CHAPTER

9

Database Development with dbGo for ADO

IN THIS CHAPTER

Introduction to dbGo

This chapter will get you programming using Microsoft's ActiveX Data Objects (ADO), which are encapsulated by Delphi's dbGo for ADO components.

dbGo for ADO is represented by those components residing on the ADO tab of the Component Palette and provide data access through the ADO framework.

Overview of Microsoft's Universal Data Access Strategy

Microsoft's strategy for Universal Data Access is to provide access to a wide range of data through a single access model. This data might consist of both relational and non-relational data. Microsoft accomplishes this through the *Microsoft Data Access Components (MDAC)*, which comes installed in all Windows 2000 systems or can be downloaded from `http://www.microsoft.com/data/`.

MDAC is comprised of three elements: OLE DB, Microsoft ActiveX Data Objects (ADO), and Open Database Connectivity (ODBC).

Overview of OLE DB, ADO, and ODBC

OLE DB is a system level interface that uses COM to provide access to many sorts of data including relational and non-relational formats. It is possible to write code that directly interfaces with the OLE DB layer; although with ADO, it's much more complex and in most cases, unnecessary.

Many OLE DB providers are implementations of the OLE DB interfaces for providing access to specific vendor data. For instance, some OLE DB providers give access to data from Paradox, Oracle, Microsoft SQL Server, the Microsoft Jet Engine, and ODBC just to name a few.

ADO is the application level interface that developers use to access data. Whereas OLE DB consists of many (more than 60) different interfaces, ADO only consists of few with which developers must concern themselves. ADO actually uses OLE DB as the underlying technology for accessing data.

ODBC was the precursor to OLE DB and is still a very useful mechanism by which developers can gain access to relational, and some non-relational, data. In fact, one of the OLE DB providers goes through the ODBC layer.

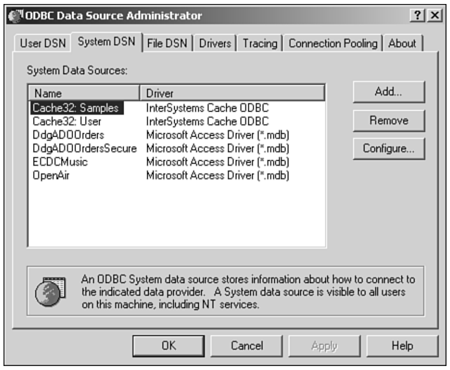
Using dbGo for ADO

dbGo for ADO is made up of the set of Delphi components that encapsulate the ADO interfaces and adapt them to the abstract way of doing database development that is common in Delphi.

The following sections will show you how to use these components. For this chapter, we will primarily use a Microsoft Access database through an ODBC provider.

Establishing an OLE DB Provider for ODBC

To establish a connection to the database, you must create an ODBC *Data Source Name (DSN)*. DSNs are similar to BDE aliases in that they allow you to provide system-level connection points with connection information for databases centrally accessible on your system. To create DSNs you must use the ODBC Administrator that ships with Windows. On Windows 2000, this is accessed via Control Panel under the Administrative Tools subdirectory. When launching this application, you'll get the dialog box shown in Figure 9.1.



Name	Driver
Cache32: Samples	InterSystems Cache ODBC
Cache32: User	InterSystems Cache ODBC
DdgADOOrders	Microsoft Access Driver (*.mdb)
DdgADOOrdersSecure	Microsoft Access Driver (*.mdb)
ECDCMusic	Microsoft Access Driver (*.mdb)
OpenAir	Microsoft Access Driver (*.mdb)

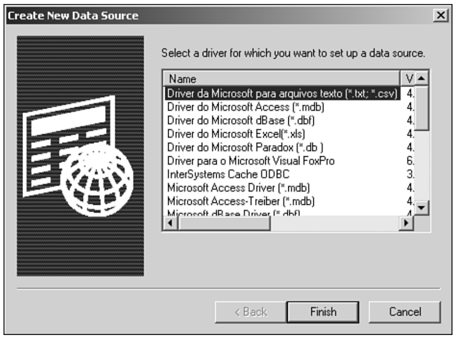
An ODBC System data source stores information about how to connect to the indicated data provider. A System data source is visible to all users on this machine, including NT services.

FIGURE 9.1
ODBC Administrator.

There are three types of DSNs:

- User DSN—User data sources are local to a computer and are accessible only when logged in as the current user.
- System DSN—System data sources are local to a computer and are accessible to any user. These are available systemwide to all users with appropriate privileges.
- File DSN—File data sources are available to all users who have the appropriate file drivers installed.

For this example, you will create a System DSN. First, launch the ODBC Administrator. Then, select the System DSN tab and click the Add button. This launches the Create New Data Source dialog box shown in Figure 9.2.



Name	V
Driver da Microsoft para arquivos texto (*.txt; *.csv)	4.
Driver do Microsoft Access (*.mdb)	4.
Driver do Microsoft dBase (*.dbf)	4.
Driver do Microsoft Excel(*.xls)	4.
Driver do Microsoft Paradox (*.db)	4.
Driver para o Microsoft Visual FoxPro	6.
InterSystems Cache ODBC	3.
Microsoft Access Driver (*.mdb)	4.
Microsoft Access-Treiber (*.mdb)	4.
Microsoft dBase Driver (*.dbf)	4.

FIGURE 9.2
The Create New Data Source dialog box.

In this dialog box, you are presented a list of available drivers. The driver you need is the Microsoft Access Driver (*.mdb). When you click Finish, you will be shown the ODBC Microsoft Access Setup dialog box (see Figure 9.3).

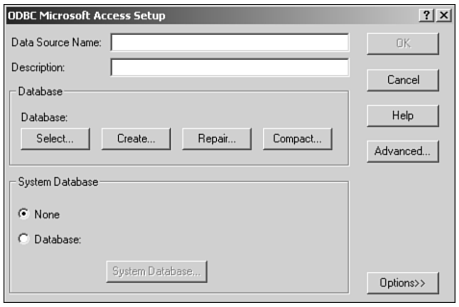


FIGURE 9.3
The ODBC Microsoft Access Setup dialog box.

Here, you must provide a DSN that will be referenced from within your Delphi application. Again, this is similar to a BDE alias. You may also provide a description if you like. Next, you must select a database by clicking Select. This will launch a File Open dialog box from which you must select a valid *.mdb file. The file that you'll use is `ddgADO.mdb` and should be installed in the `..\Delphi Developer's Guide\Data` directory where you installed the files from this book. When you click OK, your DSN will appear in the list of available System Data Sources. You can now click OK to finish working with the ODBC Administrator.

The Access Database

The database for which you just created a DSN is shown in Figure 9.4.

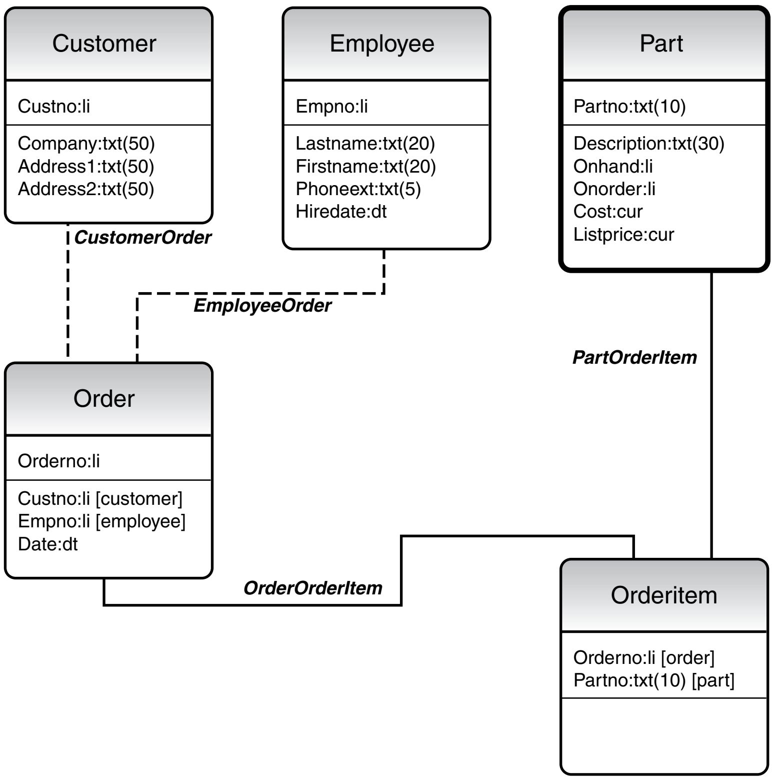


FIGURE 9.4
The sample database.

This is a simple order entry database that you'll use for the purpose of this chapter. There's nothing complicated about this database and frankly, it's not really complete. We simply put a few tables together with some meaningful relationships to show you how to use the dbGo for ADO components.

dbGo for ADO Components

All the dbGo for ADO components appear on the ADO tab of the Component Palette.

TADOConnection

`TADOConnection` encapsulates the ADO connection object. You use this component to connect to ADO provided data and through which other components hook to ADO data sources. This component is similar to the `TDatabase` component for BDE database connections. Similar to `TDatabase`, it handles functionality such as login and transactions.

Establishing a Database Connection

You can create a new application if you want or just read on to learn how to establish a database connection. You'll start with a form containing a `TADOConnection` component. You must modify the `TADOConnection.ConnectionString` property by clicking the ellipsis button on this property, which launches the `ConnectionString` Property Editor (see Figure 9.5).

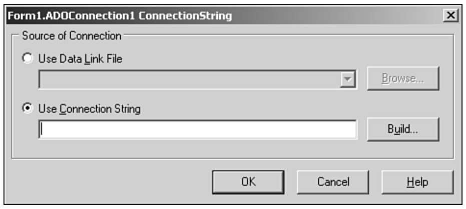


FIGURE 9.5
The `TADOConnection.ConnectionString` *Property Editor.*

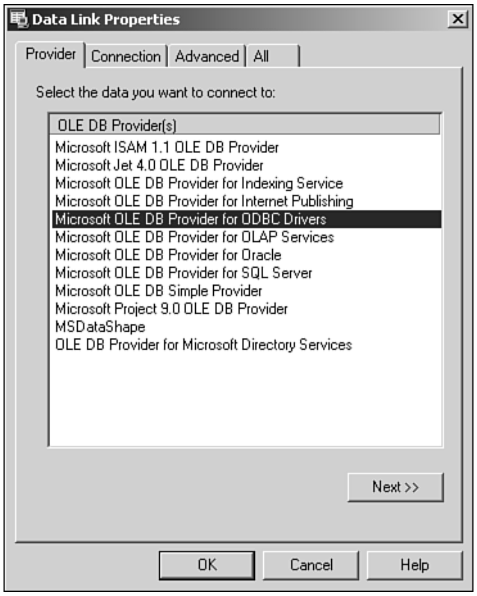
The `ConnectionString` contains one or more arguments that ADO requires to establish a connection with the database. The arguments required depend on the type of OLE DB Provider that you are using.

The `ConnectionString` Property Editor asks for the connection source from either a Data Link File (file containing the connection string) or by building the connection string, which you can later save to a file. You've already created a DSN, so you'll build a connection string that references your DSN. Click the Build button to launch the Data Link Properties dialog box (see Figure 9.6).

The first page in this dialog box allows you to select an OLE DB provider. In this case, you'll select Microsoft OLE DB Provider For ODBC Drivers as shown in Figure 9.6. Clicking the Next button takes you to the Connection Page from which you can select our DSN in the drop-down list for a Data Source Name (see Figure 9.7).

You didn't provide any security for your database, so you should be able to click Text Connection to obtain a successful connection to your database. Click OK twice to return to the main form. The connection string that results is shown here:

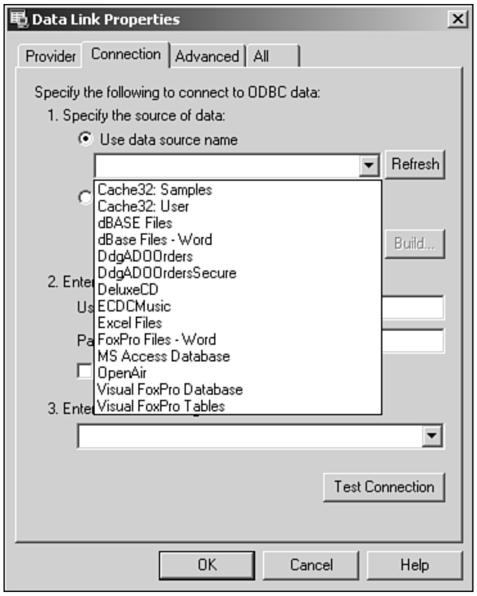
```
Provider=MSDASQL.1;Persist Security Info=False;Data Source=DdgADOOrders
```



Select the data you want to connect to:

OLE DB Provider(s)
Microsoft ISAM 1.1 OLE DB Provider
Microsoft Jet 4.0 OLE DB Provider
Microsoft OLE DB Provider for Indexing Service
Microsoft OLE DB Provider for Internet Publishing
Microsoft OLE DB Provider for ODBC Drivers
Microsoft OLE DB Provider for OLAP Services
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Simple Provider
Microsoft Project 9.0 OLE DB Provider
MSDataShape
OLE DB Provider for Microsoft Directory Services

FIGURE 9.6
The Data Link Properties dialog box.



Specify the following to connect to ODBC data:
1. Specify the source of data:
(●) Use data source name
2. Ente
Us
Pa
3. Ente

Cache32: Samples
Cache32: User
dBASE Files
dBase Files - Word
DdgADOOrders
DdgADOOrdersSecure
DeluxeCD
ECDCMusic
Excel Files
FoxPro Files - Word
MS Access Database
OpenAir
Visual FoxPro Database
Visual FoxPro Tables

FIGURE 9.7
Selecting a data source name.

Had you used a different OLE DB provider, the connection string would have been completely different. For instance, had you used the Microsoft Jet 4.0 OLE DB Provider, your connection string would be the following:

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source="C:\Program Files\Delphi '
➥Developer's Guide\Data\ddgADO.mdb";Persist Security Info=False
```

At this point, you should be able to connect to our database by setting the `TADO Connection.Connected` property to `True`. You'll be presented with a Login prompt; simply click OK to connect without entering any login information. The next section will show you how to bypass this login dialog, or to replace it with your own. The example shown here is on the CD-ROM under the `ADOConnect` directory.

Bypassing/Replacing the Login Prompt

To bypass the Login prompt, you simply have to set the `TADOConnection.LoginPrompt` property to `False`. If there are no login settings, nothing else needs to be done. However, if a username and password are required, you'll need to do some extra work.

TIP

You can test this by adding a password to the database. You can use Microsoft Access to do this; however, to add a password, you must open the database exclusively, which is a setting in the Tools, Options, Advanced Page in Microsoft Access. Otherwise, you can simply use the `ddgADOPW.mdb` file provided on the CD-ROM. The password for this database is `ddg`—go figure.

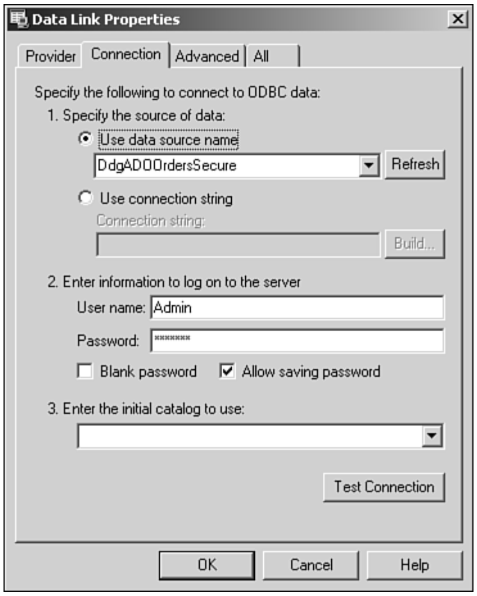
For this exercise, we've created a new DSN, `DdgADOOrdersSecure`, which refers to our database, `ddgADOPW.mdb`. If you'd like to try this example, you must create this DSN.

To bypass the login prompt on a secure database, you must provide a valid username and password in the `ConnectionString`. This can be done manually or by invoking the `ConnectionString` property editor, adding the correct username and password, and checking the Allow Saving Password check box (see Figure 9.8).

Now the `ConnectionString` appears as follows:

```
Provider=MSDASQL.1;Password=ddg;Persist Security Info=True;
➥User ID=Admin;Data Source=DdgADOOrdersSecure
```

Note the presence of the password and username (ID). Now, you should be able to set the `Connected` property to `True` while the `LoginPrompt` property is `False`.


Data Link Properties

Provider | Connection | Advanced | All

Specify the following to connect to ODBC data:

1. Specify the source of data:
   - [✓ Use data source name] DdgADOOrdersSecure
   - Use connection string
     - Connection string:
2. Enter information to log on to the server
   - User name: Admin
   - Password: *******
   - Blank password [✓ Allow saving password]
3. Enter the initial catalog to use:

FIGURE 9.8
Adding a username and password to the `ConnectionString`.

Suppose, however, that you want to provide another login dialog. In this case, you'll want to remove the password from the `ConnectionString` property and create an event handler for the `TADOConnection.OnWillConnect` event such as that shown in Listing 9.1.

LISTING 9.1 `OnWillConnect` Event Handler

```
procedure TForm1.ADOConnection1WillConnect(Connection: TADOConnection;
  var ConnectionString, UserID, Password: WideString;
  var ConnectOptions: TConnectOption; var EventStatus: TEventStatus);
var
  vUserID,
  vPassword: String;
begin
  if InputQuery('Provide User name', 'Enter User name', vUserID) then
    if InputQuery('Provide Password', 'Enter Password', vPassword) then
    begin
      UserID := vUserID;
      Password := vPassword;
    end;
end;
```

This simplified exchange represents the hand off of the username and password. A production application will likely be slightly more complex.

NOTE

It might seem that the `TADOConnection.OnLogin` event is where you would provide a username and password to stay with the `TDatabase` paradigm. However, the `TADOConnection.OnWillConnnect` event wraps the standard ADO event for this purpose. `OnLogin` is provided to be used by the `TDispatchConnection` class, which has to do with providing multitier support.

TADOCommand

The `TADOCommand` component encapsulates the ADO Command object. This component is used for executing statements that don't return resultsets such as Data Definition Language (DDL) or SQL statements. You would use this component for executing SQL statements such as `INSERT`, `DELETE`, or `UPDATE`. For instance, you'll find an example on the CD-ROM under the directory `ADOCommand`. This is a simple example that illustrates how to insert and delete a record from the employee table by using the `INSERT` and `DELETE` SQL statements. In the example, the `TADOCommand.CommandText` for the component to insert a record contains the SQL statement:

```
DELETE FROM EMPLOYEE WHERE
FirstName='Rob' AND LastName='Smith
```

The `CommandText` for the inserting `TADOCommand` component contains the SQL statement:

```
INSERT INTO EMPLOYEE (
LastName,
FirstName,
PhoneExt,
HireDate)

VALUES
(
'Smith',
 'Rob',
  '123',
  '12/28/1998')
```

To run the SQL statement, you would invoke the `TADOCommand.Execute()` method.

TADODataset

The TADODataset component retrieves data from one or more tables in a database. This component can also run SQL statements that don't return resultsets and can run user-defined stored procedures.

Much like the TADOCommand component, TADODataset can execute statements such as INSERT, DELETE, and UPDATE. However, TADODataset can also retrieve resultsets by issuing the SELECT statement. The example on the CD-ROM named ADODataset illustrates the use of the TADODataSet component. This example performs the following SELECT statement against the database:

```
SELECT * FROM Customer
```

This statement returns the entire resultset from the Customer table. You can also use SQL filtering schemes such as the WHERE clause if you need to.

In the example, we've connected a TDBNavigator component to the TADODataSet component to illustrate the ability to edit and navigate the component.

Later in this chapter, we'll further illustrate the use of TADODataSet in a sample order entry application.

BDE-Like Dataset Components

The ADO tab in the Component Palette contains three components that have been included to make transitioning from BDE applications to ADO applications easier. These components are TADOTable, TADOQuery, and TADOStoredProc. There's no reason that you can't use only the TADODataSet component when developing ADO applications. However, if it makes it easier, you can use these alternative components that are very similar to their BDE counterparts: TTable, TQuery, and TStoredProc.

TADOTable

TADOTable is a direct descendant of TCustomADODataSet. TADOTable allows you to work on a single table in the database. It operates very similar to the BDE TTable component. In fact, TADOTable adds a drop-down TableName property. Some advantages to a table type of dataset is that they support indexes. Indexes allow for sorting and quick searching. This is particularly true with non-SQL databases such as Microsoft Access. However, when using an SQL type of database, it is best to sort, filter, and so on through the SQL language. To find out more about table-type datasets, look up "Overview of ADO components" in the Delphi online help.

CAUTION

According to the Delphi online help, one of the advantages for using table-type datasets is the ease in emptying tables. The example given uses the `TCustomADODataSet.DeleteRecords()` method as the means to do this. However, a problem exists in the ADO `RecordSet` object that prevents this from working. In fact, a call to

```
TCustomADODataSet.Supports([coDelete])
```

will return `True`, yet the `DeleteRecords()` call will still fail with an exception. Therefore, to empty a table, you must use a `DELETE FROM TableName` statement, or you must loop through each record and delete it individually.

The example on the CD-ROM, `ADOTableIndex`, illustrates the use of the `TADOTable` component with an index. Additionally, it illustrates how to perform a search on the table using the `TADOTable.Locate()` function. Listing 9.2 shows partial source for this demo.

LISTING 9.2 Using the `TADOTable` Component

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i: integer;
begin
  adotblCustomer.Open;
  for i := 0 to adotblCustomer.FieldCount - 1 do
    ListBox1.Items.Add(adotblCustomer.Fields[i].FieldName);
end;

procedure TForm1.ListBox1Click(Sender: TObject);
begin
  adotblCustomer.IndexFieldNames := ListBox1.Items[ListBox1.ItemIndex];
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  adotblCustomer.Locate('Company', Edit1.Text, [loPartialKey]);
end;
```

In the `FormCreate()` event handler, you open the table and populate a `TListBox` control with all the table's field names. Then, in the `TListBox.OnClick` event handler, you set the `TADOTable.IndexFieldName` property to the field name on which we want to sort out table.

Finally, the `Button1Click()` event illustrates performing a search on the table using the `Locate()` method.

`TADOTable` is useful for those accustomed to using a `TTable` component. However, when using SQL databases, it is more efficient to use either the `TADODataSet` or `TADOQuery` components.

TADOQuery

`TADOQuery`, also a descendant of `TCustomADODataSet`, is very similar to `TADODataSet`. `TADOQuery` has a SQL property into which you would place your SQL statement. On the `TADODataSet` component, this would go in the `CommandText` property as long as `TADODataSet.CommandType` is set to `cmdText`.

We won't cover this component in great depth because most everything that applies to the `TADODataSet` component also applies to `TADOQuery`.

TADOStoredProc

The `TADOStoredProc` component allows you to use a stored procedure that exists on a database server. This is no different from using the `TADOCommand` component with its `CommandType` property set to `cmdStoredProc`. Its use is pretty much the same as `TStoredProc` discussed in Chapter 29, "Developing Client/Server Applications" of *Delphi 5 Developer's Guide*, which you'll find on the CD-ROM.

Transaction Processing

ADO supports transaction processing, and this is handled through the `TADOConnection` component. As an example, the code in Listing 9.3 is taken from our simple order entry application.

LISTING 9.3 Transaction Processing with `TADOConnection`

```
procedure TMainForm.Button1Click(Sender: TObject);
begin
  if TNewOrderForm.Execute then
  begin
    ADOConnection1.BeginTrans;
    try
      // First Create an Orders Record
      adodsOrders.Insert;
      adodsOrders.FieldByName('CustNo').Value :=
        adodsCustomer.FieldByName('CustNo').Value;
      adodsOrders.FieldByName('EmpNo').Value :=
        adodsEmployee.FieldByName('EmpNo').Value;
      adodsOrders.FieldByName('Date').Value := Date;
```

LISTING 9.3 Continued

```
      ShowMessage(IntToStr(adodsOrders.FieldByName('OrderNo').AsInteger));
      adodsOrders.Post;

      // Now create the Order Line Items.

      cdsPartList.First;
      while not cdsPartList.Eof do
      begin
        adocmdInsertOrderItem.Parameters.ParamByName('iOrderNo').Value :=
          adodsOrders.FieldByName('OrderNo').Value;
        adocmdInsertOrderItem.Parameters.ParamByName('iPartNo').Value :=
          cdsPartListPartNo.Value;
        adocmdInsertOrderItem.Execute;
        cdsPartList.Next;
      end;
      adodsOrderItemList.Requery([]);
      ADOConnection1.CommitTrans;
      cdsPartList.EmptyDataSet;
    except
      ADOConnection1.RollbackTrans;
      raise;
    end;
  end;
end;
```

The method in Listing 9.3 is responsible for creating a customer order. There are two parts to this transaction. First, the order record must be created in the Order table. Second, the order line items must be added to the OrderItem table. Because there are two table updates, it makes sense to place this into a single transaction.

Here is a skeleton of our transaction:

```
begin
    ADOConnection1.BeginTrans;
    try
      // First Create an Orders Record
      // Now create the Order Line Items.
      ADOConnection1.CommitTrans;
    except
      ADOConnection1.RollbackTrans;
      raise;
    end;
  end;
end;
```

You'll see that we encapsulate our transaction inside of a `try...except` block. `ADOConnection1.BeginTrans()` method starts the transaction. The `ADOConnection1.CommitTrans()` method commits the transaction. If there are any failures, an exception occurs and the `ADOConnection1.RollbackTrans()` method will roll back any changes that were made to any tables.

Summary

This chapter got you started working with Borland's dbGo for ADO components. These components give you the ability to use Microsoft's ADO technology for accessing both relational and non-relational data.