

Visual Basic for Testers

MARY ROMERO SWEENEY

Apress™

Visual Basic for Testers

Copyright ©2001 by Mary Romero Sweeney

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 1-893115-53-4

Printed and bound in the United States of America 12345678910

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Editorial Directors: Dan Appleman, Gary Cornell, Karen Watterson, Jason Gilmore

Technical Reviewer: Harvin Queen

Marketing Manager: Stephanie Rodriguez

Managing Editor: Grace Wong

Editor: Kiersten Burke

Production Editor: Kari Brooks

Page Composition: Diana Van Winkle, Van Winkle Design Group

Artist: Tony Jonick

Cover Designer: Karl Miyajima

Indexer: Valerie Perry

Distributed to the book trade in the United States by Springer-Verlag New York, Inc.,
175 Fifth Avenue, New York, NY, 10010

and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17,
69112 Heidelberg, Germany

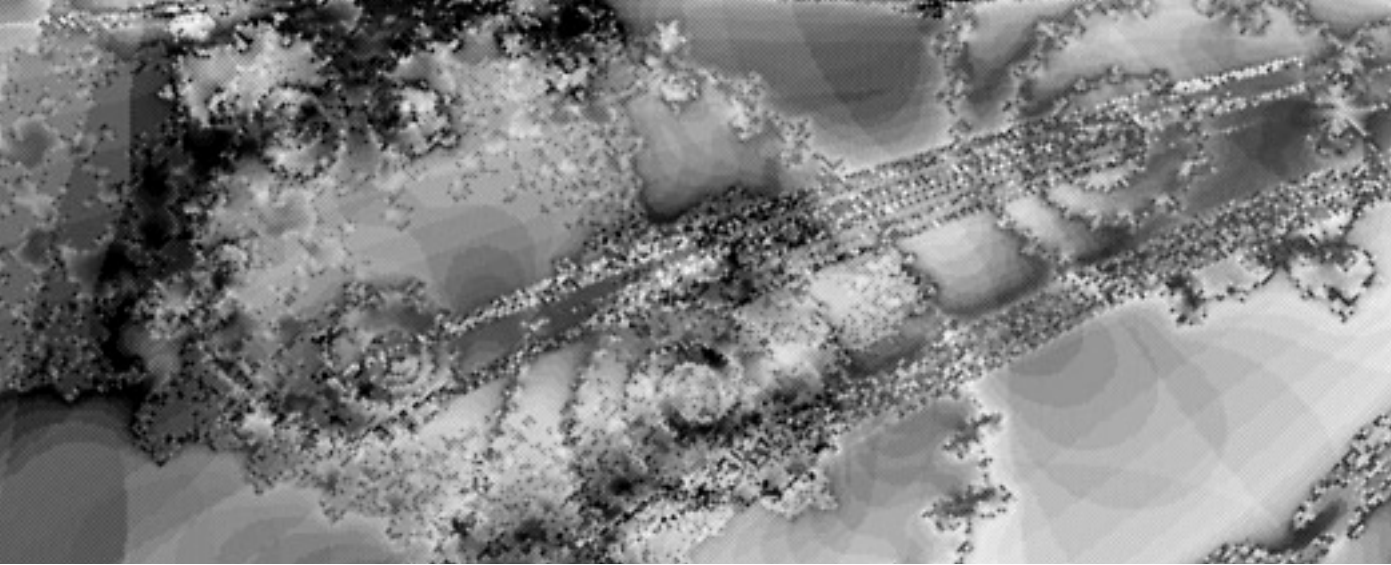
In the United States, phone 1-800-SPRINGER; orders@springer-ny.com;
<http://www.springer-ny.com>

Outside the United States, contact orders@springer.de; <http://www.springer.de>;
fax +49 6221 345229

For information on translations, please contact Apress directly at 901 Grayson Street,
Suite 204, Berkeley, CA, 94710

Phone: 510-549-5938; Fax: 510-549-5939; info@apress.com; <http://www.apress.com>

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.



Part Four

Advanced Topics in Automated Test Scripting

Introduction to Database Testing

The testing of software applications usually includes accessing and verifying data of some kind. This is true of any kind of software application you can think of these days, including, of course, Web applications. In fact, out of necessity, more focus is being placed these days on end-to-end testing of large software applications. *End-to-end testing* traces the flow of information and any bugs encountered from the user of the system, the client, all the way through to any data accessed and then back again to the original client. Going through the entire system may include passing through multiple servers and accessing heterogeneous data stores. For example, a client system such as a browser on a home computer accesses an application stored on a Web server. This Web server, in turn, passes the client's request for information—say, a price on a product—to a database server. The database server returns the request back to the Web server, which, in turn, passes the information back to the client. Testing this kind of arrangement can be complex as the tester tries to determine the source of bugs in the system's multiple layers. It is important to be able to understand and work with of all types of data to be effective at end-to-end testing. This data can be stored in many ways—for example, spreadsheets, text files, and databases. Relational database management systems (DBMS) such as Oracle, SQL Server, Informix, Sybase, etc. are used to store data for large, client-server type systems. However, many applications include data from older, nonrelational database systems. Because of this exceedingly wide field of possible data sources, we will have to limit the focus in this chapter to data stored in ODBC-compliant databases.

Understanding data involves more than can be presented in one book. To be effective at database application testing, you will also need some database background—in other words, a thorough knowledge of database design and Structured Query Language (SQL) as well as practical training and experience with a database management system (DBMS).

Visual Basic can be a very functional means to access and verify data in an ODBC-compliant relational database in several ways. First, Visual Basic contains a number of useful tools to reference and view a database and even modify its structure and data. It can also be used to programmatically access data using a

variety of data access methods. In this chapter, we will start by exploring the use of the Visual Database tools for data access and then use ADO (ActiveX Data Objects) programming to manipulate data in a SQL Server database.



NOTE See the “ODBC and OLEDB” and “SQL” sidebars in Chapter 2.

Objectives

By the end of this chapter, you will be able to:

- Use the Visual Database tools to access database components.
- Use the Visual Database tools’ Query Builder window to execute some queries useful for testing.
- Use ADO code to open and access a database.
- Access the SQL Server SQL-DMO (Distributed Management Objects) library.

Database Application Testing Using the Visual Database Tools

Visual Basic 6 can be used to support database testing both with and without doing a lot coding. The tools that don’t involve a lot of coding include the Data Form Wizard and the Visual Database tools. In Chapter 2, you learned how to use the Data Form Wizard to create a quick front end for a database. In this section, you will learn how to use the Visual Database Tools to access many types of databases and even to modify SQL Server databases. This will afford you a common way to access heterogeneous data so you can examine the state of the data and execute and test queries against it.



NOTE The Visual Database tools available in the Visual Basic 6 Enterprise edition are also available in Visual InterDev, Microsoft Visual J++, and Microsoft Visual C++ 6 Enterprise editions.

The Visual Database tools in Visual Basic are comprised of three major components: the Data View window, the Query builder, and the Data Environment designer.

Using the Data View Window

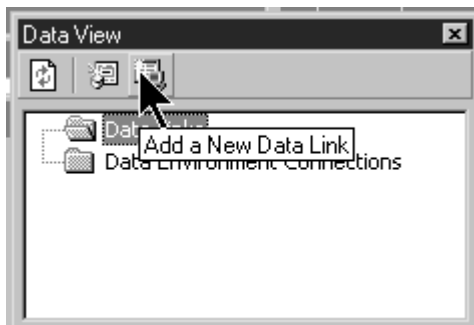
Creating a data link in the Data View window is an easy way to quickly set up a connection to a database. Once the database is open, you will be able to retrieve database objects like tables and views. This will give you a look at the structure of the database so that you can verify the presence of those same database objects. You will also be able to inspect the data and perform queries as mentioned earlier.

TO TRY THIS

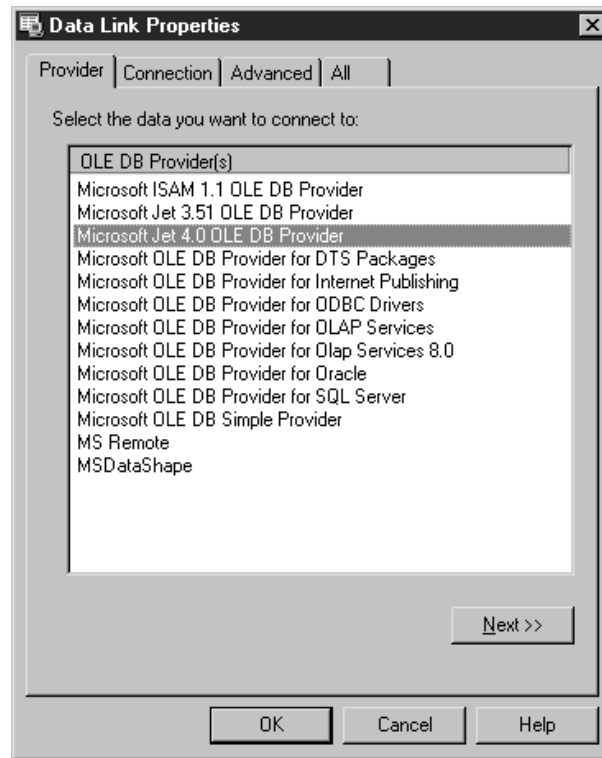
1. Select the **View > Data View Window** menu item or you can click the **Data View window** icon from the Standard toolbar.



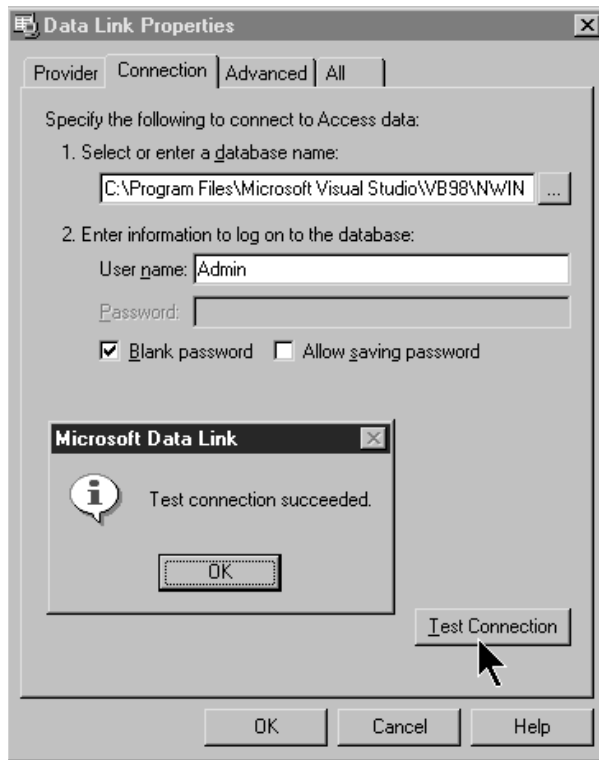
2. Click on the **Add a New Data Link** icon within the Data View window.



The **Data Link Properties** window appears with the **Data** tab displayed. From this window, you can select the OLE DB data provider for many different kinds of databases including Oracle and SQL Server. The most general one is the OLE DB data provider for ODBC databases, which will allow you to connect any ODBC database. (See Chapter 2 for a discussion of ODBC and OLE DB.)

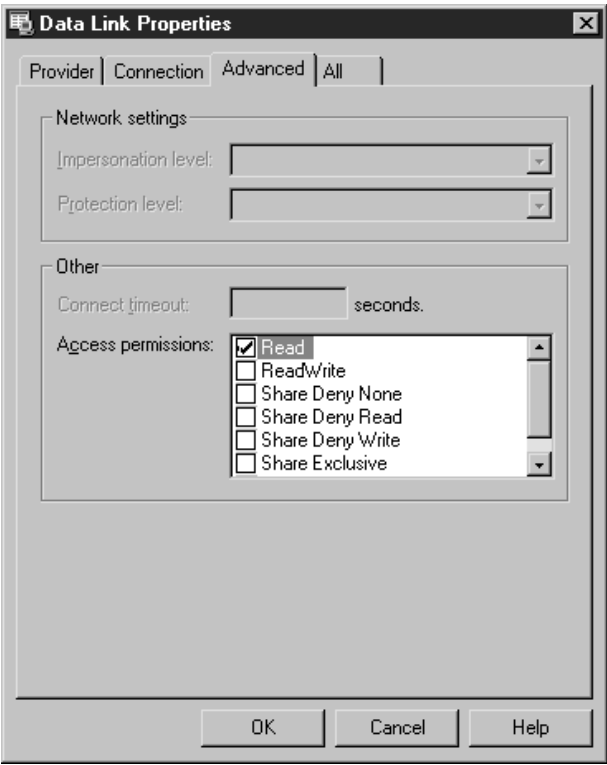


3. The Jet database providers allow us to connect to a Microsoft Access database. That is what we will select for this example since Visual Basic installs sample databases of this type (as long as you have selected that option when you installed Visual Basic). Select **Microsoft Jet 4.0 OLE DB provider**, then click **Next>>**. This takes you to the **Connection** tab of the same window (so you could have just clicked on the Connection tab also).
4. From the Connection tab, browse to the database you want to investigate. This part of the dialog will look different depending on what you selected in the previous Data tab. Since you selected a Jet provider, you are prompted only for an Admin account and password. By default, Access databases have an Admin account with no password so you can usually just specify the default here. For this example, you will link to the sample Northwind database located in the following file when Visual Basic installs: **C:\Program Files\Microsoft Visual Studio\VB98\NWIND.MDB**. You can browse to this file or type it directly.

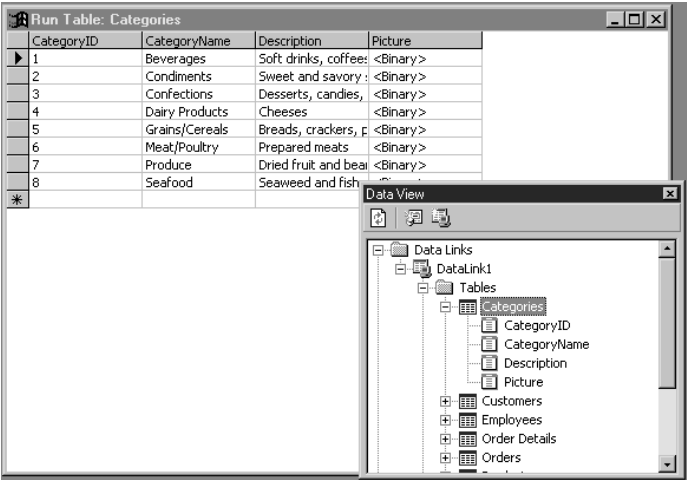


TIP You can test the connection to the databases by clicking the Test Connection button on this same dialog. If the connection is successful, it will display a Message box indicating success. If not, you will have to check with your system administrator to determine the correct database and logon specifications required.

5. From here, you can simply click **OK** and continue; however, there is one more important point to discuss on the **Advanced** tab of this dialog. By default, the Microsoft Access permissions are set to **Share Deny None** on this tab. This means that neither read nor write access can be denied to others and you also have read and write access to the database (as long as the username you specified in step 3 also has that capability). When testing, you don't usually want to modify data unless you are specifically adding test data to do so. My recommendation, in most cases, is to set your access to **Read** access on a production database. This will not allow you to change database values. If you change this value now, however, you won't be able to change data—which we are going to do in a later task. So, for now, you can leave the default access, **Share Deny None**. Click **OK** to close the dialog.



6. In the Data View window, expand the **Data Links** folder, expand the data link you just created, and then expand the **Tables** folder. You can now expand any table to see its list of fields (columns). Double-clicking the table will open a new window, the **Run table** window (which is referred to as the Query Builder window in Help but does not contain that name in its caption), and display the current contents of the table.





NOTE Double-clicking a table opens the whole table. For a sample database, this action is okay; however, when accessing a very large table, this operation may take quite awhile. There are other kinds of queries that will return useful information but don't return all of the rows in the table. We will explore other queries practical to testing next.

Using the Query Builder Window to Execute Database Queries

Creating a data link and investigating the database structure and contents as we have just done is a valuable first step in database access. You can follow pretty much all of the same steps as in the last section to access any ODBC database. This gives you a common way to access these databases so it isn't always necessary to learn each one of their individual DBMS software.

It is even more valuable to be able to execute queries against the database. The Run Table window that displays when double-clicking on a table in the Data Link window is really the Query Builder window cleverly disguised! Adding more panes to this same window will allow you to create and execute queries within it. There are quite a few valuable SQL queries for testing that will return information about the database. The next steps demonstrate how to use the Query Builder window to do just that using the data link created in the previous section.

TO TRY THIS

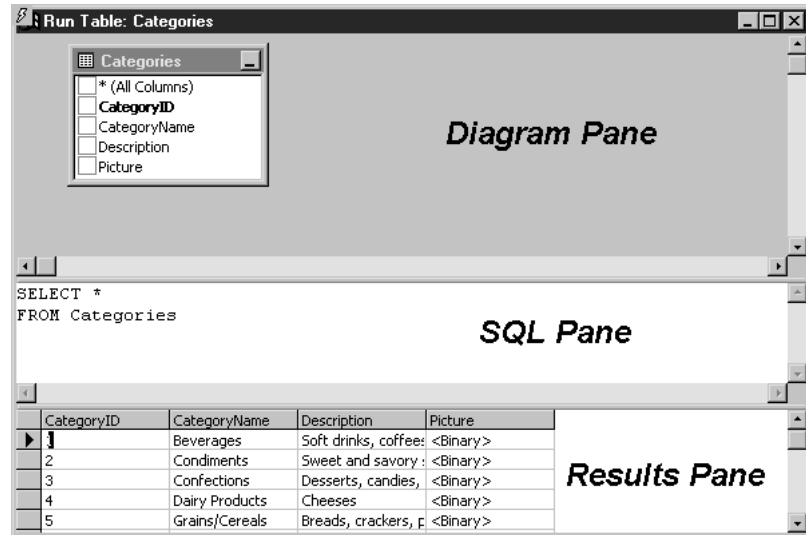
1. Make sure you have created the data link from the last section, "Using the Data View Window." Then click on the caption of the **Run Table** window (Query Builder) to make sure it has the focus.



NOTE The Run Table window pops up by double-clicking on a table in the Data View window. If you closed the Run Table window earlier, simply click on any table from the Data View window.

2. Select the **View > Show Panes** menu item and select the **Diagram** and **SQL** panes. Now the Query Builder window is set to build and run SQL

statements. The Diagram pane of the Query Builder window displays the tables in the query and, if there is more than one table, any relationships between them. The SQL pane shows the current SQL statement. It can be modified to any valid SQL statement.



3. Modify the SQL statement in the SQL pane so that it reads:

Select Count(*) from Categories.

This statement will return the number of rows in the Categories table.

4. Right-click in the **Diagram** pane (or the **Results** pane) and select **Run** from the pop-up menu. The Results pane will show the answer to the query.

There are a number of SQL statements that are valuable for testing. The following are just a sample using the data in the Northwind database (to try them, repeat steps 3 and 4 from the preceding task).

- To return the most recently ordered items from the Orders table:

```
SELECT *
FROM Orders
WHERE orderdate =
      (SELECT MAX(orderdate)
       FROM orders)
```

- To find records with duplicate primary keys in a table (there are no such duplicate records in Northwind tables. However, this is a good check for other databases where referential integrity of the data is suspect):

```
SELECT employeeid
FROM employees
GROUP BY employeeid
HAVING COUNT(employeeid) > 1
```

- To find orphan records, that is, Orders that have no Employee assigned (again, this result should yield zero rows [no data] in the Northwind database):

```
SELECT e.EmployeeID, o.Orderid
FROM employees e RIGHT OUTER JOIN
      orders o ON e.employeeid = o.employeeid
WHERE e.employeeid IS NULL
```

- You can also drag tables and views from the Data View window and drop them onto the Diagram pane of the Query Builder window. This is a quick way to run predefined queries without knowing a lot of SQL. To give this a try, place your cursor on any table in the Data View window, click it once, drag it just over the Diagram pane of the Query Builder window, and let go. From there, you can modify the SQL statement if desired.

To get the most out of these tools, you should learn more about SQL. There are many excellent books on SQL, for further information on learning this language, see Appendix A: Resources and References.

Relational Database Objects Primer

If you are unfamiliar with relational databases, you will need to get up to speed on the basics before doing any significant amount of testing. There are courses available at community colleges and many good books that can help you get up to speed; Appendix A of this book has some good resources.

To get you started, here is a description of some of the major objects in a relational database:

Tables: All data in relational databases is stored in table format. The rows of the table represent one record's worth of data. For example, each row in a Customers table would contain information about a single customer. The columns of the table represent individual pieces of data about the customer, such as the customer's name, address, and so on.

Views: A view is an alternate way to look at data from one or more tables in the database. A view's contents are generated by a query and usually contain a subset of columns from one or more tables. A view is considered a virtual table because it can be treated as though it were a table even though it isn't. For example, a view can store the SQL code to find all customers in the customer's table who live in Washington. That information actually resides in the customer's table but since it is defined in a view, we can look at it in the view as though it is a table of its own. So, a view is really just a query that is stored and has a name.

Stored Procedures: A stored procedure allows the database programmer to write a set of SQL statements and give them a name so that they can be used over and over without having to rewrite them each time they are needed. Stored procedures can contain most any SQL statement and can be used to perform simple or complex database tasks.

The Data Environment Designer

You can do a lot of data interrogation using just the Query Builder window—it's quick and easy. The Visual Database tools also provide an alternate way to access databases visually through the use of the Data Environment designer. It is set up in much the same way but with some extra steps. The value of these extra steps is that unlike the data link, you get a Connection object that you can refer to in your Visual Basic code. You can programmatically manipulate the Connection object to perform database tasks. You can also create Command objects to attach to a connection and drag and drop those Command objects to create forms based on the data. Although these activities might be valuable in testing, they are, of course, largely useful for application software development. There are a number of ways to access databases programmatically in Visual Basic. To cover all of them may be confusing so I will focus on methods that are either simple or very powerful. The data link is a very simple method that does not require a lot of code. To perform programmatic access, I will use the ADO object library commands since this library provides great flexibility and power.

Testing Databases Using ActiveX Data Objects (ADO)

There is a real alphabet soup of methods to access databases in code. Microsoft started out with DAO (Data Access Objects), which is built primarily to access Microsoft's proprietary Jet database engine used by Microsoft Access. RDO

(Remote Data Objects) had, for a long time, been the method to access data in a client-server system. RDO provides commands to access multiple types of databases since it is really just a wrapper around the ODBC API. (I warned you this was an alphabet soup!)

ODBC was built to accommodate databases of varying types. There has been so much proprietary database access that businesses found it difficult to access all the data they needed in a single program. ODBC answered this by creating drivers for databases that allow the access of data in a common way. (See Chapter 2 for a discussion of ODBC.)

Since Microsoft had DAO and RDO working just great, why did they come up with something new like ADO? Actually, ADO itself is a wrapper around OLE DB (see Chapter 2). ODBC (and its wrapper RDO) only allows for data access across Windows databases. This does not address the need to access heterogeneous data across multiple platforms such as UNIX-based systems and other non-Windows operating systems. The intent of OLE DB is to allow data access across even these varying operating systems. So, this is effectively a step beyond ODBC towards multiple-platform, heterogeneous database systems.

For testers, ADO is a good choice for data access since it allows us to learn a single method to access a variety of databases.

With all of these acronyms, it's easy to get confused. The chart in Figure 8-1 summarizes the common Windows data access methods and how ADO compares to them.

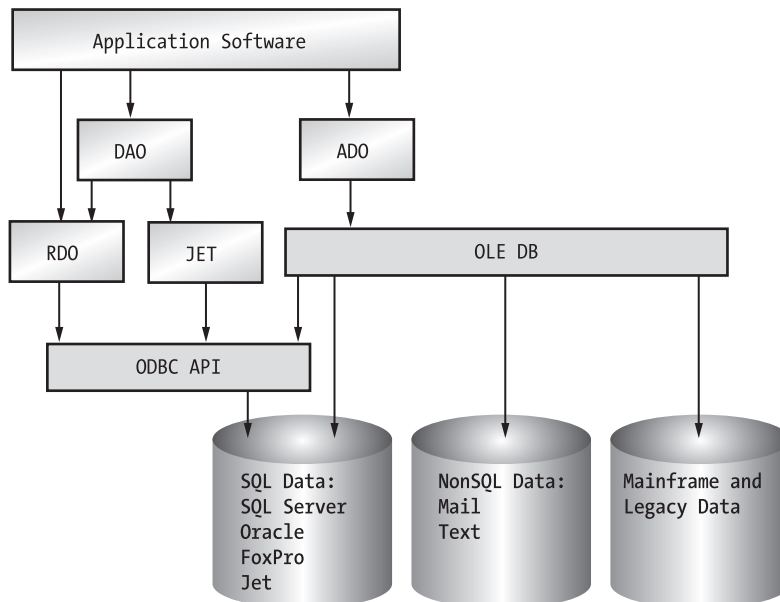


Figure 8-1. Comparison of Data Access methods for Windows programming.

Understanding ADO Architecture Basics

The ADO programming model has two important objects necessary to open a connection to a database and obtain a set of records: the Connection object and the Recordset object. ADO does have other objects, such as an optional Command object and an Error object. However, the only two that are really necessary to make a connection are the Connection and Recordset objects. The interesting thing about them is that they are not related hierarchically. You can have a Connection object that sets up a connection to a database, then creates sets of data—that is, Recordset objects—and attaches them to this connection. However, you can also just create a Recordset object and set it up with a connection when you create it. You can set it again at a later time to attach to a different connection. This allows for great flexibility in programming. If you find this confusing, don't worry, you can set up a connection to a database and then obtain a recordset from it very easily. You don't have to worry about disconnecting them unless you get to a point in programming in which you want to do so.

It's usually best to start with an example, but first, we will need to do some set up. To use the ADO library, you must set a reference to the Microsoft ActiveX Data Objects Library by selecting the **Project > References** menu item and place a check in its box in the Project References dialog (Figure 8-2).



NOTE Figure 8-2 shows a check in the box of the Microsoft ActiveX Data Objects 2.6 Library. Versions of the library available to you may vary depending on what is installed on your machine. Check the box with the highest-level version you have available. Earlier versions are there for backwards compatibility with applications that may use them.

Once you have set the reference, you can view the ADO objects and their corresponding properties and methods (similar to what we did with the Microsoft Scripting Runtime library in Chapter 5) in the Object Browser (Figure 8-3).



NOTE Press F2 in Design mode to get to the Object Browser, or View > Object Browser, or click its icon on the Standard toolbar.

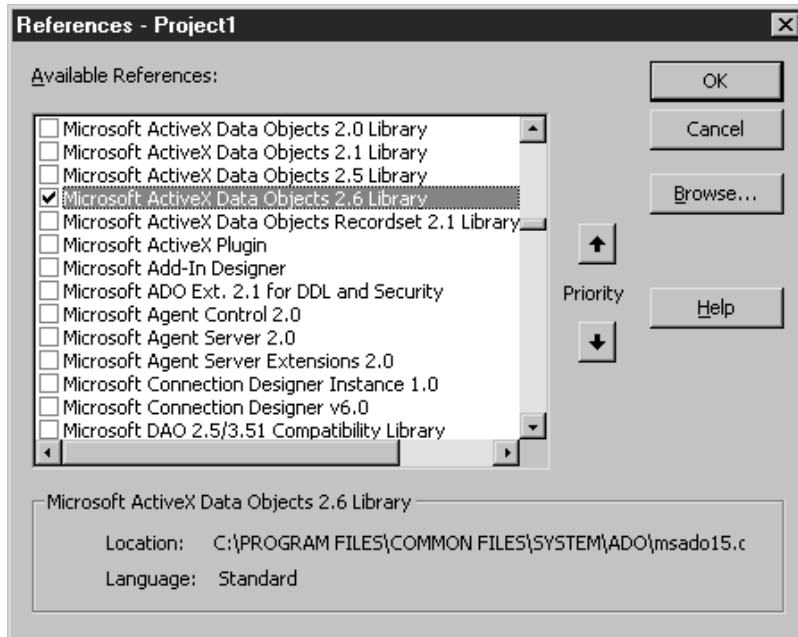


Figure 8-2. Setting a reference to the ADO library using the Project References dialog.

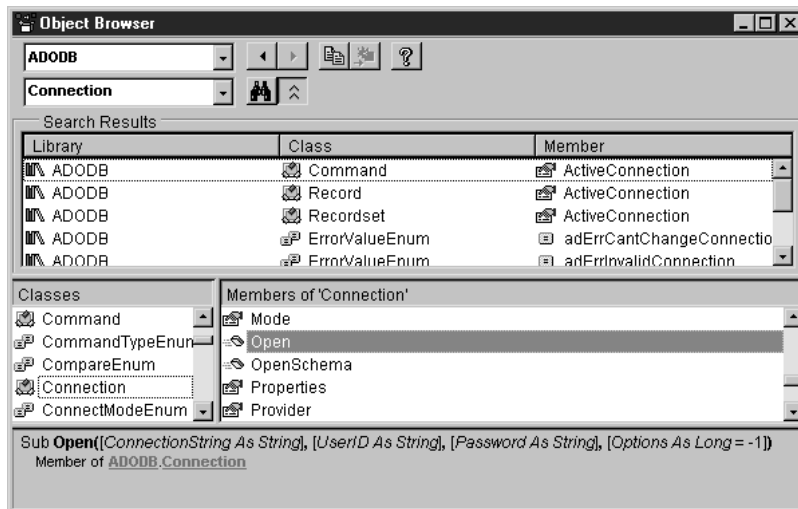


Figure 8-3. The ADO library.

Using the Connection Object

The Connection object is used to set up the information necessary to attach to a database. It has a number of properties used to specify this information, such as username and password, as well as, of course, the location of the database. The Open method of the Connection object is used to actually create the connection. There are two ways to connect to the database. One is to use a Data Source Name (DSN) and the other is to directly specify the database using a DSN-less connection.



NOTE *The Connection object has other uses as well. In addition to opening a connection, it can be used to execute SQL statements by using its Execute method. Once the connection is created, set, and open, you can specify a SQL statement to execute such as:*

```
cnn.execute _
    "Insert into categories (CategoryName) values ('Tester Products')"
```

Creating a Data Source Name (DSN)

Think of a Data Source Name as a kind of alias to a database. You create a DSN to preset the logon information for the database you want to access. You can include password and other access information in the DSN and give it a name of your choice. Then, in your Visual Basic code, you can reference this name and use it to logon to the database. Data Source Names can be created from the Control Panel. For Windows 95/98 systems, set up a DSN in the ODBC data sources area. When you double-click there, the dialogs do a good job of explaining how to set up the DSN. You will need to have an appropriate logon for the database as provided by your system administrator.

To create a DSN for a database:

1. Open your computer's Control Panel starting from the Windows TaskBar by selecting **Start > Settings > Control Panel**. In Windows 95/98, double-click on the **ODBC Data Sources** icon. In Windows 2000, select **Administrative Tools** and then **Data Sources (ODBC)**.
2. There are three different kinds of DSNs to choose from: System, File, or User DSNs, depending on the visibility you need for this database. The System DSN is the most global; the User DSN is the least global. The dialog explains the options; choose whichever is most appropriate. If you can't decide, select **System**, then click **Add**. A list of database drivers will display in the next dialog.

3. Select a driver for the kind of database you want to use such as Oracle, SQL Server, or Access and click **Finish**. You will then be prompted to create a name for the DSN. You can make that up but it should be something you will remember such as NWindDSN.
4. If you have selected the Microsoft Access driver, you will need to browse to find the Access .mdb file. Other databases drivers will walk you through a series of forms you will need to examine to enter logon information for the database.

If you have set up the DSN correctly, you will see it appear in the list. You can then use the DSN in your code to simplify calls to the database. It is possible to create a DSN-less connection, however. We will explore both ways.

Accessing a Database Using a DSN

To open an ODBC-compliant database that has an available DSN, the connection can be set with the following two lines:

```
Dim MyConn As New ADODB.Connection
MyConn.Open "YourDSN", "Admin", ""
```

In this code, MyConn is created as an object variable of the Connection class. Then, the Open method of this new object is used to connect to the database. The first argument of the Open method, YourDSN, refers to a DSN that can be created through the Control Panel's ODBC settings (see the previous sidebar, "Creating a Data Source Name"). Usually, the developer will have already created this DSN so you will simply need to find out what it is. The second argument of the Connection object's Open method is where you specify the username to connect to the database—in this case, Admin. The third argument is the password to log onto this database; in this case, there is no password for the Admin account. (That probably won't always be the case!)

Accessing a Database without a DSN

It is possible in ADO to create a DSN-less connection by specifying all of the connection criteria when you set the Connection object. For example, the following code demonstrates opening a connection to the SQL Server sample database,

Northwind, using a DSN-less connection, and logging onto the SQL Server sa (system administrator) account, assuming there is no password:

```
Dim MyConn As New ADODB.Connection
MyConn.Open "Driver={SQL Server};Server=;Database=Northwind;UID=sa;PWD="
```

In order for this to work, you must have Microsoft SQL Server installed because when the Server argument is left blank as shown, the default is to look for a local server—in other words, a server on your machine. Otherwise, you must specify the name of a server you can access. In that case, you don't need SQL Server installed on your machine but you must have access to a SQL Server database installed on the server you are connecting to. If you want to use a trusted connection to SQL Server, which means logging on using your NT or Windows 2000 username and password, then you leave both the UID (user identification) and the PWD (password) arguments unset.



NOTE *Your SQL Server database administrator determines whether you can log onto SQL Server with a trusted connection or with a SQL Server connection (called **SQL Server authentication**).*

Once the connection is established, our next step is to retrieve a set of records for testing.

Using the Recordset Object

The Recordset object can be used to issue a SQL statement to retrieve exactly what it sounds like: a set of records. Once you have a set of records, you can think of them like a card file of index cards: you can process them one-by-one and look up information in them. To get this set of records, we must first create the Recordset object variable and set it equal to a new, empty recordset:

```
Dim MyRs As ADODB.Recordset
Set MyRs = New ADODB.Recordset
```

Before retrieving the actual records for the recordset, you need to specify where the records will be processed—either on the client or the server. If you specify that you want to process on the client, you can save server resources so, in general, you will usually choose client-side processing. To set this, use the `CursorLocation` property:

```
MyRs.CursorLocation = adUseClient
```

You can also specify a cursor type. A *cursor* is essentially a pointer that points to the current record in a recordset. There are different kinds of cursors and depending on which you choose, you can specify how the data is retrieved. For example, you can choose a cursor that will allow you to see changes made by others, or not. Choices for cursors are:

Static—A fast cursor because you get a snapshot of the data as it exists at the moment you capture it. You will not get to see any additions or deletions to the data that other users may be making while you work.

Forward Only—This is the fastest cursor because it is a static cursor that can only go forward through the recordset. This cursor is appropriate for writing code to generate a report.

Dynamic—The slowest but most powerful cursor. You can move in any direction in the recordset and see all changes, additions, and deletions.

Keyset—Just like a dynamic cursor, you can move any direction in the recordset and see modifications made by other users to a particular record. However, you won't be able to see the addition of new rows to the data.

The cursor type will determine our ability to modify or view the data. For testing, you will be viewing data largely for verification rather than modification purposes; this allows you to choose a fast cursor. The static cursor is fast, even though it doesn't allow for viewing modifications performed by others. The ADO library provides four constants to access the four types of cursors: `adOpenStatic`, `adOpenDynamic`, `adOpenForwardOnly`, and `adOpenKeyset`. The following code sets the `CursorType` property of the `MyRs` recordset to a static cursor:

```
MyRs.CursorType = adOpenStatic
```

Now we can open the recordset. We use a SQL statement to determine the records to view. This SQL statement will retrieve all of the records from the Customers table in the database:

```
MyRs.Open "Select * from Customers", MyConn
```

Notice that the Connection object is an argument for the recordset's Open method. This is how you connect the Recordset object to the connection. Now we have a set of records to work with and from here on, we will use the MyRs Recordset object properties and methods to access this set of records. The next code sample shows a few of the properties and methods we can use to work with our recordset:

```
MyRs.MoveFirst 'moves to the first record
MyRs.MoveNext 'moves to the next record
Debug.print MyRs.RecordCount 'returns the # of records In the recordset:
Debug.print MyRs!Fieldname 'Will return the value of a field in the recordset
If MyRs.EOF then 'determine whether the cursor is at the end of the file
    MyRs.Close 'close the recordset
Endif
```

The syntax for all of these properties and methods can be found by exploring the ADO type library in the Object Browser.

Now you can write code to test the database. Suppose that one test requirement is to determine that a certain number of rows exist in a table within the database. The code in Listing 8-1 will open the database, count the number of rows in a table, and determine whether or not the actual number of rows found is equivalent to what is expected.



NOTE The code in Listing 8-1 presumes the existence of the LogUtil.bas module. The code will not compile correctly without it. The full text of this project can be viewed and run from following file in the Practice files: Chapter8\Demos\ADORecordVerif.vbp.

Listing 8-1. Testing the Northwind sample database by verifying row count in the Customers table.

```

Option Explicit
'*****
'* Northwind Test.
'* Verify that the expected number of records
'* in the Customers table, matches the actual number of records.
'* Dependencies: LogUtil.bas must be available
'* References: This project sets a reference to the Microsoft SQL-DMO object
'* library. It also sets a reference to the Microsoft scripting
'* run time library in order to perform the logging routines from
'* the logutil module.
'*****

Private Sub cmdShowResults_Click()
    ReadLog
End Sub

Private Sub Form_Load()
    Dim MyConn As New ADODB.Connection
    'MyConn.Open "NWindDSN", "sa", "" 'This line uses a DSN. This DSN must
    'be preset up to access the
    'Northwind sample Database on any SQL Server
    MyConn.Open "Driver={SQL Server};Server=;Database=Northwind;UID=sa;PWD="
    'The line above uses a DSN-less connection and
    'presumes you have a local server with SQL Server installed
    'If you can connect to a remote SQL Server, place its name after
    'the 'Server=' argument
    Dim MyRs As ADODB.Recordset
    Set MyRs = New ADODB.Recordset
    MyRs.CursorType = adOpenStatic
    MyRs.CursorLocation = adUseClient
    Const iEXPECTED As Integer = 91 'set number of expected items

    LogUtil.Appname = "Northwind DB Test" 'set Public application variable
    MyRs.Open "Select * from customers", MyConn
    If (Not (MyRs Is Nothing)) Then
        ' empty recordset?
        If (Not MyRs.EOF) Then
            MyRs.MoveFirst
        End If
    End If

```

```

' verify results
If MyRs.recordcount <> iEXPECTED Then
    LogToFile "****Test Failed. Actual records: " & MyRs.recordcount & _
        "; Expected records: " & iEXPECTED
Else
    LogToFile _
        "Test Passed. Actual records: " & MyRs.recordcount & _
        "; Expected records: " & iEXPECTED
End If
End If
End Sub

```

Notice that the logging of test results in Listing 8-1 is accomplished through the use of the Logging utilities module created back in Chapter 5. You can try this example by running the **ADORecordVerif.vbp** project file from the **Chapter8\Demos** folder.

Revisiting the ODBC Logon Form Template

In Chapter 2, I discussed how the many form templates and wizards provided by Visual Basic can be great learning tools. Now that you know a bit about accessing the ADO library and calling the Windows API routines (from Chapter 7), the code generated when you create a new form in a project from the ODBC Logon form template should be more intelligible. Listing 8-2 displays the code behind an ODBC Logon form.

Listing 8-2. The code generated when you add an ODBC Logon form from the template to your Visual Basic project.

```

Option Explicit
Private Declare Function SQLDataSources Lib "ODBC32.DLL" _
    (ByVal henv&, ByVal fDirection%, ByVal szDSN$, _
    ByVal cbDSNMax%, pcbDSN%, ByVal szDescription$, _
    ByVal cbDescriptionMax%, pcbDescription%) As Integer
Private Declare Function SQLAllocEnv% Lib "ODBC32.DLL" (env&)
Const SQL_SUCCESS As Long = 0
Const SQL_FETCH_NEXT As Long = 1

Private Sub Form_Load()
    GetDSNsAndDrivers
End Sub

Private Sub cmdCancel_Click()

```



```

Unload Me
End Sub

Private Sub cmdOK_Click()
    Dim sConnect As String
    Dim sADOConnect As String
    Dim sDAOConnect As String
    Dim sDSN As String

    If cboDSNList.ListIndex > 0 Then
        sDSN = "DSN=" & cboDSNList.Text & ";"
    Else
        sConnect = sConnect & "Driver=" & cboDrivers.Text & ";"
        sConnect = sConnect & "Server=" & txtServer.Text & ";"
    End If

    sConnect = sConnect & "UID=" & txtUID.Text & ";"
    sConnect = sConnect & "PWD=" & txtPWD.Text & ";"

    If Len(txtDatabase.Text) > 0 Then
        sConnect = sConnect & "Database=" & txtDatabase.Text & ";"
    End If

    sADOConnect = "PROVIDER=MSDASQL;" & sDSN & sConnect
    sDAOConnect = "ODBC;" & sDSN & sConnect

    MsgBox _
        "To open an ADO Connection, use:" & vbCrLf & _
        "    & "Set gConnection = New Connection" & vbCrLf & _
        "gConnection.Open """" & sADOConnect & """" & vbCrLf & vbCrLf & _
        "To open a DAO database object, use:" & vbCrLf & _
        "Set gDatabase = OpenDatabase(vbNullString, 0, 0, sDAOConnect)" & vbCrLf & _
        "Or to open an RDO Connection, use:" & vbCrLf & _
        "Set gRDOConnection = " & _
        "rdoEnvironments(0).OpenConnection(sDSN, rdDriverNoPrompt, 0, sConnect)"

    'ADO:
    'Set gConnection = New Connection
    'gConnection.Open sADOConnect
    'DAO:
    'Set gDatabase = OpenDatabase(vbNullString, 0, 0, sDAOConnect)
    'RDO:
    'Set gRDOConnection = _
        rdoEnvironments(0).OpenConnection(sDSN, rdDriverNoPrompt, 0, sConnect)
End Sub

```

```

Private Sub cboDSNList_Click()
    On Error Resume Next
    If cboDSNList.Text = "(None)" Then
        txtServer.Enabled = True
        cboDrivers.Enabled = True
    Else
        txtServer.Enabled = False
        cboDrivers.Enabled = False
    End If
End Sub

Sub GetDSNsAndDrivers()
    Dim i As Integer
    Dim sDSNItem As String * 1024
    Dim sDRVItem As String * 1024
    Dim sDSN As String
    Dim sDRV As String
    Dim iDSNLen As Integer
    Dim iDRVLen As Integer
    Dim lHenv As Long          'handle to the environment

    On Error Resume Next
    cboDSNList.AddItem "(None)"

    'get the DSNs
    If SQLAllocEnv(lHenv) <> -1 Then
        Do Until i <> SQL_SUCCESS
            sDSNItem = Space$(1024)
            sDRVItem = Space$(1024)
            i = SQLDataSources(lHenv, SQL_FETCH_NEXT, _
                               sDSNItem, 1024, iDSNLen, sDRVItem, 1024, iDRVLen)
            sDSN = Left$(sDSNItem, iDSNLen)
            sDRV = Left$(sDRVItem, iDRVLen)

            If sDSN <> Space(iDSNLen) Then
                cboDSNList.AddItem sDSN
                cboDrivers.AddItem sDRV
            End If
        Loop
    End If

    'some additional code has been removed for this listing
    'see the ODBC logon form for the rest.
End Sub

```

When the ODBC Logon form is displayed, the `Form_Load` event runs and executes a call to the subroutine `GetDSNsAndDrivers`. If you examine this subroutine, you will see it uses calls to Windows ODBC32.DLL API routines, `SQLAllocEnv`, and `SQLDataSources` to list all of the available DSNs and drivers. This is information you might want to include as setup information when running your tests since it may vary from system-to-system. The code in the `cmdOK_Click` event creates a connection string and has code that will allow you, the programmer, to choose between an ADO connection, a DAO connection, or an RDO connection by uncommenting the correct lines. The ADO code option can be used to open a DSN or DSN-less connection depending on the field values filled in on the form by the user. Figure 8-4 displays the ODBC Logon form.

Figure 8-4. The ODBC Logon form generated by a form template.

I hope you have noticed that there is a lot of work being done here by the form template code, which you can copy and paste rather than writing it from scratch. The ODBC Logon form and its associated code can be modified to suit your needs for logging into a database.

Another way to let Visual Basic generate code for you is to use the Data Form Wizard. In Chapter 2, we used it to quickly connect to a database and to view data in a prebuilt form. The Data Form Wizard has an option to connect to remote ODBC databases and then produces an option to generate ADO code for the connection. If you select this ADO code option when using the wizard, then create the form and view the code behind it, you will find code to create the connection and Recordset objects. You will also find the code to move between records on a form and bind the data to those fields. If you need to do anything like this, this code is a good starting point. Listing 8-3 displays a portion of the code generated by this wizard for the `Form_Load` event.

Listing 8-3. A portion of the code generated from by the Data Form Wizard.

```

Dim db As Connection
Set db = New Connection
db.CursorLocation = adUseClient
db.Open
"PROVIDER=MSDASQL;driver={SQL Server};" & _
    "server=C343600-A;uid=sa;pwd=;database=Northwind;"

Set adoPrimaryRS = New Recordset
adoPrimaryRS.Open _
    "select ProductID,ProductName,SupplierID,CategoryID, Unitprice from Products", _
    db, adOpenStatic, adLockOptimistic

```

Testing SQL Server Databases Using COM

There's another way to get at the structure and data of a database and that is if the database exports a COM library. If it does, the database can also be accessed by setting a reference to it as we have done with other applications that export libraries, like Word and Excel. SQL Server since version 6 exports its COM architecture through a library called the SQL-DMO. (DMO stands for Distributed Management Objects.) SQL-DMO is a powerful and fast way to access SQL Server once you have become familiar with the basic library objects. In fact, SQL Server's own Enterprise Manager software, a front end to the DBMS itself, is written using SQL-DMO. Figure 8-5 shows setting a reference to the SQL-DMO library in the Project References dialog. Doing this will give you access to all of the SQL database objects and allow you to write code to connect to a SQL Server database.

There is another object library for SQL Server, SQL-NS. This is the Name-space library and includes objects that can be used to access wizards and dialogs from the SQL Server Enterprise Manager. Both the SQL-DMO and SQL-NS libraries install with SQL Server. The SQL-DMO COM object library is displayed in the Object Browser as shown in Figure 8-6.

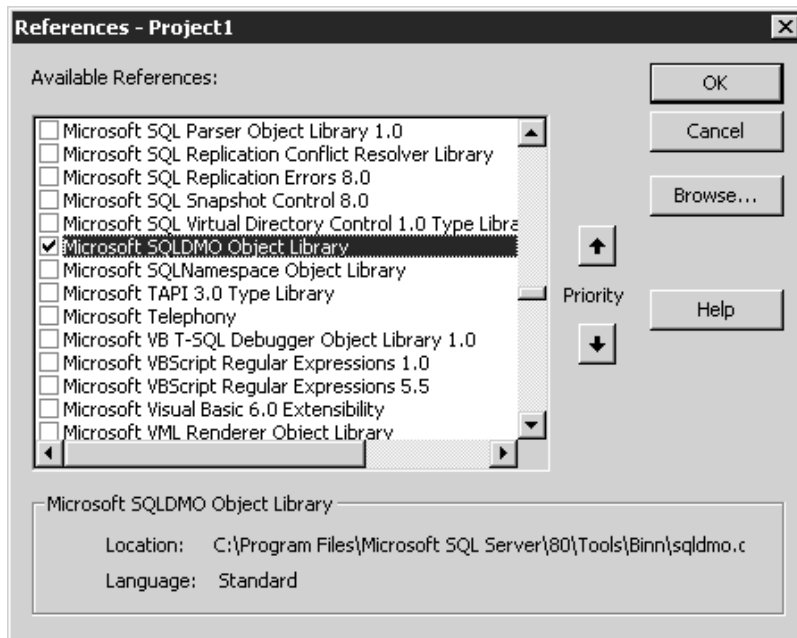


Figure 8-5. Setting a reference to SQL Server's SQL-DMO library.

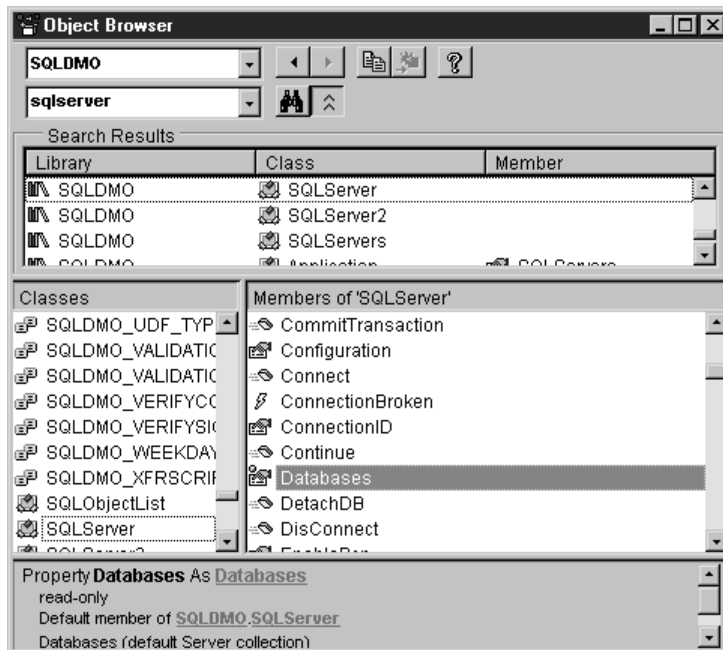


Figure 8-6. The SQL Server DMO object library.

Listing 8-4 accesses a SQL Server using the COM objects exposed by the SQL-DMO library.

Listing 8-4. Code to access a SQL Server and list all of its available databases in a list box.

```
Option Explicit
'The following code was written by Walt Rischer, President of the NW VBDA
' (Visual Basic Developer's association)
Private oServer As SQLDMO.SQLServer 'Create the Server object:
Private Const TestServer = null 'set this value to your actual server
                                'or It will default to the local server

Private Sub Form_Load()
    Dim oDB As SQLDMO.Database
    Set oServer = New SQLDMO.SQLServer
    oServer.Connect TestServer, "sa" 'connect to the server as system admin
    lstDatabases.Clear
    For Each oDB In oServer.Databases
        lstDatabases.AddItem oDB.Name 'list all databases on the server
    Next
End Sub
```

In Listing 8-4, an object is created to connect to a specific server in the Form_Load subroutine. The code can then access any of the exposed properties and methods of the object. The code above uses the databases collection in the connected SQL Server and loops through it, loading each database name (using the Name property) into a list control.

Listing 8-5 calls the PingSQLServerVersion method of the Server object to determine the version of the SQL Server installation.

Listing 8-5. Code to interrogate a SQL Server and return its version using the SQL-DMO object library.

```
Private Sub cmdPingIt_Click()
'this routine uses the PingSQLServerVersion function to determine
'The correct version of the server.
'Author: Walt Rischer
Dim strMessage As String
Dim lVersion As Long
```

```

On Error GoTo errhand 'error handling
    lVersion = oServer.PingSQLServerVersion(SERVER, "sa")
    Select Case lVersion
        Case SQLDMOSQLVer_80 'this constant only exists in SQL 2000;
            'if using SQL 7 or earlier version you must comment out
            strMessage = "SQL Server 2000"
        Case SQLDMOSQLVer_70
            strMessage = "SQL Server 7.0"
        Case SQLDMOSQLVer_65
            strMessage = "SQL Server 6.5"
        Case Else
            strMessage = "unable to determine version " & lVersion
    End Select
    MsgBox strMessage
Exit Sub
errhand:
    MsgBox "Unable to connect to server " & vbCrLf & _
        " Error: " & Err.Number & " " & Err.Description
End Sub

```

In the procedure `cmdPingIt_Click` (Listing 8-5), the `PingSQLServerVersion` method of the `Server` object is used to return the current version of the `Server`. This application is a simple utility for returning general information about a SQL Server, its available databases, and its version—useful information on a test project. It's a good start on a utility you may want to customize for your own SQL Server test project. You can access this utility in the Chapter 8 Practice file by opening the project `Chapter8\Demos\SQLServerInterface.vbp`.

Figure 8-7 shows the main form of the utility using the code in Listings 8-2 and 8-3.

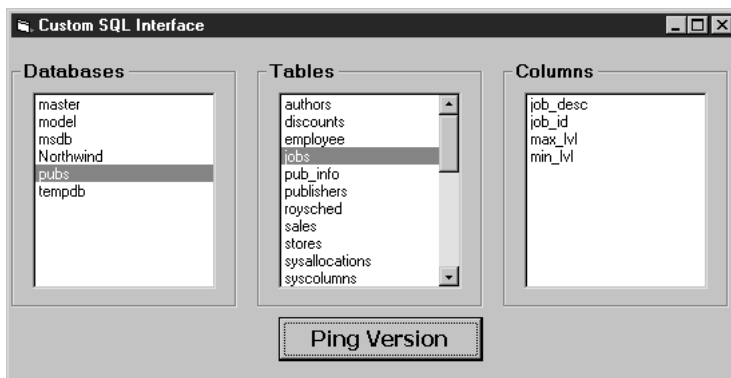


Figure 8-7. Walt Rischer's simple SQL-DMO utility.

Does all of this seem like a lot of coding work? What are the advantages to accessing a database using ADO or COM libraries like the SQL-DMO, especially when using the Visual Database tools is fast and easy? Using the Visual Database tools is easy but must be performed manually. Once you have written the code to test a database using ADO or the COM libraries, you can use the code again and again. With code, the value is that you may only have to occasionally change a few values, such as table names or expected row counts, and then run it without taking the time to do manual investigation.

Using the SQL-DMO and other COM object libraries effectively requires a little more knowledge of the structure of COM including classes and collections. I will cover this in greater detail in Chapter 9.

The Many Ways to Test Databases Using Visual Basic

So far, we have seen that we can use the Visual Database tools, the ADO library, and in some cases, COM libraries to access databases for testing purposes. I chose to present the ADO library because of its advantages in accessing data of all types but you could also choose to use DAO or RDO to access databases from within Visual Basic. It is also possible to use the ODBC32.DLL library to write code to test a database as we saw in the ODBC Logon form template code. Writing code to access the ODBC32.DLL would definitely require advanced programming skills.

Yet another way to test a database programmatically is to access the API provided specifically for that database by the developers. Not *every* application has its own set of API, but many times, they do. You will have to check with your application's developers to determine if such API exist. This would also likely require more advanced programming skills.

EXERCISE 8-1.

QUERYING A DATABASE USING VISUAL DATABASE TOOLS AND ADO

The purpose of Exercise 8-1 is to increase your familiarity with the Visual Database tools and ADO programming by comparing their use. First, you will create a connection to a database using a data link and execute a query using the Query Builder window. Then, you will write ADO code to accomplish a similar task. You will also modify the data in the database (this will work since the database you will connect to is a Microsoft Jet database) and then run your ADO code to verify the change.

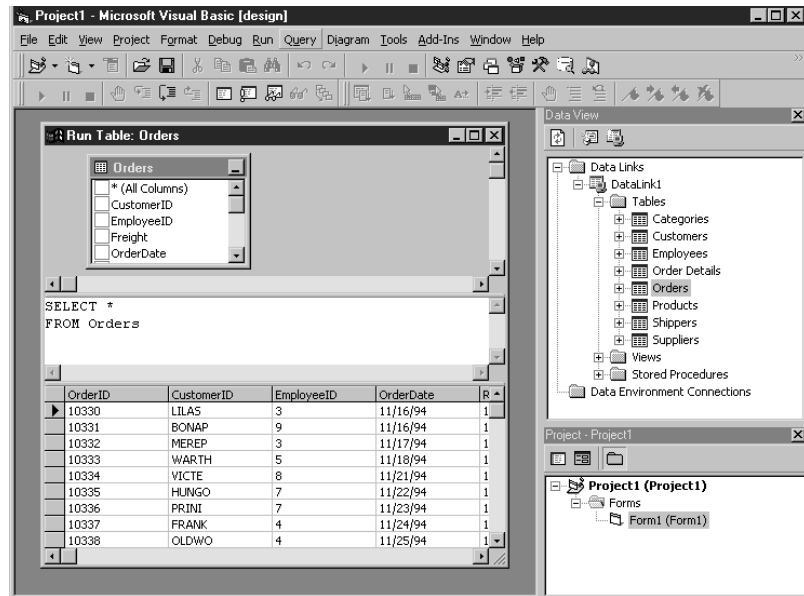
Follow These Steps to Complete the Exercise

1. Start a new Visual Basic project.
2. Follow steps 1 through 4 from the previous section, “Using the Data View Window,” in this chapter. You will be creating a data link to the **Microsoft Jet 4.0 OLE DB Provider** and specifying the **C:\Program Files\Microsoft Visual Studio\VB98\NWIND.MDB** database. When setting the Data Link properties from the Data Link Properties dialog, select the **Advanced** tab and click the **Read/Write** checkbox.



WARNING You **must** select read/write capability because you will be adding data to the database later.

3. In the Data View window, expand the **Data Links** folder, expand the data link you just created, and then expand the **Tables** folder.
4. Double-click the **Orders** table. The Query Builder window will be displayed. Click the **title bar** of this window to make sure it is selected.
5. Select the **View > Show Panes** menu item and select the **Diagram** and **SQL** panes.



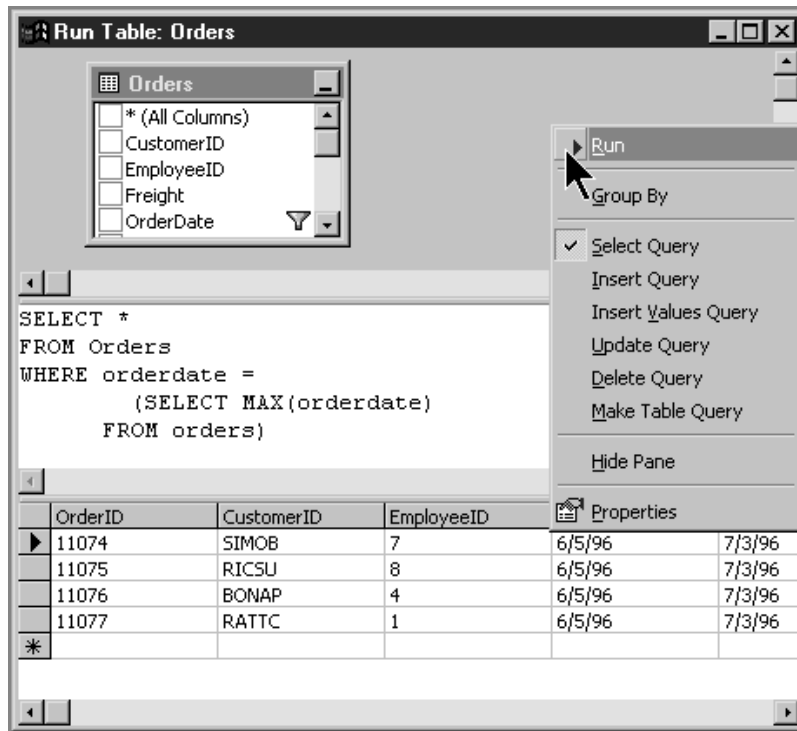
- Next, you will add SQL code into the SQL pane to return the mostly recently ordered items in the database. To accomplish this in the SQL pane, erase the **Select * from Orders** command by highlighting and backspacing. In its place, type the following command exactly as shown here:

```
SELECT *
FROM Orders
WHERE orderdate =
      (SELECT MAX(orderdate)
      FROM orders)
```



NOTE When typing into the SQL pane, you need not use the “_” underscore character to continue lines. You are not typing Visual Basic code into the SQL pane, you are typing SQL code. SQL statements have different syntax—the line-continuation character is not required in SQL syntax.

- Right-click in the **SQL** pane and select **Run** from the pop-up menu as shown:



Take note of how many records and their values have been returned. Now we will write ADO code to execute the same statements programmatically.



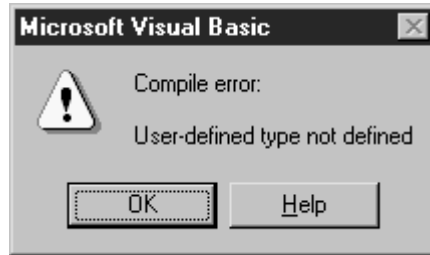
NOTE Do not close this window, you will return to it later in order to add data to the database.

8. Select the **View > Object** menu item from the Standard menu. This will display the default form for the project.
9. Add two buttons to the form. In the Properties window, change the following properties of these two new buttons as follows:
 - Name property: **cmdADO**; Caption property: **Click to Start ADO Test**
 - Name property: **cmdResults**; Caption property: **Click to View Test Results**

10. Open the Code window for the form and create the following two object variables:

```
Private cnnNW As adodb.Connection
Private rsMaxOrders As adodb.Recordset
```

11. Press **F5** or select the **Run > Start** menu item. You will get the following compile error dialog:



I wanted you to see this error at least once. You will receive this error anytime you try to use objects from a library for which you have not yet set a reference. Click **OK** to dismiss the error dialog.

12. Set the reference to the **Microsoft ActiveX Data Objects** library by selecting the **Project > References** menu item and checking the appropriate box. This test will require some logging so add a reference to the **Microsoft Scripting Runtime** library also. Then click **OK** to close the dialog. (You can now try running the program again. Nothing much will happen since you have not done anything but you will not get the error message this time.) Return to the Code window for the remaining steps.



NOTE You will find several libraries that say “Microsoft ActiveX Data Objects” with different version numbers. Which to choose? Select the highest-level version; at this writing, this is the Microsoft ActiveX Data Objects 2.6 library. The others are there for compatibility with code that may have been written to use them. Since you are writing new code, select the highest library version available.

13. Add the LogUtil Standard module you created in Chapter 5 Exercise 5-2 or you can add one that is already waiting for you in the Chapter8\Exercises folder by selecting the **Project > Add Module** menu item. The **Add Module** dialog displays. Select the **Existing** tab, find the **LogUtil.bas** file, and double-click it to add it into your project.

14. Next, you will add code to the click-event of the cmdADO button to open a connection to the Microsoft NWIND sample database that installs with Visual Basic 6 in the Program Files\Microsoft Visual Studio\VB98 library. This sample database is a Jet database (.mdb) so the provider is Microsoft.Jet.OLEDB.4.0. You will also add code to open a recordset with the same SQL command we used to return the most recently ordered items in the database, as well as add code to log the number of records found to a log file.

If you want to try to accomplish this on your own, ignore the following code and use the previous chapter as a guide.

Or, you can type the following to the cmdADO_Click event:

```
Dim strRow As String
Dim fldHold As Variant
Appname = "Northwind DB (.mdb) "
cnnNW.Provider = "Microsoft.Jet.OLEDB.4.0"
cnnNW.ConnectionString = "C:\Program Files\Microsoft Visual Studio\VB98\NWIND.MDB"
cnnNW.Open

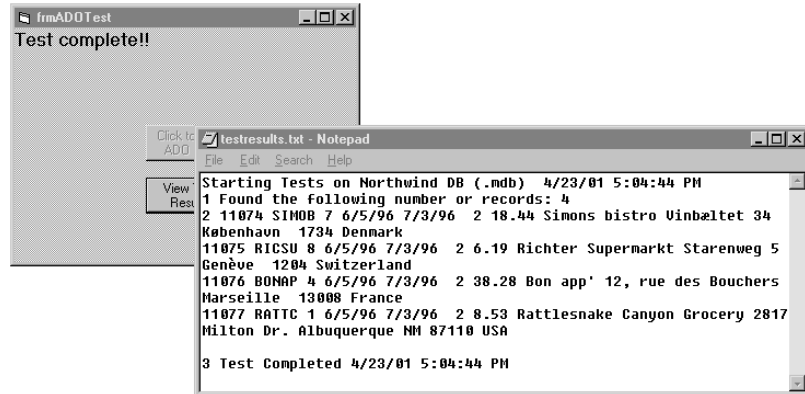
rsMaxOrders.Open "Select * from orders where orderdate = " & _
                  "(select max(orderdate) from orders)", cnnNW, adOpenStatic

rsMaxOrders.MoveFirst
LogToFile "Found the following number of records: " & rsMaxOrders.RecordCount
Do While Not rsMaxOrders.EOF
    For Each fldHold In rsMaxOrders.Fields
        strRow = strRow & fldHold & " " 'load a string with all fields in this row
    Next fldHold
    strRow = strRow & vbCrLf 'carriage return after each row
    rsMaxOrders.MoveNext    'move to the next row in the record set
Loop
LogToFile strRow            'log to the test results file
'clean up
rsMaxOrders.Close
cnnNW.Close

Print "Test complete!!"
LogToFile "Test Completed " & Now
cmdResults.Enabled = True
cmdADO.Enabled = False
```

15. Add the following line into the cmdResults_Click event so that the log file can be viewed when this button is clicked:

ReadLog



The answer for this code is located in Chapter8\Exercises\Answers\ADOExercise8_1.vbp. Do not close out of Visual Basic, the next exercise continues on with the same files.

EXERCISE 8-2.**ADDING TEST DATA**

In Exercise 8-2, you will continue your work with the Orders table in the Northwind database. You will add a new row to the Orders table using the Visual Database tools and then verify its existence by running the ADO code you created in Exercise 8-1. This exercise presumes you have accomplished Exercise 8-1.

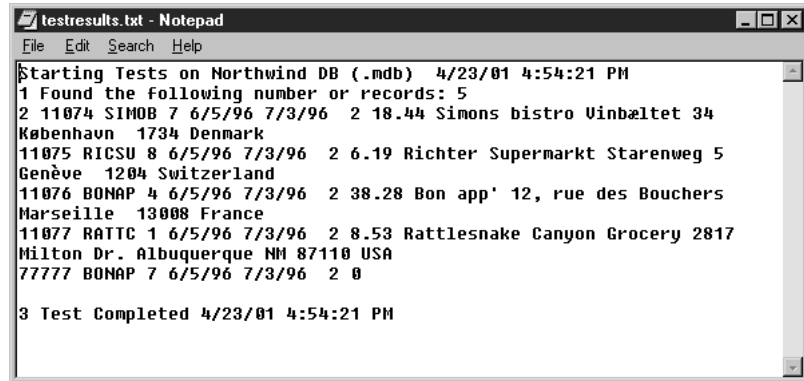
Follow These Steps to Complete the Exercise

1. Switch back to the Query Builder window. (If you closed it, open the Data Link window by selecting the **View > Data Link** menu item and expanding the data link and tables folders. You can then double-click on the **Orders** table to produce the Query Builder window once again. Use the **View > Show Panes** menu item to show the SQL and Diagram panes.)
2. You will enter a new row into the Orders table using a SQL statement. This row will have an OrderDate column equal to the most recent order so it should show up in our query to return most recent orders. You may have to adjust the date depending on how current the data in your database is. In the following statement, the orderdate is 6/5/96; modify this date, if necessary, to make sure it's the same as the most recent orderdate. Erase whatever value is in the SQL pane and type the following SQL Insert statement into the SQL pane:

```
INSERT INTO Orders
    (orderid, customerid, employeeid, orderdate, requireddate,
    shipvia)
VALUES (77777, 'BONAP', 7, '6/5/96', '7/3/96', 2)
```

3. Right-click in the **SQL** pane and select **Run** from the pop-up menu to execute this query. If you have any problems, check the syntax and try again. Once the query has successfully run, you will receive a message box saying, "1 row affected by last query." This means the new row was inserted correctly. Click **OK** to dismiss this message box.

4. Now run your own project (from Exercise 8-1 or you can run the answer from Chapter8\Exercises\Answers\ADOExercise8_1.vbp). When the Query Builder window has the focus, the Run menu item and toolbar button are disabled. To run the project first, click the **View > Object** menu item to again view your Visual Basic form. You can now run the project by selecting **F5** or the **Run > Start** menu item
5. Your test results should now show that another row qualifies.



```
testresults.txt - Notepad
File Edit Search Help
Starting Tests on Northwind DB (.mdb) 4/23/01 4:54:21 PM
1 Found the following number of records: 5
2 11074 SIMOB 7 6/5/96 7/3/96 2 18.44 Simons bistro Vinbæltet 34
København 1734 Denmark
11075 RICSU 8 6/5/96 7/3/96 2 6.19 Richter Supermarkt Starenweg 5
Genève 1204 Switzerland
11076 BONAP 4 6/5/96 7/3/96 2 38.28 Bon app' 12, rue des Bouchers
Marseille 13008 France
11077 RATTC 1 6/5/96 7/3/96 2 8.53 Rattlesnake Canyon Grocery 2817
Milton Dr. Albuquerque NM 87110 USA
77777 BONAP 7 6/5/96 7/3/96 2 0
3 Test Completed 4/23/01 4:54:21 PM
```


EXERCISE 8-3.**TESTING USING SQL-DMO**

In Exercise 8-3, you will access a SQL Server database using the SQL-DMO library.



WARNING *Microsoft SQL Server 7 or 2000 or the client components must be installed (so that you may access another SQL Server across a network) before you can begin the exercise. If you do not have access to any SQL Server, you will not be able to perform this exercise.*

Follow These Steps to Complete the Exercise

1. Open the **Chapter8\Exercises\SQLServerInterface.vbp** file in the online Practice files.
2. Start the Visual Basic debugger by pressing **F8**. Review the code by stepping through it line-by-line with the debugger. As you step through the code, use your cursor to view the contents of the object variables and properties.
3. Open the **Object Browser** and select the **SQL-DMO** library. Look up the Server object and view its properties. Look up the databases and tables collections and read the available Help for these objects.
4. Add a new button anywhere on the form and name it cmdGetData.
5. Add the following code to display the number of rows in the Sales table in the Pubs database:

```
Private Sub cmdGetData_Click()  
    MsgBox "Number of Sales in Pubs: " _  
        & oServer.Databases("Pubs").Tables("Sales").Rows  
End Sub
```

6. Compile, debug, and run your code. Save your files.
7. An answer to this exercise is in the project
Chapter8\Exercises\Answer\SQL_DMO8_3.vbp.

To verify your answer, try connecting to the Pubs database via the Visual Data-base tools.

TESTER'S CHECKLIST

When testing a relational database:



- ☐ Use the Visual Database tools to quickly and easily connect to an ODBC-compliant database and visually verify data. Use ADO or SQL-DMO to connect to a database programmatically if you need to save and run automated scripts to test the database.
- ☐ Save queries you have found that find data problems. Start with those SQL queries in the section, “Using the Query Builder Window to Execute Database Queries” earlier in this chapter. These queries can be used programmatically by using ADO or SQL-DMO or they can be used within the Visual Database tools.
- ☐ Use the ODBC Logon form template to quickly perform database connection tasks. Remember, you can copy and paste the code from this form for your own use. The Data Form Wizard also generates code for quick database access.
- ☐ Become proficient at database design and SQL.

Chapter 8 Review

- Describe how the Visual Database tools can be used to support testing of a database application.
See page 256.
- List two ways to access a database using Visual Basic code.
See pages 264 and 278.
- Explain the difference between accessing a database via SQL-DMO and ADO.
See pages 264–282.
- List the steps to connect to a database using the SQL-DMO.
See pages 278–282.
- List the steps to connect to a database using ADO.
See pages 269–272.