Updates

As you now know, the ADO cursor engine builds parameterized queries to submit updates. You can use what you learned in Chapter 4 to build equivalent parameterized queries in ADO.NET. Later in the chapter, you'll learn how to use these parameterized *Command* objects to submit the changes stored in an ADO.NET *DataSet* to your database.

Our ADO.NET *Command* objects will not be quite as dynamic as their ADO counterparts. To simplify the process, we'll build one *Command* to handle updates, one to handle insertions, and one to handle deletions. They'll be based on the following parameterized queries:

```
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 = ?
```

The UPDATE and INSERT queries submit new values to the database for each column in the original query. These queries reference each column in the original query in their WHERE clauses. This approach has its benefits and drawbacks, which I'll discuss later in the chapter.

The following code snippet builds our three parameterized *Command* objects. In each case, the code assumes that there is an externally defined *OleDbConnection* object called *cn*.

Visual Basic .NET

```
Private Function CreateUpdateCommand() 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)
    pc.Add("ProductID_New", OleDbType.Integer)
    pc.Add("Quantity_New", OleDbType.SmallInt)
    pc.Add("UnitPrice_New", OleDbType.Currency)

    pc.Add("OrderID_Orig", OleDbType.Integer)
    pc.Add("ProductID_Orig", OleDbType.Integer)
    pc.Add("Quantity_Orig", OleDbType.SmallInt)
    pc.Add("UnitPrice_Orig", OleDbType.Currency)

    Return cmd
End Function

Private Function CreateInsertCommand() 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)
    pc.Add("ProductID", OleDbType.Integer)
    pc.Add("Quantity", OleDbType.SmallInt)
    pc.Add("UnitPrice", OleDbType.Currency)

    Return cmd
End Function

Private Function CreateDeleteCommand() 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
    pc.Add("OrderID", OleDbType.Integer)
    pc.Add("ProductID", OleDbType.Integer)
    pc.Add("Quantity", OleDbType.SmallInt)
    pc.Add("UnitPrice", OleDbType.Currency)

    Return cmd
End Function
```

Visual C# .NET

```
static OleDbCommand CreateUpdateCommand()
{
    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);
    pc.Add("ProductID_New", OleDbType.Integer);
    pc.Add("Quantity_New", OleDbType.SmallInt);
    pc.Add("UnitPrice_New", OleDbType.Currency);

    pc.Add("OrderID_Orig", OleDbType.Integer);
    pc.Add("ProductID_Orig", OleDbType.Integer);
    pc.Add("Quantity_Orig", OleDbType.SmallInt);
    pc.Add("UnitPrice_Orig", OleDbType.Currency);

    return cmd;
}

static OleDbCommand CreateInsertCommand()
{
    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);
    pc.Add("ProductID", OleDbType.Integer);
    pc.Add("Quantity", OleDbType.SmallInt);
    pc.Add("UnitPrice", OleDbType.Currency);

    return cmd;
}

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

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

```
    return cmd;
}
```

Using our parameterized *Command* objects to submit updates is fairly straightforward. We need to examine the modified rows in our *DataTable* and determine the type of change stored in each of these rows (update, insert, or delete). Then we can use the contents of the row to populate the values of the parameters of the appropriate command.

After we call the *ExecuteNonQuery* method to execute the query stored in the *Command*, we can use the method's return value to determine whether the update attempt succeeded. If we successfully submit the pending change, we can call the *AcceptChanges* method of the *DataRow*. Otherwise, we can set the *DataRow* object's *RowError* property to indicate that the attempt to submit the pending change failed.

Visual Basic .NET

```
Private Sub SubmitChangesByHand()
    Dim cmdUpdate As OleDbCommand = CreateUpdateCommand()
    Dim cmdInsert As OleDbCommand = CreateInsertCommand()
    Dim cmdDelete As OleDbCommand = CreateDeleteCommand()
    Dim row As DataRow
    Dim intRowsAffected As Integer
    Dim dvrs As DataViewRowState
    dvrs = DataViewRowState.ModifiedCurrent _
           Or DataViewRowState.Deleted Or DataViewRowState.Added
    For Each row In tbl.Select("", "", dvrs)
        Select Case row.RowState
            Case DataRowState.Modified
                intRowsAffected = SubmitUpdate(row, cmdUpdate)
            Case DataRowState.Added
                intRowsAffected = SubmitInsert(row, cmdInsert)
            Case DataRowState.Deleted
                intRowsAffected = SubmitDelete(row, cmdDelete)
        End Select
        If intRowsAffected = 1 Then
            row.AcceptChanges()
        Else
            row.RowError = "Update attempt failed"
        End If
    Next row
End Sub

Private Function SubmitUpdate(ByVal row As DataRow, _
                              ByVal cmd As OleDbCommand) As Integer
    Dim pc As OleDbParameterCollection = cmd.Parameters
    pc("OrderID_New").Value = row("OrderID")
    pc("ProductID_New").Value = row("ProductID")
    pc("Quantity_New").Value = row("Quantity")
```

```
    pc("UnitPrice_New").Value = row("UnitPrice")
    pc("OrderID_Orig").Value = row("OrderID", _
                                   DataRowVersion.Original)
    pc("Quantity_Orig").Value = row("Quantity", _
                                    DataRowVersion.Original)
    pc("ProductID_Orig").Value = row("ProductID", _
                                     DataRowVersion.Original)
    pc("UnitPrice_Orig").Value = row("UnitPrice", _
                                     DataRowVersion.Original)
    Return cmd.ExecuteNonQuery
End Function

Private Function SubmitInsert(ByVal row As DataRow, _
                              ByVal cmd As OleDbCommand) As Integer
    Dim pc As OleDbParameterCollection = cmd.Parameters
    pc("OrderID").Value = row("OrderID")
    pc("ProductID").Value = row("ProductID")
    pc("Quantity").Value = row("Quantity")
    pc("UnitPrice").Value = row("UnitPrice")
    Return cmd.ExecuteNonQuery
End Function

Private Function SubmitDelete(ByVal row As DataRow, _
                              ByVal cmd As OleDbCommand) As Integer
    Dim pc As OleDbParameterCollection = cmd.Parameters
    pc("OrderID").Value = row("OrderID", DataRowVersion.Original)
    pc("ProductID").Value = row("ProductID", DataRowVersion.Original)
    pc("Quantity").Value = row("Quantity", DataRowVersion.Original)
    pc("UnitPrice").Value = row("UnitPrice", DataRowVersion.Original)
    Return cmd.ExecuteNonQuery
End Function
```

Visual C# .NET

```
static void SubmitChangesByHand()
{
    OleDbCommand cmdUpdate = CreateUpdateCommand();
    OleDbCommand cmdInsert = CreateInsertCommand();
    OleDbCommand cmdDelete = CreateDeleteCommand();
    DataViewRowState dvrs;
    dvrs = DataViewRowState.ModifiedCurrent ¦
           DataViewRowState.Deleted ¦ DataViewRowState.Added;
    int intRowsAffected = 0;
    foreach (DataRow row in tbl.Select("", "", dvrs))
    {
        switch (row.RowState)
        {
            case DataRowState.Modified:
                intRowsAffected = SubmitUpdate(row, cmdUpdate);
                break;
```

```
            case DataRowState.Added:
                intRowsAffected = SubmitInsert(row, cmdInsert);
                break;
            case DataRowState.Deleted:
                intRowsAffected = SubmitDelete(row, cmdDelete);
                break;
        }
        if (intRowsAffected == 1)
            row.AcceptChanges();
        else
            row.RowError = "Update attempt failed";
    }
}

static int SubmitUpdate(DataRow row, OleDbCommand cmd)
{
    OleDbParameterCollection pc = cmd.Parameters;
    pc["OrderID_New"].Value = row["OrderID"];
    pc["ProductID_New"].Value = row["ProductID"];
    pc["Quantity_New"].Value = row["Quantity"];
    pc["UnitPrice_New"].Value = row["UnitPrice"];
    pc["OrderID_Orig"].Value = row["OrderID",
                                   DataRowVersion.Original];
    pc["ProductID_Orig"].Value = row["ProductID",
                                    DataRowVersion.Original];
    pc["Quantity_Orig"].Value = row["Quantity",
                                   DataRowVersion.Original];
    pc["UnitPrice_Orig"].Value = row["UnitPrice",
                                    DataRowVersion.Original];
    return cmd.ExecuteNonQuery();
}

static int SubmitInsert(DataRow row, OleDbCommand cmd)
{
    OleDbParameterCollection pc = cmd.Parameters;
    pc["OrderID"].Value = row["OrderID"];
    pc["ProductID"].Value = row["ProductID"];
    pc[Quantity"].Value = row["Quantity"];
    pc["UnitPrice"].Value = row["UnitPrice"];
    return cmd.ExecuteNonQuery();
}

static int SubmitDelete(DataRow row, OleDbCommand cmd)
{
    OleDbParameterCollection pc = cmd.Parameters;
    pc["OrderID"].Value = row["OrderID", DataRowVersion.Original];
    pc["ProductID"].Value = row["ProductID",
                                DataRowVersion.Original];
    pc["Quantity"].Value = row["Quantity", DataRowVersion.Original];
    pc["UnitPrice"].Value = row["UnitPrice",
                                DataRowVersion.Original];
```

```
    return cmd.ExecuteNonQuery();
}
```

The preceding code snippet used the *DataTable* object's *Select* method to loop through the modified rows. I had a good reason to not use a *For* or *For Each* loop to examine each item in the *DataTable* object's *Rows* collection. When you successfully submit a pending deletion and call the *AcceptChanges* method of that *DataRow*, the item is removed from its parent collection. The *Select* method returns an array of *DataRow* objects. The array essentially contains pointers to the modified rows. If we remove items from the *DataTable* object's collection of *DataRow* objects, the code will still succeed.

Now it's time to put all this code to good use.

The following code snippet fetches the details for the order into a *DataTable*, modifies the contents of the order, and submits the changes to the database. The code will demonstrate that the code from the previous snippets will successfully submit pending changes. It relies on the procedures we defined earlier in the chapter. The code also includes a procedure to display the current contents of the *DataTable*, which is used to verify that we've successfully updated the contents of the order. To ensure that you can run this code snippet more than once, the code also includes a *ResetOrder* procedure, which re-creates the original contents of the order.

Visual Basic .NET

```
Dim cn As OleDbConnection
Dim da As OleDbDataAdapter
Dim tbl As DataTable = GenTable()

Sub Main()
    Dim strConn, strSQL As String
    strConn = "Provider=SQLOLEDB;Data Source=(local)\NetSDK;" & _
              "Initial Catalog=Northwind;Trusted_Connection=Yes;"
    strSQL = "SELECT OrderID, ProductID, Quantity, UnitPrice " & _
             "FROM [Order Details] WHERE OrderID = 10503 " & _
             "ORDER BY ProductID"
    cn = New OleDbConnection(strConn)
    da = New OleDbDataAdapter(strSQL, cn)

    cn.Open()
    ResetOrder()
    da.Fill(tbl)
    DisplayOrder("Initial contents of database")
    ModifyOrder()
    DisplayOrder("Modified data in DataSet")
    SubmitChangesByHand()
    tbl.Clear()
```

```
    da.Fill(tbl)
    DisplayOrder("New contents of database")
    cn.Close()
End Sub

Private Sub ModifyOrder()
    Dim row As DataRow

    row = tbl.Rows(0)
    row.Delete()

    row = tbl.Rows(1)
    row("Quantity") = CType(row("Quantity"), Int16) * 2

    row = tbl.NewRow
    row("OrderID") = 10503
    row("ProductID") = 1
    row("Quantity") = 24
    row("UnitPrice") = 18.0
    tbl.Rows.Add(row)
End Sub

Public Sub DisplayOrder(ByVal strStatus As String)
    Dim row As DataRow
    Dim col As DataColumn
    Console.WriteLine(strStatus)
    Console.WriteLine("        OrderID      ProductID       " & _
                      "Quantity         UnitPrice")
    For Each row In tbl.Select("", "ProductID")
        For Each col In tbl.Columns
            Console.Write(vbTab & row(col) & vbTab)
        Next
        Console.WriteLine()
    Next
    Console.WriteLine()
End Sub

Private Sub ResetOrder()
    Dim strSQL As String
    Dim cmd As OleDbCommand = cn.CreateCommand()
    strSQL = "DELETE FROM [Order Details] WHERE OrderID = 10503"
    cmd.CommandText = strSQL
    cmd.ExecuteNonQuery()
    strSQL = "INSERT INTO [Order Details] " & _
             "    (OrderID, ProductID, Quantity, UnitPrice) " & _
             "    VALUES (10503, 14, 70, 23.25) "
    cmd.CommandText = strSQL
    cmd.ExecuteNonQuery()
    strSQL = "INSERT INTO [Order Details] " & _
             "    (OrderID, ProductID, Quantity, UnitPrice) " & _
             "    VALUES (10503, 65, 20, 21.05)"
```

```
    cmd.CommandText = strSQL
    cmd.ExecuteNonQuery()
End Sub

Public Function GenTable() As DataTable
    Dim tbl As New DataTable("Order Details")
    Dim col As DataColumn
    With tbl.Columns
        col = .Add("OrderID", GetType(Integer))
        col.AllowDBNull = False
        col = .Add("ProductID", GetType(Integer))
        col.AllowDBNull = False
        col = .Add("Quantity", GetType(Int16))
        col.AllowDBNull = False
        col = .Add("UnitPrice", GetType(Decimal))
        col.AllowDBNull = False
    End With
    tbl.PrimaryKey = New DataColumn() {tbl.Columns("OrderID"), _
                                       tbl.Columns("ProductID")}
    Return tbl
End Function
```

Visual C# .NET

```
static OleDbConnection cn;
static OleDbDataAdapter da;
static DataTable tbl;

static void Main(string[] args)
{
    string strConn, strSQL;
    strConn = "Provider=SQLOLEDB;Data Source=(local)\\NetSDK;" +
              "Initial Catalog=Northwind;Trusted_Connection=Yes;";
    strSQL = "SELECT OrderID, ProductID, Quantity, UnitPrice " +
             "FROM [Order Details] WHERE OrderID = 10503 " +
             "ORDER BY ProductID";
    cn = new OleDbConnection(strConn);
    da = new OleDbDataAdapter(strSQL, cn);
    tbl = GenTable();

    cn.Open();
    ResetOrder();
    da.Fill(tbl);
    DisplayOrder("Initial contents of database");
    ModifyOrder();
    DisplayOrder("Modified contents of DataSet");
    SubmitChangesByHand();
    tbl.Clear();
    da.Fill(tbl);
    DisplayOrder("New contents of database");
```

```
    cn.Close();
}

static void ModifyOrder()
{
    DataRow row;

    row = tbl.Rows[0];
    row.Delete();

    row = tbl.Rows[1];
    row["Quantity"] = (Int16) row["Quantity"] * 2;

    row = tbl.NewRow();
    row["OrderID"] = 10503;
    row["ProductID"] = 1;
    row["Quantity"] = 24;
    row["UnitPrice"] = 18.0;
    tbl.Rows.Add(row);
}

static void DisplayOrder(string strStatus)
{
    Console.WriteLine(strStatus);
    Console.WriteLine("        OrderID      ProductID       " +
                      "Quantity         UnitPrice");
    foreach(DataRow row in tbl.Select("", "ProductID"))
    {
        foreach(DataColumn col in tbl.Columns)
            Console.Write("\t" + row[col] + "\t");
        Console.WriteLine();
    }
    Console.WriteLine();
}

static void ResetOrder()
{
    string strSQL;
    OleDbCommand cmd = cn.CreateCommand();
    strSQL = "DELETE FROM [Order Details] WHERE OrderID = 10503"
    cmd.CommandText = strSQL;
    cmd.ExecuteNonQuery();
    strSQL = "INSERT INTO [Order Details] " +
             "    (OrderID, ProductID, Quantity, UnitPrice) " +
             "    VALUES (10503, 14, 70, 23.25) "
    cmd.CommandText = strSQL;
    cmd.ExecuteNonQuery();
    strSQL = "INSERT INTO [Order Details] " +
             "    (OrderID, ProductID, Quantity, UnitPrice) " +
             "    VALUES (10503, 65, 20, 21.05)";
    cmd.CommandText = strSQL;
```

```
    cmd.ExecuteNonQuery();
}

static DataTable GenTable()
{
    DataTable tbl = new DataTable("Order Details");
    DataColumn col;
    col = tbl.Columns.Add("OrderID", typeof(int));
    col.AllowDBNull = false;
    col = tbl.Columns.Add("ProductID", typeof(int));
    col.AllowDBNull = false;
    col = tbl.Columns.Add("Quantity", typeof(Int16));
    col.AllowDBNull = false;
    col = tbl.Columns.Add("UnitPrice", typeof(Decimal));
    col.AllowDBNull = false;
    tbl.PrimaryKey = new DataColumn[] {tbl.Columns["OrderID"],
                                       tbl.Columns["ProductID"]};
    return tbl;
}
```

We just wrote a huge amount of code in order to submit pending updates. The code that we used to generate the parameterized *Command* objects is specific to the initial query. The code in the *SubmitChangesByHand* procedure, however, is generic. It examines the cached changes in our *DataTable*, determines the type of change stored in each modified *DataRow*, calls a function to execute the query to submit the pending change, and then marks the *DataRow* appropriately, depending on the function's return value.

Essentially, we just re-created the updating functionality available through the *DataAdapter* object, which I'll cover next.