Database Programming with C#

CARSTEN THOMSEN

Apress™

Database Programming with C#

Copyright © 2002 by Carsten Thomsen

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): 1-59059-010-4

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.

Technical Reviewer: Douglas Milnes

Editorial Directors: Dan Appleman, Peter Blackburn, Gary Cornell, Jason Gilmore, Karen Watterson, John Zukowski

Managing Editor: Grace Wong

Copy Editors: Nicole LeClerc, Ami Knox

Production Editor: Tory McLearn

Compositor: Impressions Book and Journal Services, Inc.

Indexer: Valerie Haynes Perry

Cover Designer: Tom Debolski

Marketing Manager: Stephanie Rodriguez

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, email orders@springer-ny.com, or visit http://www.springer-ny.com.

Outside the United States, fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 9th Street, Suite 219, Berkeley, CA 94710.

Email info@apress.com or visit 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.

The source code for this book is available to readers at http://www.apress.com in the Downloads section.

You will need to answer questions pertaining to this book in order to successfully download the code.

CHAPTER 6

Using Stored Procedures, Views, and Triggers

How to Use Stored Procedures, Views, and Triggers

Server-side processing, which is when you let a server process your queries and the like, is probably a concept you have heard of and it's the very topic of this chapter. Well, to some extent anyway. I discuss three specific ways of doing server-side processing: stored procedures, triggers, and views. The good thing about server-side processing is that you can use the power and resources of your server for doing purely data-related processing and thus leave your client free to do other stuff, and your network clearer of data that the client doesn't want. It's not always appropriate to do so, but in many cases you can benefit from it.

This chapter includes several hands-on exercises that will take you through creating stored procedures, views, and triggers. See the Exercise items that appear throughout the text.

Although this chapter primarily focuses on SQL Server 2000 features, some of the functionality can certainly be reproduced in the other DBMSs I cover in this book:

- *SQL Server 7.0:* All functionality shown in this chapter can be reproduced. However, SQL Server 7.0 doesn't support the INSTEAD OF triggers described in "Using Triggers."

- *Microsoft Access:* Microsoft Access doesn't support stored procedures or triggers. However, views can be reproduced as queries in Microsoft Access, but you can't do this from within the VS .NET IDE; you have to use other means, like the Microsoft Access front-end. If you are unfamiliar with

Microsoft Access, I can recommend you read the following book to get you up to speed: *From Access to SQL Server*, by Russell Sinclair. Published by Apress, September 2000. ISBN: 1893115-240.

- *Oracle:* Oracle supports all the server-side processing described in this chapter.

- *MySQL:* For the examples in this book, I have been using MySQL version 3.23.45, which doesn't support triggers, views, or stored procedures, meaning there is no example code for MySQL in this chapter. However, at the time of writing (March 2002), an alpha version (4.0) of MySQL is available for download from `http://www.mysql.com`. The final version 4.0 is supposed to support stored procedures, views, and triggers. Even when these server-side processing means are available in MySQL, it's still not possible to create any of these items from within the VS .NET IDE.

The code for this chapter has examples for all the listed DBMSs where appropriate.

Optimization Issues

When I talk about optimizing performance of an application, there are a number of things to consider, but let's just make one thing clear before I go on: I am only talking distributed applications and not stand-alone applications that sit nicely on a possibly disconnected single PC. These stand-alone applications are also called *single tier* or *monolithic applications*.[1] The applications I discuss here use a network of some sort to access data and business services.

Okay, now that the basics are out of the way, I can focus on the obstacles that can lead to decreasing performance and how you need to know these obstacles well when you start the optimization process. You should keep such obstacles in mind when you design your application. However, the various resources, such as network bandwidth, processor power, available RAM, and so on, most often change over time, and then you'll have to reconsider if your application needs changing.

Table 6-1 lists all the varying factors that can influence the performance of your application, which could be a topic for an entire book. However, although I only describe these factors briefly, I want you to be aware of the resources mentioned; they have great influence on what server-side processing resources you should choose when you design your application. In general, it's often the client queries and not the server itself that create the biggest performance problems.

1. Stand-alone applications don't have to be single tier, but they generally are.

Table 6-1. Performance Resources Optimization

RESOURCE NAME	DESCRIPTION
Network resources	When speaking of network resources, I am referring to the actual bandwidth of the network. Consider your network setup—whether you are on a LAN or you are accessing resources over a WAN such as the Internet, and so on. If you have a low bandwidth, it's obvious that you want to transfer as little data across the network as possible. If on the other hand you have plenty of bandwidth, you might want to transfer large amounts of data across the network. However, best practices prescribe that you only transfer the data needed across your network, even when you have wide bandwidth.
Local processing resources	If you have the raw power available on your local box, it can be good to do most of the data processing there. Mind you, it all depends on the available bandwidth and the processing resources on the server.
Server processing resources	Server-side processing is desirable, if the server has resources to do so. Another thing you should consider is whether it has the resources to serve all your clients, if you let the server do some of the data processing.
Data distribution	Although strictly speaking this isn't a resource as such, it's definitely another issue you might need to consider. If your data comes from various different and even disparate data sources, it often doesn't make too much sense to have one server process data from all the data sources, just to send the result set to the client. In most cases, it makes sense to have all the data delivered directly to the client.

Table 6-1 just provides a quick overview. Table 6-2 shows you some different application scenarios.

Table 6-2. Different Application Scenarios

CLIENT MACHINE	SERVER	NETWORK	RECOMMENDATION
Limited processing resources	Plenty of processing resources	Limited bandwidth	Now, this one is obvious. You should use the raw processing power of the server to process the data and only return the requested data. This will save resources on the network and on the client.
Plenty of processing resources	Plenty of processing resources	Limited bandwidth	Hmm, processing could be done on either the client or the server, but it really depends on the amount of data you need to move across the network. If it's a limited amount of data, processing on either side will do, but if it's a lot of data, then let the server do the processing. Another solution could be to store the data locally and then use replication or batch processing to update the server.
Plenty of processing resources	Limited processing resources	Limited bandwidth	In this case, processing should be done on the client, but it really depends on the amount of data you need to move across the network. If it's a limited amount of data, the client should do the processing; but if it's a lot of data, you might consider letting the server do some of the processing, or even better; upgrade your server.
Plenty of processing resources	Limited processing resources	Plenty of bandwidth	Okay, don't think too hard about this one—processing should be done on the client.

I could add plenty more scenarios to the list, but I think you get the picture. You'll rarely encounter a situation that matches a straightforward scenario with a simple answer. It's your job to know about all the potential issues when you design your application and have to decide on where to process your data. Quite often different aspects of an application have different data processing needs, so the answer may vary even within a single application. One book that will help

you with many common problems you may encounter with SQL Server is this one:

- *SQL Server: Common Problems, Tested Solutions,* by Neil Pike. Published by Apress, October 2000. ISBN: 189311581X.

Troubleshooting Performance Degradation

When you realize that you have performance problems or when you just want to optimize your server, you need one or more tools to help. SQL Server and Windows NT/2000 provides a number of tools you can use when troubleshooting and here are a few of them:

- Database Consistency Checker (DBCC) (SQL Server)
- Performance Monitor (Windows NT/2000)
- Query Analyzer (SQL Server)
- System Stored Procedures (SQL Server)

I'll briefly describe what you can use these tools for and give you links for obtaining more information.

Database Consistency Checker

The *Database Consistency Checker* (DBCC) is used for checking the logic as well as the consistency of your databases using T-SQL DBCC statements. Furthermore, many of the DBCC statements can also fix the problems detected when running. DBCC statements are T-SQL enhancements and as such must be run as SQL scripts. Here is one example of a DBCC statement:

```
DBCC CHECKDB
```

This DBCC statement is used for checking the structural integrity of the objects in the database you specify. It can also fix the problems found when running. There are many DBCC statements, and this isn't the place to go over these, but check SQL Server Books Online (included with SQL Server) for more information about DBCC.

Performance Monitor

The *Performance Monitor* (perfmon) is used for tracking and recording activity on your machine or rather any machine within your enterprise. perfmon comes with Windows NT/2000/XP and is located in the Administrative Tools menu, but you can also run it from a command prompt, or the Run facility of Windows Start Menu, by executing perfmon. Any of the Windows platforms mentioned produces counters that can be tracked or polled by perfmon at regular intervals if needed. SQL Server also comes with counters that can be tracked or polled by perfmon. Some of the more general counters are used for polling processor time, disk access, memory usage, and so on. Arguably the best of it all is the ability to save a session of all activity recorded or polled within any given time frame. You can then play back a saved session, whenever appropriate. This is especially important when you want to establish a baseline against which to compare future session recordings.

Check your Windows NT/2000/XP documentation for more information about perfmon.

Query Analyzer

The *Query Analyzer* is an external tool that comes with SQL Server for analyzing and optimizing your queries. You can find it in the menus created by SQL Server Setup.

Query Analyzer can be used for validating your queries in the form of script files and queries you type yourself in the query window. Besides validating a query, you can get Query Analyzer to analyze it by running. The analysis includes an execution plan, statistics, and a trace of the query being executed. Queries can get complicated, and many do when joining tables, and it isn't always obvious how much processing a particular query will take. There's normally more than one way to get to complex data, so the trace is invaluable in optimizing your data requests.

See SQL Server Books Online (included with SQL Server) for more information about Query Analyzer. You can actually invoke the Query Analyzer part of the SQL Server Books Online help text from within Query Analyzer by pressing F1.

System Stored Procedures

The *System Stored Procedures* is a set of stored procedures that comes with SQL Server for database administrators to use for maintaining and administering SQL Server. There are a number of System Stored Procedures, including two XML ones, and I certainly can't cover them here, but I can mention some of the

functionality they cover: they let you see who's logged on to the system, administer registration with Active Directory, set up replication, set up full-text search, create and edit maintenance plans, and administer a database in general.

See SQL Server Books Online (comes with SQL Server) for more information about the System Stored Procedures.

Using Stored Procedures

A *stored procedure* is a precompiled batch[2] of SQL statement(s) that is stored on the database server. The SQL statements are always executed on the database server. Stored procedures have long been a good way of letting the server process your data. They can significantly reduce the workload on the client, and once you get to know them you'll wonder how you ever managed without them.

There is certainly more to a stored procedure than just mentioned, but I do think this is the most significant aspect of a stored procedure. Think about it: it's a way of grouping a batch of SQL statements, storing it on the database server, and executing it with a single call. The fact that the stored procedure is precompiled will save you time as well when executed. Furthermore, the stored procedure can be executed by any number of users, meaning you might save a lot of bandwidth just by calling the stored procedure instead of sending the whole SQL statement every time.

A stored procedure can contain any SQL statement that your database server can understand. This means you can use stored procedures for various tasks, such as executing queries—both so-called action queries, such as DELETE queries, and row-returning queries, such as SELECT statements.

Another task you can use a stored procedure for is database maintenance. Use it to run cleanup SQL statements when the server is least busy and thus save the time and effort of having to do this manually. I won't cover maintenance tasks in this chapter, but they are important, and you should be aware of the various tasks you can perform with stored procedures. If you're like me, you have been or are working for a small company that doesn't have a database administrator, in which case you're in charge of keeping the database server running. Granted, it's not an ideal situation, but you certainly get to know your DBMS in different ways than you would just being a programmer, and that's not bad at all.

To sum it up: a stored procedure is a precompiled SQL statement or batch of SQL statements that is stored on the database server. All processing takes place on the server, and any result requested by a client is then returned in a prepared format.

2. Actually some stored procedures only hold one SQL statement.

Why Use a Stored Procedure?

You should use a stored procedure in the following cases (please note that other cases do apply, depending on your circumstances):

- Executing one or more related SQL statements on a regular basis
- Hiding complex table structures from client developers
- Returning the result of your SQL statements because you have a limited bandwidth on your network
- Delegating data processing to the server because you have limited processing resources on your client
- Ensuring processes are run, on a scheduled basis, without user intervention

Granted, there can be substantially more work in setting up a stored procedure than in just executing the SQL statement(s) straight from the client, but my experience has confirmed that the extra work saves you at least tenfold the time once you start coding and using you application. Even SQL Server itself and other major DBMSs use stored procedures for maintenance and other administrative tasks.

One last thing I want to mention is the fact that if you base a lot of your data calls on stored procedures, it can be much easier to change the data calls at a later date. You can simply change the stored procedure and not the application itself, meaning you don't have to recompile a business service or even your client application, depending on how you have designed your application. On the negative side, stored procedures are often written using database vendor–specific SQL extensions, which mean that they're hard to migrate to a different RDBMS. This of course is only a real concern if you're planning to move to another RDBMS.

Planning a Move to a Different RDBMS

If you're planning to move to another RDBMS from SQL Server, or just want to make it as easy as possible should management decide so in the future, it'll probably be a good idea to look up the following T-SQL statements in the SQL Server Books Online Help Documentation:

- **SET ANSI_DEFAULTS**: This statement sets the ANSI defaults on for the duration of the query session, trigger, or stored procedure.

- **SET FIPS_FLAGGER**: This statement can be used to check for compliance with the ANSI SQL-92 standard.

If you use these statements appropriately, they can certainly help ease the move from SQL Server to another ANSI SQL-92–compliant RDBMS.

Creating and Running a Stored Procedure

Creating a stored procedure is fairly easy, and you're probably used to working with the Enterprise Manager that comes with SQL Server or a different stored procedure editor for SQL Server or Oracle. If this is the case, you may want to check out the facilities in the Server Explorer in the VS .NET IDE. Among other things, it's much easier to run and test a stored procedure directly from the text editor. Anyway, here's how you would create a stored procedure for the example UserMan database:

1. Open up the Server Explorer window.

2. Expand the UserMan database on your database server.

3. Right-click the Stored Procedures node and select New Stored Procedure.

This brings up the Stored Procedure text editor, which incidentally looks a lot like your C# code editor. Except for syntax checking and other minor stuff, they are exactly the same (see Figure 6-1).

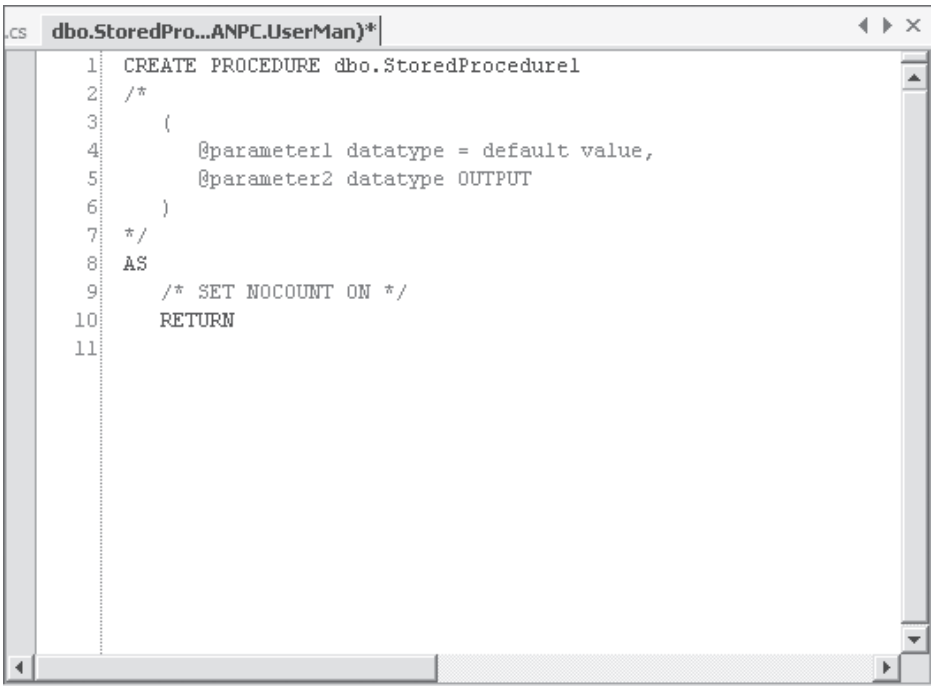

Figure 6-1. Stored procedure editor with SQL Server default template

> **NOTE** *With SQL Server it's only possible to use T-SQL for your stored procedures. However, the upcoming version of SQL Server, code-named Yukon, will have support for the .NET programming languages. Knowing this, perhaps you'll want to create your stored procedures in C# or VB .NET.*

Creating a Simple Stored Procedure

Once you've created a stored procedure, you need to give it a name. As you can see from your stored procedure editor, the template automatically names it `StoredProcedure1`. If you're wondering about the dbo prefix, it simply means that the stored procedure is created for the dbo user. In SQL Server terms, dbo stands for *database owner*, and it indicates who owns the database object, which is a stored procedure in this case. If you've been working with SQL Server for a while, you probably know the term *broken ownership chain*. An ownership chain is the dependency of a stored procedure upon tables, views, or other stored procedures.

Generally, the objects that a view or stored procedure depend on are also owned by the owner of the view or stored procedure. In such a case there are no problems, because SQL Server doesn't check permissions in this situation. (It doesn't really have to, does it?) However, when one or more of the dependent database objects are owned by a different user than the one owning the view or stored procedure, the ownership chain is said to be broken. This means that SQL Server has to check the permissions of any dependent database object that has a different owner. This can be avoided, if the same user, such as dbo, owns all of your database objects. I am not telling you to do it this way, but it's one option available to you.

Okay, let's say you've deleted the `StoredProcedure1` name and replaced it with `SimpleStoredProcedure`. To save the stored procedure before continuing, press Ctrl+S. If you saved your stored procedure at this point, you would notice that you don't have to name it using a Save As dialog box, because you've already named it. The editor will make sure that the stored procedure is saved on the database server with the name you've entered, which in this case is `SimpleStoredProcedure`. You shouldn't save it until you've renamed it, because you'll end up having to remove unwanted stored procedures.

Although you can see your stored procedure in the Stored Procedure folder of the SQL Server Enterprise Manager and the Stored Procedure node in the Server Explorer, there isn't actually an area in your database designated for just stored procedures. The stored procedure is saved to the system tables as are most other objects in SQL Server.

As soon as you have saved it, the very first line of the stored procedure changes. The SQL statement CREATE PROCEDURE is changed so that the first line reads:

```
ALTER PROCEDURE dbo.SimpleStoredProcedure
```

Why? Well, you just saved the newly created stored procedure, which means that you can't create another with the same name. Changing CREATE to ALTER takes care of that. It's that simple. In case you're wondering what happens when you change the name of your stored procedure and the SQL statement still reads ALTER PROCEDURE . . . , I can tell you: the editor takes care of it for you and creates a new procedure. Try it and see for yourself! Basically, this means that CREATE PROCEDURE is never actually needed; one can simply use ALTER PROCEDURE, even on brand new procedures. However, this can be a dangerous practice, if you inadvertently change the name of your stored procedure to the name of an already existing one.

The `SimpleStoredProcedure` doesn't actually do a lot, does it? Okay, let me show you how to change that. In Figure 6-1, you can see two parts of the stored procedure: The first part is the header and then there is the actual stored procedure itself. The header consists of all text down to and including Line 7. Basically,

the header declares how the stored procedure should be called, how many arguments to include and what type of arguments, and so on. Since this is a very simple procedure, I don't want any arguments, so I'll leave the commented-out text alone.

If you haven't changed the default editor settings, text that is commented out or any comments you have inserted yourself are printed in green. In a SQL Server stored procedure, comments are marked using start and end tags: /* for the comment start tag and */ for the comment end tag. This has one advantage over the way you insert comments in your C# code in that you don't have to have a comment start tag on every line you want to comment out. You only need to have both a start and end tag.

The second part of the stored procedure is the part that starts with the AS clause on Line 8. The AS clause indicates that the text that follows is the body of the stored procedure, the instructions on what to do when the stored procedure is called and executed.

EXERCISE

1) Create a stored procedure, name it **SimpleStoredProcedure**, and save it as described earlier.

2) Type the following text on Line 10 in the `SimpleStoredProcedure` in place of the RETURN statement:

   SELECT COUNT(*) FROM tblUser

   Now the stored procedure should look like the example in Figure 6-2. The stored procedure will return the number of rows in the tblUser table. Please note that it's generally good practice to keep the RETURN statement as part of your stored procedure, but I'm taking it out and leaving it for an explanation later, when I discuss return values and how they're handled in code.

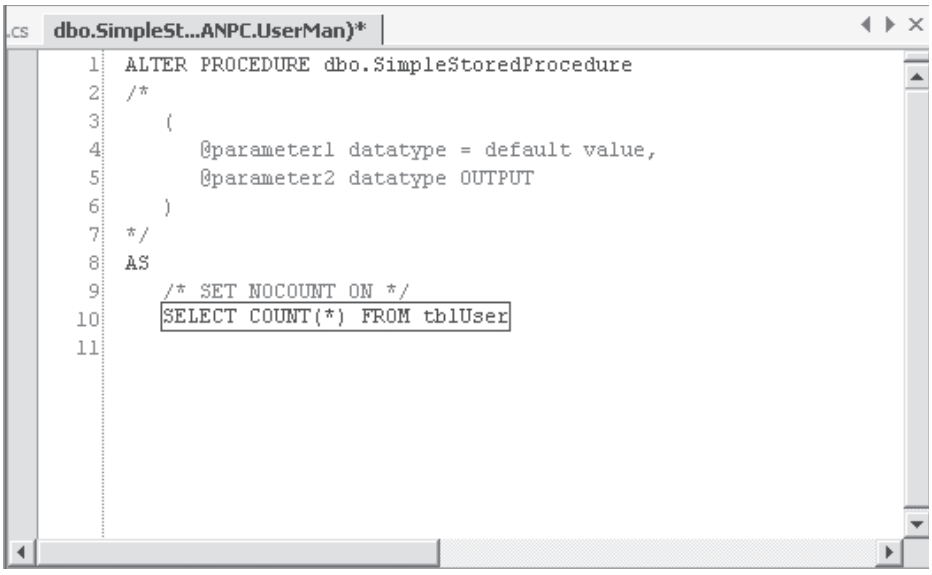

Figure 6-2. Stored procedure that returns the number of rows in the tblUser table

3) Don't forget to save the changes using Ctrl+S.

Running a Simple Stored Procedure from the IDE

Of course, there's no point in having a stored procedure that just sits there, so here's what you do to run it: if you have the stored procedure open in the stored procedure editor window, you can right-click anywhere in the editor window and select Run Stored Procedure from the pop-up menu. If you do this with the stored procedure you created in the exercise in the previous section, the Output window, located just below the editor window, should display the output from the stored procedure as shown in Figure 6-3.

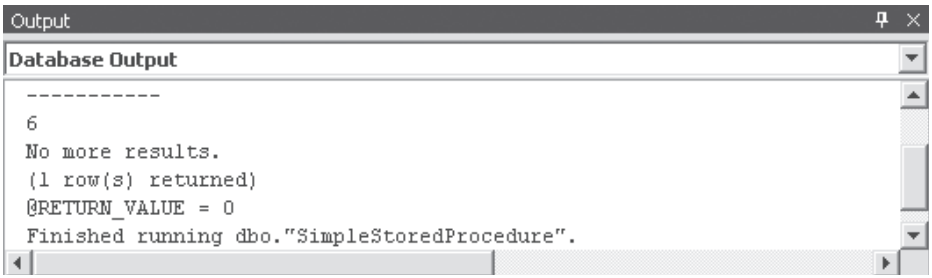

Figure 6-3. The Output window with output from `SimpleStoredProcedure`

If you have closed down the stored procedure editor window, you can run the stored procedure from the Server Explorer. Expand the database node, right-click the Stored Procedures node, and select Run Stored Procedure from the pop-up menu. This will execute the stored procedure the exact same way as if you were running it from the editor window.

Running a Simple Stored Procedure from Code

Okay, now that you have a fully functional stored procedure, you can try and run it from code. Listing 6-1 shows you some very simple code that will run the stored procedure. The example code in this listing uses data classes that were introduced in Chapters 3A and 3B.

Listing 6-1. Running a Simple Stored Procedure

```
public void ExecuteSimpleSP() {
    SqlConnection cnnUserMan;
    SqlCommand cmmUser;
    object objNumUsers;

    // Instantiate and open the connection
    cnnUserMan = new SqlConnection(STR_CONNECTION_STRING);
    cnnUserMan.Open();

    // Instantiate and initialize command
    cmmUser = new SqlCommand("SimpleStoredProcedure", cnnUserMan);
    cmmUser.CommandType = CommandType.StoredProcedure;

    objNumUsers = cmmUser.ExecuteScalar();
    MessageBox.Show(objNumUsers.ToString());
}
```

The code in Listing 6-1 retrieves the return value from the stored procedure. Now, this isn't usually all you want from a stored procedure, but it merely demonstrates what a simple stored procedure looks like. The stored procedure itself could just as well have had a `DELETE FROM tblUser WHERE LastName='Johnson'` SQL statement. If you want to execute this from code, you need to know if the stored procedure returns a value or not. It doesn't in this case, so you need to use the **ExecuteNonQuery** method of the **SqlCommand** class.

EXERCISE

1) Create a new stored procedure and save it with the name **uspGetUsers**.

2) Type in the following text on Line 10 in place of the RETURN statement:

SELECT * FROM tblUser

Now the stored procedure should look like the one in Figure 6-4. This stored procedure will return all rows in the tblUser table.

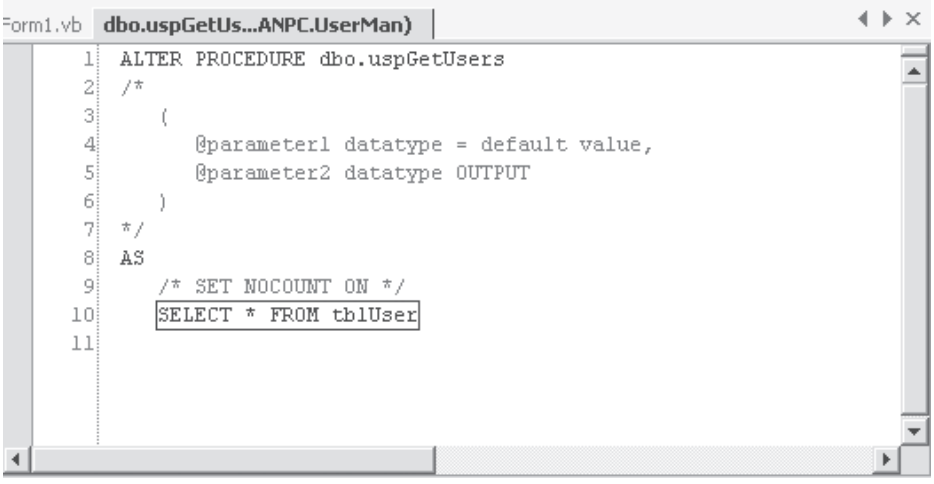

Figure 6-4. The `uspGetUsers` *stored procedure*

3) Don't forget to save the changes using Ctrl+S.

What you need now is some code to retrieve the rows from the stored procedure (see Listing 6-2).

Listing 6-2. Retrieving Rows from a Stored Procedure

```
public void ExecuteSimpleRowReturningSP() {
    SqlConnection cnnUserMan;
    SqlCommand cmmUser;
    SqlDataReader drdUser;

    // Instantiate and open the connection
    cnnUserMan = new SqlConnection(STR_CONNECTION_STRING);
    cnnUserMan.Open();

    // Instantiate and initialize command
    cmmUser = new SqlCommand("uspGetUsers", cnnUserMan);
    cmmUser.CommandType = CommandType.StoredProcedure;

    // Retrieve all user rows
    drdUser = cmmUser.ExecuteReader();
}
```

The example in Listing 6-2 retrieves the rows returned from the stored procedure by using the **ExecuteReader** method of the **SqlCommand** class. Please note that this method and the related **ExecuteXmlReader** method are the only options for retrieving rows as the result of a function call with the Command class.

Creating a Stored Procedure with Arguments

Sometimes it's a good idea to create a stored procedure with arguments[3] instead of having more stored procedures essentially doing the same. It also gives you some flexibility with regards to making minor changes to your application without having to recompile one or more parts of it, because you can add to the number of arguments and keep existing applications running smoothly by specifying a default value for the new arguments.

Another reason for using arguments with stored procedures is to make the stored procedure behave differently, depending on the input from the arguments. One argument might hold the name of a table, view, or another stored procedure to extract data from.

3. I'm using the word *argument* here, but I might as well call it *parameter*, like T-SQL does. However, the two words are synonymous in this case.

TIP *In SQL Server you can use the* `EXECUTE sp_executesql` *statement and System Stored Procedure with arguments of type* ***ntext***, ***nchar***, *or* ***nvarchar*** *to execute parameterized queries. See the SQL Server Books Online Help Documentation for more information.*

EXERCISE

1) Create a new stored procedure and save it with the name **uspGetUsersByLastName**.

2) Type in the following text on Lines 10 and 11 in place of the RETURN statement:

SELECT * FROM tblUser
WHERE LastName = @strLastName

3) Uncomment Lines 2 to 7, and insert the following text instead of Lines 3 and 4:

@strLastName varchar(50)

The stored procedure should look like the one in Figure 6-5. This stored procedure will return all rows in the tblUser table where the LastName column matches the `strLastName` argument.

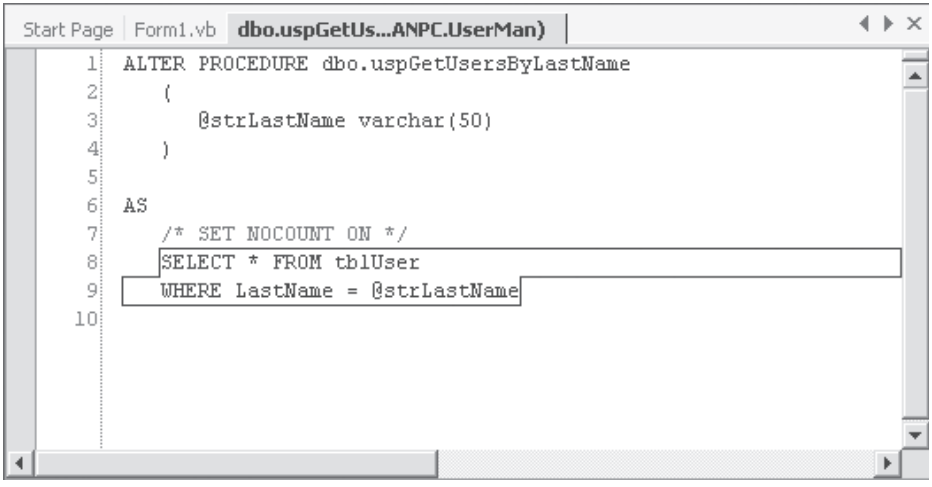

Figure 6-5. The `uspGetUsersByLastName` *stored procedure*

4) Don't forget to save your changes using Ctrl+S.

Arguments in stored procedures can be either input or output. If you include an input argument, you don't have to specify anything after the data type, but if you use an output argument, you need to specify the OUTPUT keyword after the data type.

> **NOTE** *I only cover the absolute basics of how to create a stored procedure in this chapter. If you need more information, I suggest you look up the CREATE PROCEDURE statement in the Books Online help application that comes with SQL Server.*

Running a Stored Procedure with Arguments from the IDE

Try and run the stored procedure you created in the last exercise and see how the argument affects how it's run. You can try running the stored procedure from either the editor window or the Server Explorer window. The Run dialog box asks you for a value for the `strLastName` argument. Type **Doe** in the text box and click OK. Now all users with the last name of Doe are returned as the result of the stored procedure.

Using a Stored Procedure with Arguments

The `uspGetUsersByLastName` stored procedure seems to work, so try and run it from code. Listing 6-3 shows how you would do this.

Listing 6-3. Retrieving Rows from a Stored Procedure with an Input Argument

```
public void GetUsersByLastName() {
    SqlConnection cnnUserMan;
    SqlCommand cmmUser;
    SqlDataReader drdUser;
    SqlParameter prmLastName;

    // Instantiate and open the connection
    cnnUserMan = new SqlConnection(STR_CONNECTION_STRING);
    cnnUserMan.Open();

    // Instantiate and initialize command
    cmmUser = new SqlCommand("uspGetUsersByLastName", cnnUserMan);
    cmmUser.CommandType = CommandType.StoredProcedure;
    // Instantiate, initialize and add parameter to command
    prmLastName = cmmUser.Parameters.Add("@strLastName", SqlDbType.VarChar,
        50);
```

```
17    // Indicate this is an input parameter
18    prmLastName.Direction = ParameterDirection.Input;
19    // Set the value of the parameter
20    prmLastName.Value = "Doe";
21
22    // Return all users with a last name of Doe
23    drdUser = cmmUser.ExecuteReader();
24 }
```

In Listing 6-3, a **SqlParameter** object specifies the input parameter of the stored procedure. On Lines 15 and 16, I ask the command object to create and associate a parameter with the `@strLastName` argument. The value of this parameter is set to "Doe", which effectively means that only rows containing a last name of Doe are returned.

As you can see, I have specified that the parameter is an input argument using the **ParameterDirection** enum, although you don't really have to, because this is the default. Don't worry too much about parameter and argument; they are essentially the same thing.

Creating a Stored Procedure with Arguments and Return Values

So far I have created stored procedures that return a single value or a result set (rows) and a stored procedure that takes an input argument. In many cases, this is all you want, but sometimes it's not enough. What if you want a value and a result set returned at the same time? Actually, you may want several values and a result set, but I'm sure you get the idea. In such instances, you can use output arguments.

Actually, you can return as many different values and result sets as you want by including multiple SELECT statements after the AS clause, but I personally think this approach looks messy. If I return rows and one or more values, I generally use OUTPUT arguments for the values. I guess to some extent this is a matter of preference. However, you should be aware that including an output parameter is a faster approach than having it returned in a **DataSet** object, but sometimes you might need the richer functionality of the **DataSet** class, once the values have been returned.

Instead of using the following example code to return a scalar value, two result sets, and another scalar value in that order:

```
...
AS
   SELECT 19
   SELECT * FROM tblUser
   SELECT * FROM tblUserRights
   SELECT 21
```

I would use something like this:

```
...
AS
   SELECT * FROM tblUser
   SELECT * FROM tblUserRights
```

The two return values should then be returned as OUTPUT arguments. But it's your call, my friend, as to which approach you prefer to use. Please note that OUTPUT arguments can also serve as INPUT arguments by default, meaning you can actually supply a value in the OUTPUT argument when calling the stored procedure, and get a different value back. Just like a value passed by reference from one procedure to another.

EXERCISE

Create a new stored procedure and save it with the name **uspGetUsersAndRights**. This stored procedure should return the value 55 for the OUTPUT argument `lngNumRows`, and then all rows in the tblUser table and all rows in the tblUserRights table.

The stored procedure should look like the one in Figure 6-6.

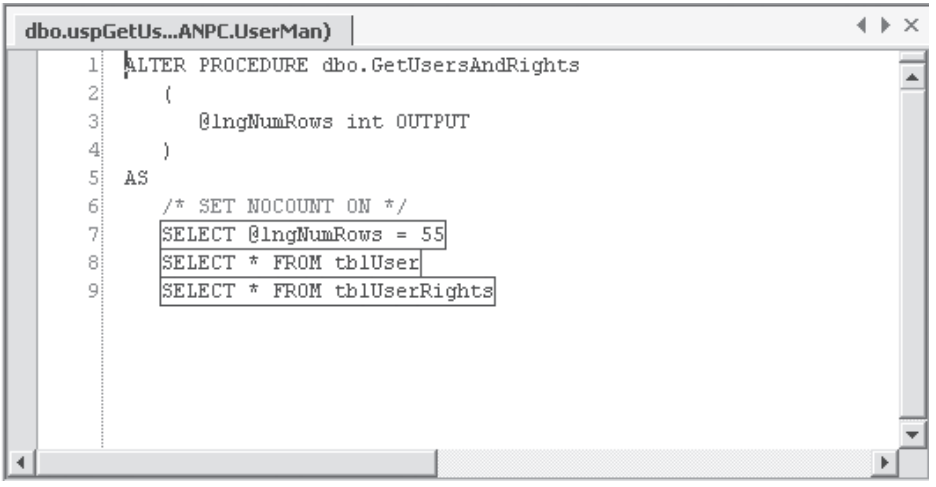

Figure 6-6. The `uspGetUsersAndRights` *stored procedure*

In the uspGetUsersAndRights stored procedure, shown in Figure 6-6, you can see that the @lngNumRows int argument is set to the value 55 on Line 7. However, using a default value for the argument you can achieve the same result, by changing Line 3 like this:

```
@lngNumRows int = 55 OUTPUT
```

This means that if for some reason you don't set the value of this parameter when calling the stored procedure or within the stored procedure itself, it'll return 55 as the output value. Default argument values also work for input arguments, and they're specified the same way, using the equal sign followed by the default value, right after the data type.

Running a Stored Procedure with Arguments and Return Values from the IDE

If you've created and saved the stored procedure in the previous exercise, test it by running it. You can try running the stored procedure from either the editor window or the Server Explorer window. The Output window, located just below the editor window, will display the output from the stored procedure, and it should look similar to the output in Figure 6-7.

> **NOTE** *Syntax testing of your stored procedure is done when you save it, and I have a feeling you have already encountered this. If not, just know that's how it is—syntax errors are caught when you try to save your stored procedure.*

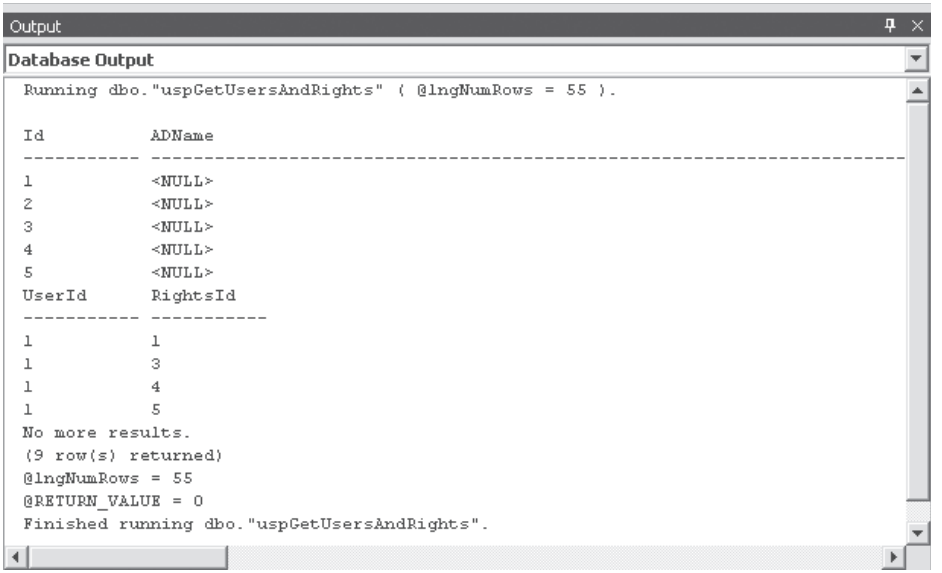

Figure 6-7. The Output window with output from the `uspGetUsersAndRights` *stored procedure*

Using a Stored Procedure with Arguments and Return Values

Listing 6-4 shows the code to execute the `uspGetUsersAndRights` stored procedure programmatically.

Listing 6-4. Retrieving Rows and Output Values from a Stored Procedure

```
public void GetUsersAndRights() {
    SqlConnection cnnUserMan;
    SqlCommand cmmUser;
    SqlDataReader drdUser;
    SqlParameter prmNumRows;

    // Instantiate and open the connection
    cnnUserMan = new SqlConnection(STR_CONNECTION_STRING);
    cnnUserMan.Open();

    // Instantiate and initialize command
    cmmUser = new SqlCommand("uspGetUsersAndRights", cnnUserMan);
    cmmUser.CommandType = CommandType.StoredProcedure;
    // Instantiate, initialize and add parameter to command
    prmNumRows = cmmUser.Parameters.Add("@lngNumRows", SqlDbType.Int);
    // Indicate this is an output parameter
```

```
17    prmNumRows.Direction = ParameterDirection.Output;
18    // Get first batch of rows (users)
19    drdUser = cmmUser.ExecuteReader();
20
21    // Display the last name of all user rows
22    while (drdUser.Read()) {
23       MessageBox.Show(drdUser["LastName"].ToString());
24    }
25
26    // Get next batch of rows (user rights)
27    if (drdUser.NextResult()) {
28       // Display the id of all rights
29       while (drdUser.Read()) {
30          MessageBox.Show(drdUser["RightsId"].ToString());
31       }
32    }
33 }
```

In Listing 6-4, two result sets are returned, and therefore I use the **NextResult** method of the DataReader class to advance to the second result set on Line 27. Otherwise this stored procedure works pretty much the same as one with input parameters, although the parameter direction is specified as an output on Line 17.

Retrieving a Value Specified with RETURN

In a stored procedure, you can use the RETURN statement to return a scalar value. However, this value cannot be retrieved using the **ExecuteScalar** method of the Command class, as it would when you use the SELECT statement (refer back to Figure 6-2). Of course there is a way of retrieving this value, which I show you after the following exercise.

EXERCISE

Create a new stored procedure and save it with the name **uspGetRETURN_VALUE**. This stored procedure should return the value 55 as the RETURN_VALUE. The stored procedure should look like the one in Figure 6-8.

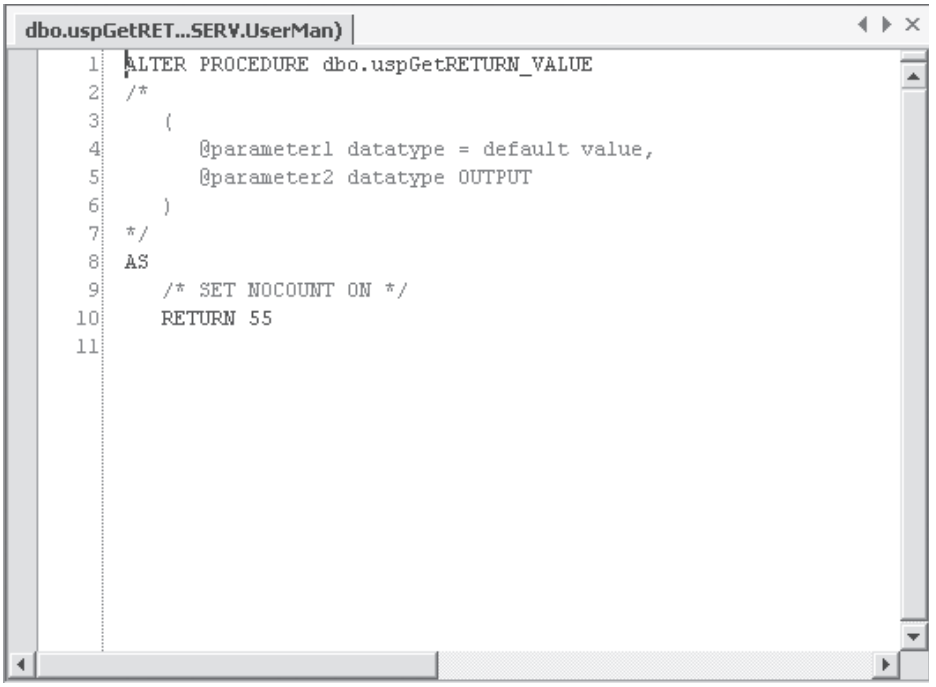

Figure 6-8. The uspGetRETURN_VALUE *stored procedure*

Listing 6-5 shows you how to retrieve the value from code.

Listing 6-5. Retrieving RETURN_VALUE from a Stored Procedure

```
public void GetRETURN_VALUE() {
    SqlConnection cnnUserMan;
    SqlCommand cmmUser;
    SqlParameter prmNumRows;
    object objResult;

    // Instantiate and open the connection
    cnnUserMan = new SqlConnection(STR_CONNECTION_STRING);
    cnnUserMan.Open();

```

```
    // Instantiate and initialize command
    cmmUser = new SqlCommand("uspGetRETURN_VALUE", cnnUserMan);
    cmmUser.CommandType = CommandType.StoredProcedure;
    // Instantiate, initialize and add parameter to command
    prmNumRows = cmmUser.Parameters.Add("@RETURN_VALUE", SqlDbType.Int);
    // Indicate this is a return value parameter
    prmNumRows.Direction = ParameterDirection.ReturnValue;
    // Get RETURN_VALUE like this, . . .
    objResult = cmmUser.ExecuteScalar();
    // or like this
    MessageBox.Show(prmNumRows.Value.ToString());
}
```

In Listing 6-5, the **ExecuteScalar** method gets the RETURN_VALUE from a stored procedure. Normally, you would use this method to return the value in the `lngResult` variable, but this variable will contain the default value, 0, in this case. However, because I have specified the **Direction** property of the `prmNumRows` parameter with the **ReturnValue** member of the **ParameterDirection** enum, I can simply look at the **Value** property of the parameter after executing the command.

Changing the Name of a Stored Procedure

If you change the name of your stored procedure in the editor window, the stored procedure is saved with the new name when you save (Ctrl+S). However, if you're not using this method to copy an existing stored procedure, you should be aware that the old stored procedure still exists. So you'll have to delete it if you don't want it.

Viewing Stored Procedure Dependencies

In SQL Server Enterprise Manager, you can see what tables and other objects your stored procedure uses or is dependent on. Open up Enterprise Manager, expand your SQL Server, expand databases and your database, select the Stored Procedures node, right-click the stored procedure you want to see the dependencies for, and select All Task Display Dependencies from the pop-up menu. This brings up the Dependencies dialog box, where you can see what database objects your stored procedure depends on and vice versa. This is also called the *ownership chain*.

Running Oracle Stored Procedures

Oracle stored procedures and stored functions are different from those of SQL Server. That's why I've chosen to explain the Oracle stored procedures in a separate section. When discussing what you can do with stored procedures/functions in Oracle compared to stored procedures in SQL Server, it's pretty much the same, but the implementation is quite different.

SQL Server stored procedures can return a value, just like a function in C#, whereas in Oracle you have stored procedures and stored functions. This means that if you want a return value that isn't a parameter, you must use a stored function. In this chapter, you won't see how to create stored procedures and stored functions in Oracle, but you can use the Oracle Database Project located in the example code, which you can download from the Apress Web site (`http://www.apress.com`) or the UserMan site (`http://www.userman.dk`), to create the tables, stored procedures, views, and triggers used by the Oracle example code. Please consult your Oracle documentation if you need more information on how to implement stored procedures and stored functions in Oracle.

When you use ADO.NET and ADO for that matter, you can't use the **ExecuteScalar** method of the DataReader class to retrieve a return value, as shown in Listings 6-1 and 6-5 and discussed in the "Retrieving a Value Specified with RETURN" section. This is also true if you execute a stored function. You need to return any return values in output parameters, just as I've demonstrated in Listing 6-4. If you only need to return a value, as in Listing 6-1, which is what the Oracle stored function in Listing 6-6 does, you can do as is shown in Listing 6-7, which is really more or less the same code as in Listing 6-1.

Listing 6-6. A Simple Oracle Stored Function

```
CREATE OR REPLACE FUNCTION SIMPLESTOREDFUNCTION
RETURN NUMBER
AS
    lngNumRows NUMBER;
BEGIN
    SELECT COUNT(*) INTO lngNumRows FROM TBLUSER;
    RETURN lngNumRows;
END SIMPLESTOREDFUNCTION;
```

In Listing 6-6, you can see an Oracle stored function that returns the number of rows in the tblUser table. You can see in Listing 6-7 how you can access this stored function and retrieve the return value.

Listing 6-7. Running a Simple Oracle Stored Function

```
 1 public void ExecuteSimpleOracleSF() {
 2     OleDbConnection cnnUserMan;
 3     OleDbCommand cmmUser;
 4     OleDbParameter prmNumRows;
 5     object objReturnValue;
 6
 7     // Instantiate and open the connection
 8     cnnUserMan = new OleDbConnection(STR_CONNECTION_STRING);
 9     cnnUserMan.Open();
10
11     // Instantiate and initialize command
12     cmmUser = new OleDbCommand("SimpleStoredFunction", cnnUserMan);
13     cmmUser.CommandType = CommandType.StoredProcedure;
14     // Instantiate output parameter and add to parameter
15     // collection of command object
16     prmNumRows = cmmUser.CreateParameter();
17     prmNumRows.Direction = ParameterDirection.ReturnValue;
18     prmNumRows.DbType = DbType.Int64;
19     prmNumRows.Precision = 38;
20     prmNumRows.Size = 38;
21     cmmUser.Parameters.Add(prmNumRows);
22
23     // Retrieve and display value
24     objReturnValue = cmmUser.ExecuteScalar();
25     MessageBox.Show(cmmUser.Parameters[0].Value.ToString());
26 }
```

In Listing 6-7, I've actually used the **ExecuteScalar** method of the DataReader class on Line 24, but if you look carefully, you'll see that I don't use the value returned from the function call (`objReturnValue`) as in Listing 6-1. However, I do retrieve the return value in the `prmNumRows` parameter, which is instantiated, initialized, and set up as a return value on Lines 12 through 21, and display it after executing the command on Line 25. I use **ExecuteScalar** method, because it has the least overhead of any of the **Execute** methods of the DataReader class. So even if you don't use the return value from the **ExecuteScalar** method, which is always **null** when calling an Oracle stored function or stored procedure, you can still get the return value from the stored function. The trick is add a parameter to the command object and make sure you set the **Direction** property of the parameter object to the **ReturnValue** member of the **ParameterDirection** enum, as is shown on Line 13.

If you want to use an Oracle stored procedure instead of a stored function, like the one shown in Listing 6-8, to retrieve one or more simple data types using output parameters, you can use the example code shown in Listing 6-9.

Listing 6-8. A Simple Oracle Stored Procedure

```
CREATE OR REPLACE PROCEDURE  SIMPLESTOREDPROCEDURE
(lngNumRows OUT NUMBER)
AS
BEGIN
    SELECT COUNT(*) INTO lngNumRows FROM TBLUSER;
END SIMPLESTOREDPROCEDURE;
```

The Oracle stored procedure in Listing 6-8 accepts one output parameter (`lngNumRows`) and sets this parameter to the number of rows in the tblUser table when executed. You can see how you can call this stored procedure from code, in Listing 6-9.

Listing 6-9. Running a Simple Oracle Stored Procedure

```
public void ExecuteSimpleOracleSP() {
    OleDbConnection cnnUserMan;
    OleDbCommand cmmUser;
    OleDbParameter prmNumRows;
    object objReturnValue;

    // Instantiate and open the connection
    cnnUserMan = new OleDbConnection(STR_CONNECTION_STRING);
    cnnUserMan.Open();

    // Instantiate and initialize command
    cmmUser = new OleDbCommand("SimpleStoredProcedure", cnnUserMan);
    cmmUser.CommandType = CommandType.StoredProcedure;
    // Instantiate output parameter and add to parameter
    // collection of command object
    prmNumRows = cmmUser.CreateParameter();
    prmNumRows.Direction = ParameterDirection.Output;
    prmNumRows.DbType = DbType.Int64;
    prmNumRows.Precision = 38;
    prmNumRows.Size = 38;
    cmmUser.Parameters.Add(prmNumRows);

    // Retrieve and display value
    objReturnValue = cmmUser.ExecuteScalar();
    MessageBox.Show(cmmUser.Parameters[0].Value.ToString());
}
```

In Listing 6-9, I again use the **ExecuteScalar** method of the DataReader class on Line 24 for retrieving a value from a stored procedure. The example code on Listing 6-9 really isn't all that different from Listing 6-7, but it does show you how to call a stored procedure instead of a stored function.

The Oracle stored procedures and stored functions, and the example code to execute them shown so far, only deal with simple return values. If you need to return result sets, such as in Listings 6-2, 6-3, and 6-4, you need to use cursors in the stored procedures.[4] Listing 6-10 shows a stored procedure that returns a result set using cursors.

Listing 6-10. Oracle Stored Procedure Returning Result Set

```
 1 CREATE OR REPLACE PACKAGE PKGTBLUSER
 2 AS
 3    TYPE CUR_TBLUSER IS REF CURSOR RETURN TBLUSER%ROWTYPE;
 4 END PKGTBLUSER;
 5
 6 CREATE OR REPLACE PROCEDURE USPGETUSERSBYLASTNAME
 7    (ROWS OUT PKGTBLUSER.CUR_TBLUSER, strLastName IN VARCHAR2)
 8 IS
 9 BEGIN
10 OPEN ROWS FOR SELECT * FROM TBLUSER
11 WHERE LASTNAME = strLastName;
12 END USPGETUSERSBYLASTNAME;
```

In Listing 6-10, you can see how I first create a package definition (Lines 1 through 4) in my Oracle database, and in this package I define the `CUR_TBLUSER` cursor type, which is of data type **REF CURSOR,**[5] that returns rows from the tblUser table. The package definition only holds the type declaration, which is used in the stored procedure. Please note that the notion of Oracle package definitions and package bodies are beyond the scope of this book, although you can certainly use a package body instead of the stored procedure shown on Lines 6 through 12. Please see your Oracle documentation for more information on packages.

You need to declare the cursor type in a package, because you're using it as the data type for one of the parameters in the `USPGETUSERSBYLASTNAME` stored procedure. You can't declare a data type in the parameters section of a stored

4. You can also use cursors with a stored function, but because I won't be using the function return value from this point on, I'll concentrate on using stored procedures.
5. This is short for REFERENCE CURSOR, and basically it's used as a pointer to the original data. Please see your Oracle documentation for more information.

procedure, which is why you need it declared elsewhere. If you look at the parameters declaration on Line 7, you can see that I need to use the full path to the data type, PKGTBLUSER.CUR_TBLUSER. Lines 10 and 11 of Listing 6-10 is where the rows that match the passed last name criterion, are retrieved with the CUR_TBLUSER cursor and saved in the ROWS OUT parameter. Listing 6-11 shows you how to retrieve the result set from the stored procedure.

Listing 6-11. Retrieving Result Set from Oracle Stored Procedure

```
 1 public void OracleGetUsersByLastName() {
 2     OleDbConnection cnnUserMan;
 3     OleDbCommand cmmUser;
 4     OleDbParameter prmLastName;
 5     OleDbDataReader drdUser;
 6
 7     // Instantiate and open the connection
 8     cnnUserMan = new OleDbConnection(STR_CONNECTION_STRING);
 9     cnnUserMan.Open();
10
11     // Instantiate and initialize command
12     cmmUser = new OleDbCommand("USPGETUSERSBYLASTNAME", cnnUserMan);
13     cmmUser.CommandType = CommandType.StoredProcedure;
14     // Instantiate, initialize and add parameter to command
15     prmLastName = cmmUser.Parameters.Add("strLastName", OleDbType.VarChar,
16        50);
17     // Indicate this is an input parameter
18     prmLastName.Direction = ParameterDirection.Input;
19     // Set the type and value of the parameter
20     prmLastName.Value = "Doe";
21
22     // Retrieve rows
23     drdUser = cmmUser.ExecuteReader();
24     // Loop through the returned rows
25     while (drdUser.Read()) {
26        // Display the last name of all user rows
27        MessageBox.Show(drdUser["LastName"].ToString());
28     }
29 }
```

In Listing 6-11, you can see how I set up the command object on Lines 12 through 13, and then prepare the prmLastName input parameter with the value of "Doe". I then call the **ExecuteReader** method of the Command class, which returns the DataReader with all users with a last name of Doe. If you compare the stored procedure in Listing 6-10 and the example code in Listing 6-11, you'll see

that there's a mismatch of the number of parameters. The stored procedure has two parameters, the last name input parameter and the result set output parameter. However, I only set up one parameter in Listing 6-11 and that's the last name input parameter. The command object takes care of returning the result set as the return value of the function call (**ExecuteReader**) instead of as an output parameter. It almost works the same as with the SQL Server example code in Listing 6-3.

> **NOTE** *It doesn't matter where you place the* `ROWS OUT` *parameter in the stored procedure parameter declaration—that is, whether you place it first as is done in Listing 6-10, or last like this:*
> `(strLastName IN VARCHAR2, ROWS OUT PKGTBLUSER.CUR_TBLUSER)`

There are other ways of calling a stored procedure in your Oracle database, such as using the `ODBC {call storedprocedurename}` syntax, but I've chosen to show you the way that looks and feels as close to the one used for calling SQL Server procedures.

Using Views

A *view* is, as the word suggests, a display of data in your database. Perhaps it helps to think of a view as a virtual table. It can be a subset of a table or an entire table, or it can be a subset of several joined tables. Basically, a view can represent just about any subset of data in your database, and you can include other views in a view. Including a view in another view is called *nesting*, and it can be a valuable way of grouping display data. However, nesting too deeply can also result in performance problems and can certainly make it a real challenge to track down errors. There isn't really any magic to a view or any big secrets that I can let you in on; it's simply just a great tool for manipulating your data. In the rest of this section, I am going to look at why, when, where, and how you should use a view.

> **NOTE** *The example code shown in this section is SQL Server only, but if you take a look at the accompanying example code, you'll see that it works exactly the same with Microsoft Access queries. Only the SQL Server .NET Data Provider has been changed to the OLE DB .NET Data Provider. The same goes for Oracle views. See the example code, which is almost identical to the Microsoft Access code. Views aren't supported in MySQL 3.23.45.*

View Restrictions

A view is almost identical to a row-returning query, with just a few exceptions. Some of the restrictions are detailed here:

- COMPUTE and COMPUTE BY clauses cannot be included in your view.
- ORDER BY clauses aren't allowed in a view, unless you specify the TOP clause as part of the SELECT statement. However, you can index a view with SQL Server 2000.
- The INTO keyword cannot be used to create a new table.
- Temporary tables cannot be referenced.

There are other restrictions, so please check with your SQL Server documentation and/or Help Files.

Why Use a View?

Like stored procedures, views are used for server-side processing of your data, but whereas stored procedures mainly are used for security and performance reasons, views are generally used to secure access to your data and to hide complexity of queries that contain many joins. You may want to use a view for a variety of reasons:

- *Security:* You don't want your users to access the tables directly, and with the help of a view you can restrict users to seeing only the parts of a table they are allowed to see. You can restrict access to specific columns and/or rows and thus make it easy for your users to use for their own queries.
- *Encryption:* You can encrypt a view so that no one can see where the underlying data comes from. Mind you, this is an irreversible action, meaning that the textual SQL statements that form the view can't be retrieved again!
- *Aggregated data:* Views are often used on large scale systems to provide aggregated data.

There are other reasons for creating a view, but the mentioned reasons are certainly two of the most common.

Creating a View

It's easy to create a view. If you are used to working with the SQL Server's **Server Manager**, you should check out what the **Server Explorer** has to offer you. Here's how you create a view using the UserMan database as an example:

1. Open up the Server Explorer window.
2. Expand the UserMan database on your database server.
3. Right-click the Views node and select New View.

This brings up the View Designer, which in fact is the same as the Query Designer.

> **NOTE** *The Query Designer is described in detail in Chapter 4. Although the View Designer and Query Designer have the same look and feel, you cannot create views that don't adhere to the view restrictions mentioned in the section, "View Restrictions."*

The Add Table dialog box is also shown when the View Designer is displayed. In this dialog box, simply select the tables you want to retrieve data from and click Add. Click Close when all the required tables have been added.

As you start selecting in the Diagram pane the columns that should be output when the view is run, the SQL pane and the Grid pane change accordingly. When you are done selecting the columns to output, you should save the view using Ctrl+S.

EXERCISE

1) Create a new view. This view should contain the following tables: tblUser, tblRights, and tblUserRights. The following fields should be output: tblUser.LoginName, tblUser.FirstName, tblUser.LastName, and tblRights.Name.

2) Save the view under the name **viwUserInfo**. The new view should look like the one in Figure 6-9.

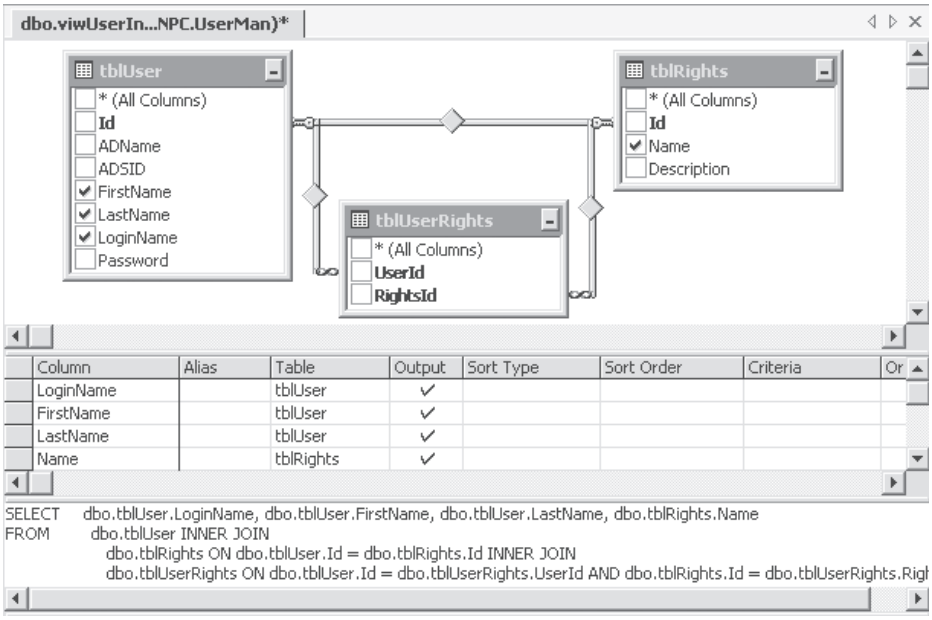

Figure 6-9. The `viwUserInfo` *view*

Running a View from the IDE

"Running a view" is perhaps not the most appropriate phrase when you think about it. On the other hand, the view does have to retrieve the data from all the tables referenced in the view, so I guess this phrase will have to do.

Anyway, you can run a view from the View Designer by right-clicking a blank area of the View Designer and selecting Run from the pop-up menu. The data retrieved by the view is then displayed in the Results pane of the View Designer.

EXERCISE

1) Run the `viwUserInfo` view from the View Designer. The Results pane now displays rows like the ones in Figure 6-10.

	LoginName	FirstName	LastName	Name
▶	UserMan	John	Doe	AddUser
*				

Figure 6-10. The results of running the `viwUserInfo` *view*

2) Notice that the Name field of the tblRights table seems a bit confusing, because it doesn't really show what Name means. So in the Grid pane, you should add the text **RightsName** to the Alias column in the Name row.

3) Run the view again and notice how the new column name appears in the Results pane.

4) Save the view with Ctrl+S.

Using a View from Code

Actually, it's very easy to use a view in code, because a view is referenced like any standard table in your database, which means that you can retrieve data using a command object or a data adapter that fills a data set, and so on.

> **NOTE** *Please see Chapter 3B for specific information on how to manipulate data in a table.*

Retrieving Read-Only Data from a View in Code

The simplest use of a view is for display purposes, like when you just need to display some information from one or more related tables. Because in the example code I don't have to worry about updates, I don't have to set up anything particular. Listing 6-12 demonstrates how to return all rows from a view and populate a data reader.

Listing 6-12. Retrieving Rows in a View

```
public void RetrieveRowsFromView() {
    SqlConnection cnnUserMan;
    SqlCommand cmmUser;
    SqlDataReader drdUser;

    // Instantiate and open the connection
    cnnUserMan = new SqlConnection(STR_CONNECTION_STRING);
    cnnUserMan.Open();

    // Instantiate and initialize command
    cmmUser = new SqlCommand("SELECT * FROM viwUserInfo", cnnUserMan);
    // Get rows
    drdUser = cmmUser.ExecuteReader();
}
```

Listing 6-12 is just like any other row-returning query, except that a view is queried instead of a table.

Manipulating Data in a View from Code

Listing 6-12 shows you how to retrieve data from a view into a data reader, and this means the data cannot be updated, because the data reader doesn't allow updates. However, it's possible to update data in a view. The problem with this is that various versions of SQL Server support different levels of update support for views. If you only have one table in a view, this isn't a problem at all; however, if you have multiple tables in a view, only SQL Server 2000 supports updating rows in more than one of the source tables. Besides the mentioned problems, you'll certainly run into even bigger problems if you need to migrate to a different RDBMS. Generally, I would discourage updating data in views.

EXERCISE

1) Create a new view. This view should contain the tblUser table. The following fields should be output: Id, FirstName, LastName, and LoginName.

2) Save the view under the name **viwUser**. The new view should look like the one in Figure 6-11.

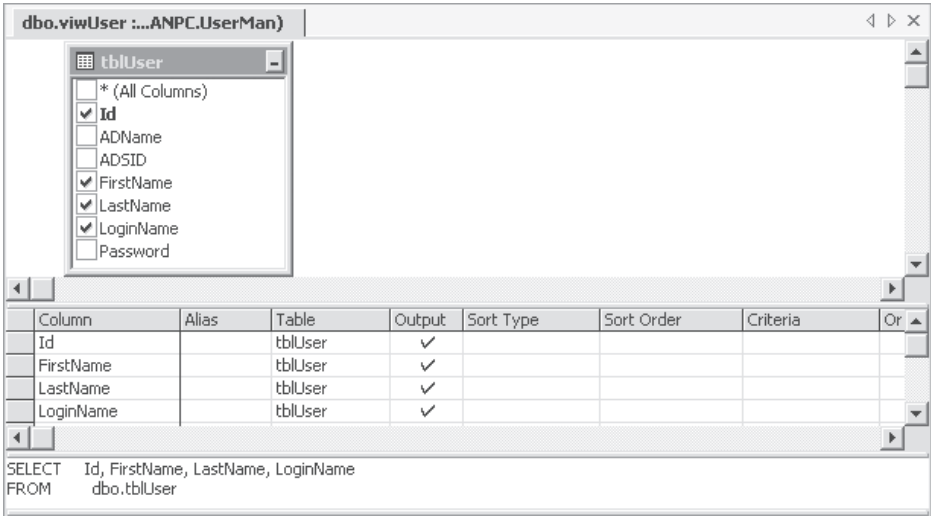

Figure 6-11. The `viwUser` *view*

This view, which can be located on SQL Server 7.0 as well as SQL Server 2000, can be manipulated using the code in Listing 6-13.

Listing 6-13. Manipulating Data in a View Based on a Single Table

```
public void ManipulatingDataInAViewBasedOnSingleTable() {
    const string STR_SQL_USER_SELECT = "SELECT * FROM viwUser";
    const string STR_SQL_USER_DELETE = "DELETE FROM viwUser WHERE Id=@Id";
    const string STR_SQL_USER_INSERT = "INSERT INTO viwUser(FirstName, " +
        "LastName, LoginName, Logged, Description) VALUES(@FirstName, " +
        "@LastName, @LoginName)";
    const string STR_SQL_USER_UPDATE = "UPDATE viwUser SET FirstName=" +
        "@FirstName, LastName=@LastName, LoginName=@LoginName WHERE Id=@Id";

    SqlConnection cnnUserMan;
    SqlCommand cmmUser;
```

```
12    SqlDataAdapter dadUser;
13    DataSet dstUser;
14
15    SqlCommand cmmUserSelect;
16    SqlCommand cmmUserDelete;
17    SqlCommand cmmUserInsert;
18    SqlCommand cmmUserUpdate;
19
20    SqlParameter prmSQLDelete, prmSQLUpdate;
21
22     // Instantiate and open the connection
23    cnnUserMan = new SqlConnection(STR_CONNECTION_STRING);
24    cnnUserMan.Open();
25
26    // Instantiate and initialize command
27    cmmUser = new SqlCommand("SELECT * FROM viwUser", cnnUserMan);
28    // Instantiate the commands
29    cmmUserSelect = new SqlCommand(STR_SQL_USER_SELECT, cnnUserMan);
30    cmmUserDelete = new SqlCommand(STR_SQL_USER_DELETE, cnnUserMan);
31    cmmUserInsert = new SqlCommand(STR_SQL_USER_INSERT, cnnUserMan);
32    cmmUserUpdate = new SqlCommand(STR_SQL_USER_UPDATE, cnnUserMan);
33    // Instantiate command and data set
34    cmmUser = new SqlCommand(STR_SQL_USER_SELECT, cnnUserMan);
35    dstUser = new DataSet();
36
37    dadUser = new SqlDataAdapter();
38    dadUser.SelectCommand = cmmUserSelect;
39    dadUser.InsertCommand = cmmUserInsert;
40    dadUser.DeleteCommand = cmmUserDelete;
41    dadUser.UpdateCommand = cmmUserUpdate;
42
43    // Add parameters
44     prmSQLDelete = dadUser.DeleteCommand.Parameters.Add("@Id", SqlDbType.Int,
45       0, "Id");
46    prmSQLDelete.Direction = ParameterDirection.Input;
47    prmSQLDelete.SourceVersion = DataRowVersion.Original;
48
49    cmmUserUpdate.Parameters.Add("@FirstName", SqlDbType.VarChar, 50,
50       "FirstName");
51    cmmUserUpdate.Parameters.Add("@LastName", SqlDbType.VarChar, 50,
52       "LastName");
53    cmmUserUpdate.Parameters.Add("@LoginName", SqlDbType.VarChar, 50,
54       "LoginName");
```

```
55     prmSQLUpdate = dadUser.UpdateCommand.Parameters.Add("@Id", SqlDbType.Int,
56        0, "Id");
57     prmSQLUpdate.Direction = ParameterDirection.Input;
58     prmSQLUpdate.SourceVersion = DataRowVersion.Original;
59
60     cmmUserInsert.Parameters.Add("@FirstName", SqlDbType.VarChar, 50,
61        "FirstName");
62     cmmUserInsert.Parameters.Add("@LastName", SqlDbType.VarChar, 50,
63        "LastName");
64     cmmUserInsert.Parameters.Add("@LoginName", SqlDbType.VarChar, 50,
65        "LoginName");
66
67     // Populate the data set from the view
68     dadUser.Fill(dstUser, "viwUser");
69
70     // Change the last name of user in the second row
71     dstUser.Tables["viwUser"].Rows[1]["LastName"] = "Thomsen";
72     dstUser.Tables["viwUser"].Rows[1]["FirstName"] = "Carsten";
73     // Propagate changes back to the data source
74     dadUser.Update(dstUser, "viwUser");
75 }
```

In Listing 6-13, a data adapter and a data set were set up to retrieve and hold data from the `viwUser` view. The LastName column of row 2 is then updated as well as the data source with the changes in the data set. This simple demonstration was designed to show you how to work with views based on a single table.

Using Triggers

A *trigger* is actually a stored procedure that automatically invokes (triggers) when a certain change is applied to your data. Triggers are the final server-side processing functionality that I'll discuss in this chapter. Until SQL Server 2000 was released, triggers were a vital part of enforcing referential integrity, but with the release of SQL Server 2000, you now have that capability built in. In the rest of this section, I'll show you what a trigger is and when and how you can use it, but there is little C# programming involved with using triggers, because they operate entirely internally, only passing status or error indicators back to the client.

Triggers respond to data modifications using INSERT, UPDATE, and DELETE operations. Basically, you can say that a trigger helps you write less code; you can incorporate business rules as triggers and thus prevent the inclusion of data that is invalid because it violates your business rules.

SQL Server implements AFTER triggers, meaning that the trigger is invoked after the modification has occurred. However, this doesn't mean that a change can't be rolled back, because the trigger has direct access to the modified row and as such can roll back any modification. When SQL Server 2000 was released you also got support for the notion of BEFORE triggers, which you might know from the Oracle RDBMS. In SQL Server 2000, they are called INSTEAD OF triggers.

> **NOTE** *The example code shown in this section is SQL Server only, but if you take a look at the accompanying example code, you'll see that Oracle after triggers work almost the same as SQL Server triggers, although the syntax is different. Only the SQL Server .NET Data Provider has been changed to the OLE DB .NET Data Provider. Triggers aren't supported in Microsoft Access or MySQL 3.23.45.*

Why Use a Trigger?

Triggers are automatic, so you don't have to apply the business logic in your code. Here's a perfect situation for a business rule: you need to check if a member of an organization has paid his or her annual fee and therefore is allowed to order material from the organization's library. An INSERT trigger could perform the lookup in the members table when a member tries to order material and check if the member has paid the annual fee. This is exactly what makes a trigger more useful than a constraint in some situations, because a trigger can access columns in other tables, unlike a constraint, which can only access columns in the current table or row. If your code is to handle your business rule, this would mean that you need to look up the member's information in the members table before you can insert the order in the orders table. With the trigger, this lookup is done automatically, and an exception is thrown if you try to insert an order for library material if the member hasn't paid his or her annual fee. Furthermore, you don't have to rely on another front-end code developer to know what the business rules are.

In short, use a trigger for keeping all your data valid or to comply with your business rules. Think of triggers as an extra validation tool, while at the same time making sure you have set up referential integrity.

> **NOTE** *With SQL Server 2000, you shouldn't use triggers for referential integrity (see Chapter 2), because you can set that up with the Database Designer. See Chapter 4 for information on the Database Designer.*

Creating a Trigger

It's quite easy to create a trigger. This can be done using the Server Manager that comes with SQL Server, but I'll use the Server Explorer. Here's how you create a trigger for the example UserMan database:

1. Open up the Server Explorer window.

2. Expand the UserMan database on your database server.

3. Expand the Tables node.

4. Right-click the table for which you want to create a trigger and select New Trigger from the pop-up menu.

This brings up the trigger text editor, which is more or less the same editor you use for your C# code (see Figure 6-12).

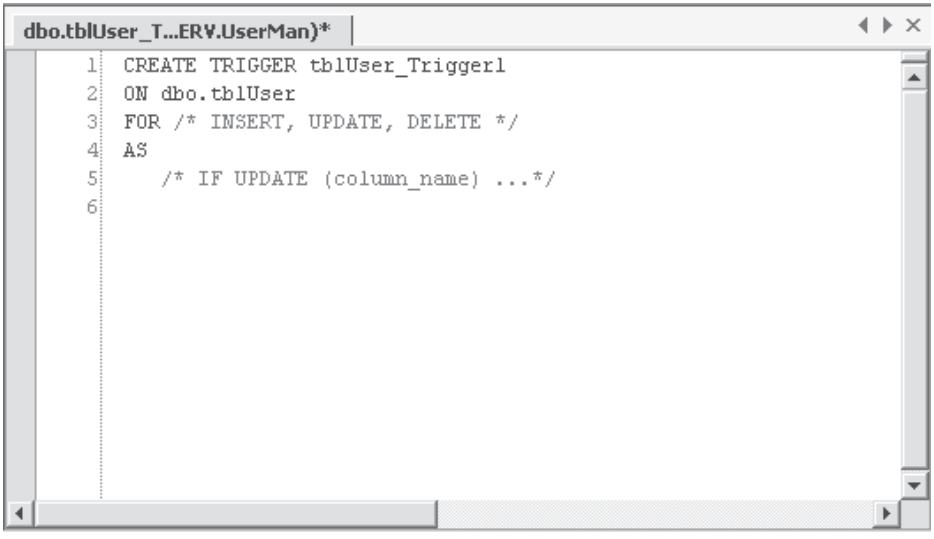

Figure 6-12. Trigger editor with default template

In the trigger editor, you can see that the template automatically names a new trigger `Trigger1` prefixed with the name of the table. Actually, if another trigger with this name already exists, the new trigger is named `Trigger2`, and so on.

Once you are done editing your trigger, you need to save it by pressing Ctrl+S. As soon as you have saved it, the very first line of the stored procedure changes. The SQL statement CREATE TRIGGER is changed so that the first line reads as follows:

```
ALTER TRIGGER dbo. . . .
```

> **NOTE** *The trigger editor performs syntax checking when you save your trigger, meaning if the syntax of your trigger is invalid, you aren't allowed to save it to your database.*

EXERCISE

1) Create a new trigger for the tblUser table and save it with the name **tblUser_Update**.

2) This is an update trigger, so you need to change the text on Line 3 to **FOR UPDATE**.

3) Replace the text on Line 5 and down with the following:

```
DECLARE @strFirstName varchar(50)
/* Get the value for the FirstName column */
SELECT @strFirstName = (SELECT FirstName FROM inserted)
/* Check if we're updating the LastName column.
If so, make sure FirstName is not NULL */
IF UPDATE (LastName) AND @strFirstName IS NULL
BEGIN
  /* Roll back update and raise exception */
  ROLLBACK TRANSACTION
  RAISERROR ('You must fill in both LastName and FirstName', 11, 1)
END
```

Now the stored procedure should look like the one in Figure 6-13.

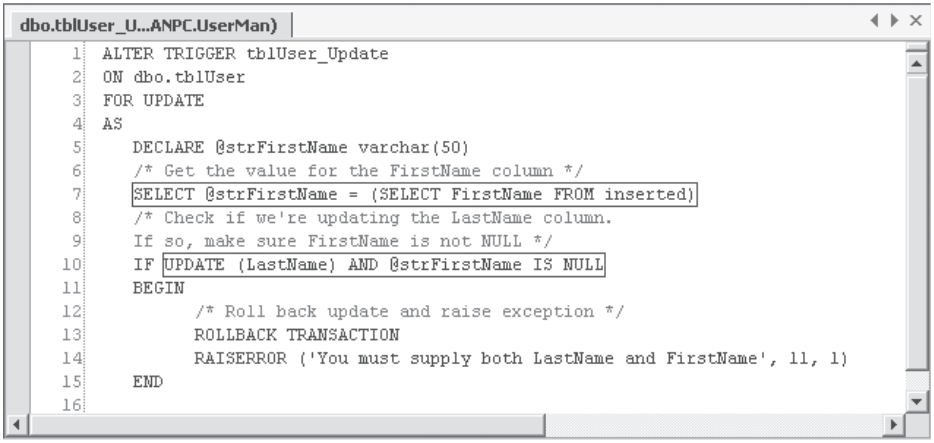

Figure 6-13. The `tblUser_Update` *trigger*

4) Don't forget to save the changes using Ctrl+S.

When a trigger has been saved to the database, you can locate it under the table to which it belongs in the **Server Explorer**.

The `tblUser_Update` trigger is invoked when updating a row in the user table. The trigger first tests to see if the LastName column is updated. If it is, then the trigger checks to see if the FirstName column is empty, because if it is the update is rolled back and an exception is raised. Please note that this trigger is designed to work with only one updated or inserted row at a time. If more rows are inserted at the same time, the trigger will have to be redesigned to accommodate this. However, this trigger only serves as a demonstration. The functionality of this trigger can easily be implemented using constraints, because the check I perform is done in the same table. If I had looked up a value in a different table, then the trigger would be your only choice.

Please see your SQL Server documentation if you need more information on how to create triggers. Listing 6-14 shows you how to execute the new trigger, demonstrating how to raise an exception you can catch in code.

Listing 6-14. Invoking Trigger and Catching Exception Raised

```
public void TestUpdateTrigger() {
    const string STR_SQL_USER_SELECT = "SELECT * FROM tblUser";
    const string STR_SQL_USER_DELETE = "DELETE FROM tblUser WHERE Id=@Id";
    const string STR_SQL_USER_INSERT = "INSERT INTO tblUser(FirstName, " +
       "LastName, LoginName) VALUES(@FirstName, @LastName, @LoginName)";
    const string STR_SQL_USER_UPDATE = "UPDATE tblUser SET " +
       FirstName=@FirstName, LastName=@LastName, LoginName=@LoginName WHERE" +
       " Id=@Id";

    SqlConnection cnnUserMan;
    SqlCommand cmmUser;
    SqlDataAdapter dadUser;
    DataSet dstUser;

    SqlCommand cmmUserSelect;
    SqlCommand cmmUserDelete;
    SqlCommand cmmUserInsert;
    SqlCommand cmmUserUpdate;

    SqlParameter prmSQLDelete, prmSQLUpdate, prmSQLInsert;

    // Instantiate and open the connection
    cnnUserMan = new SqlConnection(STR_CONNECTION_STRING);
    cnnUserMan.Open();

    // Instantiate and initialize command
    cmmUser = new SqlCommand("SELECT * FROM tblUser", cnnUserMan);
    // Instantiate the commands
    cmmUserSelect = new SqlCommand(STR_SQL_USER_SELECT, cnnUserMan);
    cmmUserDelete = new SqlCommand(STR_SQL_USER_DELETE, cnnUserMan);
    cmmUserInsert = new SqlCommand(STR_SQL_USER_INSERT, cnnUserMan);
    cmmUserUpdate = new SqlCommand(STR_SQL_USER_UPDATE, cnnUserMan);
    // Instantiate command and data set
    cmmUser = new SqlCommand(STR_SQL_USER_SELECT, cnnUserMan);
    dstUser = new DataSet();

    dadUser = new SqlDataAdapter();
    dadUser.SelectCommand = cmmUserSelect;
    dadUser.InsertCommand = cmmUserInsert;
    dadUser.DeleteCommand = cmmUserDelete;
    dadUser.UpdateCommand = cmmUserUpdate;

```

```
43     // Add parameters
44     prmSQLDelete = dadUser.DeleteCommand.Parameters.Add("@Id", SqlDbType.Int,
45        0, "Id");
46     prmSQLDelete.Direction = ParameterDirection.Input;
47     prmSQLDelete.SourceVersion = DataRowVersion.Original;
48
49     cmmUserUpdate.Parameters.Add("@FirstName", SqlDbType.VarChar, 50,
50        "FirstName");
51     cmmUserUpdate.Parameters.Add("@LastName", SqlDbType.VarChar, 50,
52        "LastName");
53     cmmUserUpdate.Parameters.Add("@LoginName", SqlDbType.VarChar, 50,
54        "LoginName");
55     prmSQLUpdate = dadUser.UpdateCommand.Parameters.Add("@Id", SqlDbType.Int,
56        0, "Id");
57     prmSQLUpdate.Direction = ParameterDirection.Input;
58     prmSQLUpdate.SourceVersion = DataRowVersion.Original;
59
60     cmmUserInsert.Parameters.Add("@FirstName", SqlDbType.VarChar, 50,
61        "FirstName");
62     cmmUserInsert.Parameters.Add("@LastName", SqlDbType.VarChar, 50,
63        "LastName");
64     cmmUserInsert.Parameters.Add("@LoginName", SqlDbType.VarChar, 50,
65        "LoginName");
66
67     // Populate the data set from the view
68     dadUser.Fill(dstUser, "tblUser");
69
70     // Change the name of user in the second row
71     dstUser.Tables["tblUser"].Rows[1]["LastName"] = "Thomsen";
72     dstUser.Tables["tblUser"].Rows[1]["FirstName"] = null;
73
74     try {
75        // Propagate changes back to the data source
76        dadUser.Update(dstUser, "tblUser");
77     }
78     catch (Exception objE) {
79        MessageBox.Show(objE.Message);
80     }
81  }
```

In Listing 6-14, the second row is updated, and the LastName column is set to “Thomsen” and the FirstName to a **null** value. This will invoke the update trigger that throws an exception, which is caught in code and displays the error message.

This should give you a taste for using triggers, and they really aren't that hard to work with. Just make sure you have a well-designed database that doesn't use triggers for purposes that can easily be achieved by other means such as referential integrity.

Viewing Trigger Source

In SQL Server Enterprise Manager, you can see the source for your triggers. Open up Enterprise Manager, expand your SQL Server, and expand databases and your database. Next, select the Tables node, right-click the table you have created the trigger for, and select All Task Manage Triggers from the pop-up menu. This brings up the Triggers Properties dialog box, where you can see and edit the triggers for the selected table.

In the Server Explorer in the VS .NET IDE, you can also view the trigger source by expanding your database, expanding the Tables node, expanding the table with the trigger, and double-clicking the trigger.

Summary

In this chapter, I discussed how to create various server-side objects for server-side processing of your data. I demonstrated stored procedures, views, and triggers, and showed you how to create, run, and execute a stored procedure from code; how to create, run, and use a view from code, including updating the view; and finally how to create triggers.

I went into enough details about stored procedures, views, and triggers as to what a C# programmer needs to know, but if you are also responsible for coding the SQL Server database and you need more information and example code, I can certainly recommend you read this book:

- *Code Centric: T-SQL Programming with Stored Procedures and Triggers,* by Garth Wells. Published by Apress, February 2001. ISBN: 1893115836.

The next chapter is about hierarchical databases. I'll discuss how you use the LDAP protocol to access a network directory database like the Active Directory and you'll see how to access information stored on Exchange Server 2000.

Index

Symbols and Numbers

\ (backslash) in Active Directory hierarchies, 420
* (asterisks), advisory about, 434
< (less than) column comparison symbol, role in Query Designer join types, 323
<= (less than or equal to) column comparison symbol, role in Query Designer join types, 323
<> (not equal to) column comparison symbol, role in Query Designer join types, 323
= (equal to) column comparison symbol, role in Query Designer join types, 323
== operator, using in ADO.NET connected layer, 87
> (greater than) column comparison symbol, role in Query Designer join types, 323
>= (greater than or equal to) column comparison symbol, role in Query Designer join types, 323
1NF (First Normal Form)
  achieving, 47
  checking against UserMan database schema, 53
2NF (Second Normal Form)
  checking against UserMan database schema, 53
  conforming to, 48
3NF (Third Normal Form)
  achieving, 48–49
  checking against UserMan database schema, 53
4NF (Fourth Normal Form)
  checking against UserMan database schema, 53
  conforming to, 49–50
5NF (Fifth Normal Form)
  achieving, 51
  checking against UserMan database schema, 53
7304 and 7309 error messages, troubleshooting in Microsoft Exchange Server 2000, 445

A

Aborted member of **MessageQueueTransactionStatus** Enum, description of, 483
abstract keyword in C#, relationship to OOP, 510–512
AcceptChanges
  DataRow class property, 246
  DataSet class method, 199, 215
  DataTable class method, 221
AcceptChangesDuringFill DataAdapter class property, description of, 172
access control, securing message queuing with, 499–502
accessibility of databases, understanding, 32
ACL (Access Control List), role in securing message queuing, 499–500
Active Directory
  accessing programmatically, 415
  accessing with OLE DB .NET Data Provider, 429–436
  binding to objects in, 421–422
  checking for existence of properties with, 426
  editing exiting properties with, 427
  examining **System.DirectoryServices** namespace in, 415–417
  exploring, 414–415
  identifying objects in, 420
  introduction to, 413
  manipulating object property values with, 426–429
  manipulating user properties with, 428–429
  retrieving SID for users with, 435–436
  searching for objects of specific classes in, 423–424
  searching objects in, 422–425

Active Directory *(continued)*
specifying information for retrieval with OLE DB .NET data providers, 431–432
studying **DirectoryEntry** class in, 417–422
updating, 429
updating cache after making changes to, 426
validating users with, 585
Web site for, 415, 434
Active Directory class, calling for UserMan example application, 581
Active Directory connections
specifying domains for LDAP access, 431
specifying OLE DB providers for, 430–431
Active Directory nodes or objects, returning nondefault properties from, 424–425
Active Directory objects
adding to UserMan example application, 579–581
updating, 435
Add Database Reference dialog box, displaying, 297–298
Add Server dialog box, displaying, 286
Add Table dialog box, displaying in Database Designer, 307
AddNew method
of **CurrencyManager** class, 540
of **DataView** class, 240
Addr or Address **ConnectionString** property value, details of, 66–67
Admin Queue, receiving message rejection notification in, 493–494
ADName column in tblUser column for CUser class, details of, 514, 516
ADO (Active Data Objects)
accessing ODBC data sources in, 59
explanation of, 55
resources for, 56
using with ADO.NET disconnected layer, 276–278
versus ADO.NET, 56, 59
ADO **Recordset** classes, populating **DataSet** objects with, 205–206
ADO.NET (Active Data Objects.NET)
versus ADO, 56, 59
resources for, 57
role of data-related namespaces in, 58–59
ADO.NET connected layer
clearing OLE DB object pooling in, 93–94
closing connections in, 86
command class methods in, 138–139
command class properties in, 129–137
comparing connection objects in, 87–88
comparing **State** to **ConnectionState** in, 89
comparing with **ReferenceEquals** method in, 88
data providers and drivers in, 59–63
data providers in, 59
data-related namespaces in, 58–59
declaring and instantiating **XmlReader** objects in, 168
defining transaction boundaries in, 99
disabling ODBC connection pooling in, 94–96
disabling OLE DB connection pooling in, 93
disposing of connections in, 86–87
DNs and RDNs in, 420
drivers in, 59
executing commands in, 139–142
handling Command class exceptions in, 143–146
handling Connection class exceptions in, 110–123
handling connection state changes in, 89–90
handling fill operation errors in, 184–185
handling messages in, 97–98
handling row updates in, 185–188
introduction to, 55
manipulating connection state in, 88–96
opening connections in, 85–86
pooling connections for **SqlConnection**, 92–93
pooling connections in, 90–96
role of Connection class in, 63–64
role of **ConnectionString** property in, 65–76
role of transactions in, 98
setting command properties in, 178–184
support for manual transactions, 99
using == operator in, 87
using command objects in, 123
using CommandBuilder class in, 188–191

using Connection class events with, 85–86
using Connection class methods with, 80–84
using Connection class properties with, 76–80
using DataReader class with, 146
using **Equals** method in, 87–88
using members of **ConnectionState** Enum with, 89
ADO.NET disconnected layer
introduction to, 193
role of data source locking in, 265–276
using classic ADO and COM Interop with, 276–278
using cursors in, 261–265
using **DataColumn** class in, 249–252
using **DataRelation** class in, 252–261
using **DataRow** class in, 245–249
using **DataSet** class with, 193–194
using **DataTable** class with, 218–223
using **DataView** class in, 238–245
using XML with **DataSet** class in, 195–196
ADSID column in tblUser column for CUser class, details of, 514
adsPath, retrieving from Active Directory with OLE DB .NET data providers, 432
AFTER triggers, functionality of, 406
aggregated data, using views for, 398
Alias grid pane column in Query Designer, 326
AllowDBNull DataColumn class property, description of, 250
AllowDelete DataView class property, description of, 239
AllowEdit DataView class property, description of, 239
AllowNew DataView class property, description of, 239
Alt+F11 keyboard shortcut, accessing Macros IDE with, 22
anomalies in normal forms, explanation of, 46
Antechinus C# Programming Editor, Web site for, 15
Append grid pane column in Query Designer, 326
Application Name **ConnectionString** property value, details of, 66–67
ApplicationException class, explanation of, 356
ApplyDefaultSort DataView class property, description of, 239
Apress Web site
accessing, 57
downloading UserMan classes from, 567
ArgumentException exception, 351
handling for DataReader, 160
throwing, 113
thrown by **CommandTimeout** property, 143
thrown by **CommandType** property, 144
thrown by **UpdatedRowSource** property, 145–146
ArgumentNullException exception, 351, 353
ArgumentOutOfRangeException exception, 170, 351, 354
arguments
advisory about usage with **SqlXmlAdapter** and **SqlXmlCommand** classes, 617
creating stored procedures with, 382–384
using stored procedures with, 384–385
arguments and return values, running stored procedures with, 385–387
ASP (Active Server Pages), evolution of, 5
ASP.NET data binding, maintaining state in, 548–549
ASP.NET, explanation of, 5
ASP.NET server controls, binding to data sources, 545–548
assemblies
functionality of, 9–11
interaction of namespaces with, 13
assembly attributes, customizing, 11
Assert Debug class method, 363
assertions
definition of, 360
using with **Debug** class, 362–364
asterisks (*), advisory about, 434
AttachDBFilename **ConnectionString** property value, details of, 66–67
AttributeCount XmlReader class property, 162
auditing, securing message queuing with, 502
authenticated messages, sending, 494–495
authentication, securing message queuing with, 490–495

AuthenticationType property of **DirectoryEntry** class, description of, 417
AUTO SELECT . . . FOR XML statement keyword, description of, 588
AutoIncrement DataColumn class property, description of, 250
AutoIncrementSeed DataColumn class property, description of, 250
AutoIncrementStep DataColumn class property, description of, 250
automatic garbage collection, role in .NET Framework, 8
automatic transactions
  explanation of, 110
  use of, 586

B

backslash (\) in Active Directory hierarchies, 420
BaseURI XmlReader class property, 162
BEFORE triggers, functionality of, 406
Begin method of Transaction class, description of, 109
BeginEdit DataRow class property, description of, 246
BeginLoadData DataTable class method, description of, 221
BeginTransaction method
  exception handling, 111–112
  of **OdbcConnection** class, 83, 101
  of **OleDbConnection** class, 81
  of **SqlConnection** class, 80, 100–101
BINARY BASE64 SELECT . . . FOR XML statement keyword, description of, 588
binding context, examining for Windows forms, 527–528
BindingContext object, role in Windows forms, 527–528, 538–539
Bindings property of **CurrencyManager** class, description of, 540
Body setting for message queue encryption, explanation of, 497–498
BOF (Beginning-Of-File), moving cursors to, 263–264
Broken member of **ConnectionState** Enum, description of, 89
broken ownership chain, explanation of, 376
browsers
  appearance after restarting virtual directory in SQLXML 2.0, 602
  troubleshooting in SQLXML 2.0, 609–610
browsing scope, definition of, 19
Bulk Load feature of SQLXML 2.0, resource for, 593

C

*.cmd extension, explanation of, 303
C (country) moniker, role in LDAP syntax, 421
C# keywords related to OOP, list of, 510–511
Cancel Command class method, description of, 138
CancelCurrentEdit CurrencyManager class method, description of, 540
CancelEdit DataRow class property, description of, 246
CanResolveEntity XmlReader class property, 162
Caption DataColumn class property, description of, 250
cascades, role in referential integrity, 44
CaseSensitive property
  of **DataSet** class, 197
  of **DataTable** class, 219
catch blocks
  role in exception handling, 344, 348–349
  role in filtering exceptions, 355
ChangeDatabase method
  of **OleDbConnection** class, 81, 83
  of **SqlConnection** class, 80
  throws exception, 114
Chaos member of **IsolationLevel** Enum, description of, 102
child tables
  explanation of, 39
  and referential integrity, 44–45
ChildColumns DataRelation class property, description of, 254
ChildKeyConstraint DataRelation class property, description of, 254
ChildRelations DataTable class property, description of, 219
Children property of **DirectoryEntry** class, description of, 417
ChildTable DataRelation class property, description of, 254
Class View in Solution Explorer, switching to, 507
classes, wrapping as components, 582
clear **DataSet**, explanation of, 209
clear **DataTable**, explanation of, 227
Clear method
  of **DataSet** class, 199
  of **DataTable** class, 221

ClearErrors DataRow class property, description of, 246
client-side cursors, explanation of, 262
clients, creating for UserMan example application, 582–583
close connection, explanation of, 86
Clone method
  of **DataSet** class, 199
  of **DataTable** class, 221
  using with **DataSet** objects, 210
  using with **DataTable** objects, 228
Close method
  of DataReader class, 152
  of **DirectoryEntry** class, 419
  of **OdbcConnection** class, 83
  of **OleDbConnection** class, 81
  of **SqlConnection** class, 80
  of **XmlReader** class, 164
Closed member of **ConnectionState** Enum, description of, 89
CLR (common language runtime)
  handling exceptions with, 359
  versus JVM, 9
  .NET Framework adherence to, 6–8
CLR tasks, built-in type of, 7
CLS (Common Language Specification), .NET Framework adherence to, 4, 6
CN (common name) moniker, role in LDAP syntax, 420–421
code, running simple stored procedures from, 380–382, 401–405
column and row change events, examining order of, 232–233
column attributes for tables, displaying with Database Designer, 310
column changes, handling in **DataTable** objects, 233–234
Column grid pane item in Query Designer, 326
ColumnChanged DataTable class event, description of, 224
ColumnChanging DataTable class event, description of, 224
ColumnMapping DataColumn class property, description of, 250
ColumnName DataColumn class property, description of, 250
columns
  adding to tables with Table Designer, 315–316
  checking for null values in, 149–150
  choosing in Data Form Wizard, 532
  in database table rows, definition of, 36
  role in databases, 36
  sorting with Query Designer, 324
Columns DataTable class property, description of, 219
COM components, using from within .NET Framework, 276–278
COM Interop
  resource for, 56
  using with ADO.NET disconnected layer, 276–278
COM types, generating metadata for, 277–278
Command class
  methods, 129–137
  properties, 138–139
command files, adding for scripts, 303–304
command mode, role in VS .NET IDE, 18–19
command object parameters, adding to UserMan example application, 575–576
Command objects
  instantiating for UserMan database, 573–574
  using in ADO.NET connected layer, 123
command properties, setting in ADO.NET connected layer, 178–184
Command Window in VS .NET IDE, modes for, 18–19
CommandBuilder class, using in ADO.NET connection layer, 188–191
commands, executing in ADO.NET connected layer, 139–142
CommandText property
  of **OdbcCommand** class, 135
  of **OleDbCommand** class, 132
  of **SqlCommand** class, 130
CommandTimeout property
  of **OdbcCommand** class, 135of **OleDbCommand** class, 132
  of **SqlCommand** class, 130
  throws **ArgumentException** exception, 143
CommandType property
  of **OdbcCommand** class, 135–136
  of **OleDbCommand** class, 132–133
  of **SqlCommand** class, 130
  throws **ArgumentException** exception, 144
Commit method of Transaction class, description of, 109
CommitChanges DirectoryEntry class method, description of, 419

Committed member of **MessageQueueTransactionStatus** Enum, description of, 483
common language runtime. *See* CLR
Common Language Specification. *See* CLS
common type system. *See* CTS
compare connections, explanation of, 87–88
comparisons, performing in ADO.NET connected layer, 87
components, wrapping classes as, 582
composite keys
  in databases, definition of, 43
  role in databases, 43
Computer Management MMC versus Server Explorer, advisory when creating public message queues, 457
Compute DataTable class method, description of, 221
concurrency violations
  handling, 273–275
  ignoring, 271–273
  occurrence of, 269–270
Connect As dialog box, displaying, 287
Connect Timeout or Connection Timeout **ConnectionString** property value, details of, 68–69
Connecting member of **ConnectionState** Enum, description of, 89
Connection class
  events, 84–85
  exceptions handling, 110–123
  explanation of, 63
  functionality of, 63–64
  methods of, 80–84
  properties of, 76–79
Connection Lifetime **ConnectionString** property value, details of, 68–69
Connection objects, comparing in ADO.NET connected layer, 87–88
Connection OdbcCommand class property, details of, 136
Connection OleDbCommand class property, details of, 133
connection oriented programming, explanation of, 450
connection pools
  explanation of, 90–96
  resetting connections in, 93
Connection property of Transaction class, description of, 108
Connection Reset **ConnectionString** property value, details of, 70–71
Connection Reset value, resetting connections in connection pools with, 93
Connection SqlCommand class property, details of, 130
connection state, manipulating in ADO.NET connected layer, 88–96
connection state messages, handling in ADO.NET connected layer, 97–98
connectionless programming, explanation of, 450
connections
  choosing in Data Form Wizard, 530
  closing in ADO.NET connection layer, 86
  deleting with Server Explorer, 285
  disposing in ADO.NET connected layer, 86–87
  handling with Server Explorer, 282–285
  opening in ADO.NET connected layer, 85–86
  removing from pools, 92
  setting with VS .NET IDE, 26
ConnectionState, comparing **State** to, 89
ConnectionString property
  examining after setting, 76
  example of white space in, 91
  exception handling, 112–113
  functionality of, 65
  values for, 66–75
ConnectionTimeout property
  exception handling, 113
  of **OdbcConnection** class, 79
  of **OleDbConnection** class, 78
  of **SqlConnection** class, 77
constraints, adding with Table Designer, 319–320
Constraints DataTable class property, description of, 219
constructors, adding to CUser class for wrapped databases, 518–519
Contains public property and method of **BindingContext** class, description of, 538
ContinueUpdateOnError DataAdapter class property, description of, 172

copy **DataSet**, explanation of, 209
copy **DataSet** structure, explanation of, 210
copy **DataTable**, explanation of, 227
copy **DataTable** rows, explanation of, 230–232
copy **DataTable** structure, explanation of, 227
Copy method
  of **DataSet** class, 200
  of **DataTable** class, 222
CopyTo DataView class method, 240
CopyTo DirectoryEntry class method, description of, 419
Count DataView class property, description of, 239
Count property of **CurrencyManager** class, description of, 540
Create Command File dialog box, displaying, 303
Create Database dialog box, displaying, 292
Create Message Queue dialog box, displaying, 290
CreateCommand method
  of **OdbcConnection** class, 83
  of **OleDbConnection** class, 81
  of **SqlConnection** class, 80
CreateParameter Command class method, description of, 138
Criteria grid pane column in Query Designer, 326
cross-language inheritance, role in .NET Framework, 8
Ctrl+Alt+J keyboard shortcut, opening Object Browser with, 19
Ctrl+Alt+K keyboard shortcut, accessing Task List window with, 27
Ctrl+Alt+S keyboard shortcut, accessing Server Explorer with, 281
Ctrl+S keyboard shortcut
  saving changes to stored procedures with, 381, 383
  saving database diagrams with, 310
  saving SQL Editor scripts with, 336
  saving triggers with, 408
  saving views with, 399
Ctrl+Shift+C keyboard shortcut, switching to Class View in Solution Explorer with, 507
Ctrl+Shift+N keyboard shortcut, displaying New Project dialog box with, 296
CTS (common type system), .NET Framework adherence to, 4, 6
Culture, Processor, OS, manifest components of assemblies, description of, 10
CurrencyManager class methods and properties, 540–541
CurrencyManager object, role in Windows forms, 527, 538–544
Current Language **ConnectionString** property value, details of, 70–71
Current property of **CurrencyManager** class, description of, 540
cursor locations, options for, 262–263
cursor types, 263–265
cursors, using in ADO.NET disconnected layer, 261–265
CUser class
  adding command object parameters to, 575–576
  adding constructors and destructors to, 518–519
  adding error event class to, 520–521
  adding events to, 519–524
  creating, 513–524
  declaring event and method raising event for, 522
  declaring event delegate for, 522
  displaying method receiving event in, 523
  displaying private database constants and variables for, 567–568
  filling data sets in, 576
  hooking event receiver with event in, 523
  hooking up public properties to data sets in, 576–578
  instantiating and initializing data adapters in, 574–575
  instantiating command objects in, 573–574
  instantiating **DataSet** objects in, 574
  locating code for, 579
  making connection object shared in, 570
  opening database connection for, 570–572
  setting maximum column length variables for, 569–570
  specifying parent classes in, 578
  triggering invalid LoginName error event in, 523

D

DAP (Directory Access Protocol), introduction to, 413

data
displaying in databases, 397
manipulating in views from code, 402–405
data adapters, instantiating and initializing in UserMan example application, 574–575
data binding, suspending and resuming, 543–544
data-bound controls
choosing right data storage for, 549–551
communication with data sources, 528
controlling validation of edit operations in, 543–544
versus manual data hooking, 525–526
retrieving current row of data sources for, 542
retrieving and setting current position in data sources for, 542–543
retrieving number of rows in data sources for, 542
using Data Form Wizard with, 528–536
using with Web forms, 545–551
using with Windows forms, 527
data-bound Web forms, creating and updating, 551–561
data-bound Windows form controls, creating, 544
data classes for .NET Data Providers, list of, 63
data connections
choosing in Data Form Wizard, 530
closing in ADO.NET connection layer, 86
deleting with Server Explorer, 285
disposing in ADO.NET connected layer, 86–87
handling with Server Explorer, 282–285
opening in ADO.NET connected layer, 85–86
removing from pools, 92
setting with VS .NET IDE, 26
data distribution, performance resources optimization of, 369
Data Form Wizard
choosing data connections in, 530
choosing data display in, 533
choosing data sets in, 529
choosing tables and columns in, 532
choosing tables and views in, 531
creating data-binding forms with, 551–557
creating forms for data-bound controls with, 528–536
examining code created by, 536–537
examining frmUser.cs file in, 537
examining objUser object declaration in, 536–537
opening frmUser form with, 536
role of **Dispose** methods in, 536
sample form created by, 534
data integrity
in databases, explanation of, 44
role in databases, 44
Data Link Properties dialog box, displaying, 283–284
data providers
in ADO.NET connected layer, 59–62
role in ADO and ADO.NET, 57
data-related exceptions, handling, 357–359
data-related namespaces, role in ADO.NET connected layer, 58–59. *See also* namespaces
data sets
choosing in Data Form Wizard, 529
filling in UserMan example application, 576
Data Source **ConnectionString** property value, details of, 66–67
data source locking, role in ADO.NET disconnected layer, 265–276
data source messages, handling in ADO.NET connection layer, 97–98
data sources
binding ASP.NET server controls to, 545–548
binding Windows form controls to, 537–538
locking data at, 266–268
retrieving and setting current position for data-bound controls, 542–543
retrieving number of rows for data-bound controls, 541–542
updating from **DataSet** objects, 618–619
updating with DataAdapter classes, 206–209
updating with **SqlXmlCommand** class, 631–635
data storage, choosing for data-bound controls, 549–551
data wrappers
definition of, 505

reasons for use of, 505
DataAdapter classes
explanation of, 170–171
events, 174–175
methods of, 173–174
populating **DataSet** objects with, 203–206
preparing for CommandBuilder class, 189–191
properties of, 172
role in ADO.NET, 59, 170–172
setting command properties, 178–184
updating data sources with, 206–209
DataAdapter object, instantiating, explanation of 176–178
Database **ConnectionString** property value, details of, 66–67
DATABASE **ConnectionString** property value, details of, 66–67
Database Designer
adding relationships with, 311
adding tables with, 309
creating tables with, 309–313
deleting and removing tables with, 309
deleting relationships with, 311
designing databases with, 306–313
displaying column attributes with, 310
editing database properties with, 311
database diagrams
changing table views for, 313
changing viewable area of, 312
creating with Database Designer, 306–313
moving view ports in, 312–313
overview of, 311–312
saving, 314
showing relationship names in, 312
sizing tables automatically in, 313
zooming in, 311–312
database objects
adding to database projects, 300–304
creating with Server Explorer, 285
identifying, 37
database projects
adding database objects to, 300–304
creating, 296–298
creating folders for, 298–299
definition of, 296
deleting folders from, 299
database properties, editing with Database Designer, 311
Database property
of **OdbcConnection** class, 79
of **OleDbConnection** class, 78
of **SqlConnection** class, 77
throws exceptions, 114
Database Query database object template, description of, 301
database tables versus message queues, 452
databases. *See also* SQL Server databases
benefits of, 32–33
columns in, 36
composite keys in, 43
data integrity in, 44
declarative referential integrity (DRI) in, 45
definition of, 31–32
designing with Database Designer, 306–313
designing with Visio for Enterprise Architect, 305
determining objects for, 37–38
displaying data in, 397
domain integrity in, 45
entity integrity in, 44
fields in, 36
foreign keys in, 42–43
hierarchical, description of, 33–34
indexes in, 43
keys in, 42
lookup keys in, 42
normalization in, 46
null values in, 36
primary keys in, 42–43
procedural referential integrity (PRI) in, 45
reasons for use of, 32–33
records in, 36
referential integrity in, 44–45
rows in, 36
tables in, 36
wrapping, 513–524
DataBind method, role in binding ASP.NET server controls to data sources, 545, 547–548
DataBindings dialog box, displaying, 560
DataColumn class
properties of, 250–252
using in ADO.NET disconnected layer, 249–252
DataColumn objects, declaring and instantiating, 252
DataGrid Properties dialog box, displaying, 553
DataGridCancelCommand event procedure, displaying, 554–555
DataGridEditCommand event procedure, displaying, 553

DataGridUpdateCommand event procedure, displaying, 555–556
DataReader classes
closing, 149
exception handling, 160
handling multiple results with, 150
instantiating, 176–178
methods of, 151–160
properties of, 151
reading rows in, 148
usage of, 160–161
use of server-side cursors by, 263
using with ADO.NET connected layer, 146
DataReader objects, declaring and instantiating, 147–148
DataRelation class
properties of, 254
using in ADO.NET disconnected layer, 252–261
DataRelation objects
declaring and instantiating, 254–256
exceptions, 253
DataRow and **DataColumn** classes, functionality of, 232–233
DataRow class, using in ADO.NET disconnected layer, 245–249
DataRow objects
building, 249
declaring and instantiating, 249
DataSet class
events of, 202–203
methods of, 199–201
properties of, 197–199
using with ADO.NET disconnected layer, 193–194
DataSet Designer, creating typed data sets with, 338–340
DataSet objects
accepting or rejecting changes to data in, 214–218
building UserMan database schema as, 256–261
clearing data from, 209
copying data and structures of, 209–210
detecting and handling changes to data in, 212–214
instantiating, 203
instantiating in CUser class, 574
merging data between, 210–212
populating with DataAdapter class, 203–206
populating with **SqlXmlAdapter** class, 614–616
updating data sources from, 618–619
using **AcceptChanges** method with, 215
using **RejectChanges** method with, 216–218
using with data-bound controls, 550
DataSet property
of **DataRelation** class, 254
of **DataTable** class, 219
DataSetName DataSet class property, description of, 197
DataSource property
of **OdbcConnection** class, 79
of **OleDbConnection** class, 78
of **SqlConnection** class, 77
DataTable class
events of, 223–224
methods of, 220–223
properties of, 218–220
using with **DataSet** class, 194, 218–223
DataTable objects
building, 225–227
clearing data from, 227
copying, 227–229
copying rows in, 230–232
declaring and instantiating, 224–225
examining order of column and row change events in, 232–233
handling column changes in, 233–234
handling row changes in, 235–236
handling row deletions in, 236–238
merging **DataSet** objects with, 212
populating, 227
searching and retrieving filtered data views from, 229–230
using with CommandBuilder class, 188–189
DataType DataColumn class property, description of, 250
DataView class
events of, 241
methods of, 240
properties of, 239
using in ADO.NET disconnected layer, 238–245
DataView objects
declaring and instantiating, 242
searching, 243
sorting, 244–245
DataViewManager DataView class property, description of, 239
DBCC (Database Consistency Checker), functionality of, 371–372
DBMS (database management system), definition of, 35

dbo (database owner), role in creating SQL Server stored procedures, 376
DBObject virtual name, querying objects with, 605–606
DBQ **ConnectionString** property value, details of, 66–67
DC (domain component) moniker, role in LDAP syntax, 421
dead-letter message queues
  explanation of, 485–486
  rejecting messages in, 492
debug assertions, using, 363–364
Debug class
  advisory about, 364
  methods and properties of, 366
  using, 359–366
  using error messages with, 364–366
debugger in VS .NET IDE, explanation of, 20–21
debugging, enabling and disabling, 360–362
DefaultValue DataColumn class property, description of, 250
DefaultView DataTable class property, description of, 219
DefaultViewManager DataSet class property, description of, 197
Delete DataRow class property, description of, 246
Delete DataView class method, 240
DELETE queries, using Query Designer for, 331–332
DeleteCommand DataAdapter class property, description of, 172
DeleteTree DirectoryEntry class method, description of, 419
denormalization, explanation of, 51
deployment tools, upgrades in VS .NET IDE, 23
Depth property of DataReader class, description of, 151
Depth XmlReader class property, 162
DesignTimeVisible property
  of **OdbcCommand** class, 136
  of **OleDbCommand** class, 133
  of **SqlCommand** class, 130
destructors, adding to CUser class for wrapped databases, 518–519
deterministic finalization versus automatic garbage collection, 8
diagram pane of Query Designer, functionality of, 322–325
DiffGrams
  advisory about using with SQLXML 2.0 Managed classes, 619
  generating and executing, 633–635
direct URL queries in SQLXML 2.0, advisory about, 606
DirectoryEntries class in **System.DirectoryServices** namespace, description of, 416
DirectoryEntry class
  description of, 416
  methods of, 419–420
  studying in Active Directory, 417–422
  in **System.DirectoryServices** namespace, 415
DirectorySearcher class
  description of, 416
  in **System.DirectoryServices** namespace, 415
  using with objects in Active Directory, 422–423
DisplayExpression DataTable class property, description of, 219
Dispose methods, role in Data Form Wizard code, 536
dispose of connection, explanation of, 86–87
distributed applications versus standalone applications, 368
DLL (dynamic link library) classes versus assemblies, 10
DN (distinguished name), role in Active Directory, 420
domain integrity in databases, explanation of, 45
DRI (declarative referential integrity) in databases, explanation of, 45
DRIVER **ConnectionString** property value, details of, 66–67
driver messages, handling in ADO.NET connection layer, 97–98
Driver OdbcConnection class property, description of, 79
drivers. *See* OLE DB drivers
DSN **ConnectionString** property value, details of, 68–69
dynamic cursors, explanation of, 265
Dynamic Help, role in VS .NET IDE, 21–22

E

edit mode, generating data-bound Web forms in, 555
edit operations, controlling validation for data-bound controls, 543–544
ELEMENTS SELECT . . . FOR XML statement keyword, description of, 588

encapsulation, role in OOP, 510
encrypted messages, sending and receiving, 498–499
encryption
  securing message queuing with, 495–499
  using views for, 398
EndCurrentEdit CurrencyManager class method, description of, 540
EndEdit DataRow class property, description of, 246
EndLoadData DataTable class method, description of, 222
EnforceConstraints DataSet class property, description of, 197
Enlist **ConnectionString** property value, details of, 70–71
entity integrity
  in databases, explanation of, 44
  role in databases, 44
enums, tip when passing arguments to methods and properties, 522
EOF XmlReader class property, 162
equal to (=) column comparison symbol, role in Query Designer join types, 323
Equals method, using in ADO.NET connected layer, 87–88
Equals method
  of **OdbcConnection** class, 83
  of **OleDbConnection** class, 82
  of **SqlConnection** class, 80
ER/Studio, Web site for, 37
ErrorCode property of **OleDbException** class, details of, 117
Errors property
  of **OdbcException** class, 121
  of **OleDbException** class, 117
ER*win*, Web site for, 37
events, adding to CUser class for wrapped databases, 519–524
Exception class
  examining, 346–348
  methods of, 348
exception handlers
  handling exceptions in, 348–354
  using two or more in single procedures, 345–346
exception handling
  enabling, 345
  functionality of, 343–344
  role of **catch** blocks in, 344
  role of **finally** blocks in, 344–345
  role of **try** statement in, 344
  types of handlers in, 345
Exception type, 350
exceptions
  CLR handling of, 359
  creating, 356
  filtering, 354–356
  handling data-related type of, 357–359
  throwing, 356–357
  types of, 350–351
  understanding generation of, 111
Exchange Server 2000
  accessing, 436–447
  accessing as linked server from SQL Server, 442–447
  retrieving contacts for UserMan user from, 438–439
  reviewing content of sample folder with, 440–441
ExecuteNonQuery method
  description of, 138
  usage of, 140–141, 631–632
ExecuteReader method
  description of, 138
  usage of, 140–141
ExecuteScalar method
  description of, 138
  usage of, 140, 142
ExecuteXmlReader method
  description of, 139
  usage of, 140, 142
Executing member of **ConnectionState** Enum, description of, 89
Exists DirectoryEntry class method, description of, 419
ExOLEDB (Microsoft OLE DB Exchange Server Provider), using, 438–439
EXPLICIT SELECT . . . FOR XML statement keyword, description of, 588
Expression DataColumn class property, description of, 251
ExtendedProperties property
  of **DataColumn** class, 251
  of **DataRelation** class, 254
  of **DataSet** class, 197
  of **DataTable** class, 219
external transactions, role in message queues, 480

F

Fail method, role in using error messages with Debug class, 365
FAQs (Frequently Asked Questions), about databases, list of, 35–51
fault handlers, role in exception handling, 345

Fetching member of **ConnectionState** Enum, description of, 89
FieldCount property of DataReader class, description of, 151
fields, role in databases, 36
FIFO (First In, First Out) principle versus message queues, 453
FIL **ConnectionString** property value, details of, 70–71
file-based XML templates, executing from code with SQLXML 2.0, 617
File Name **ConnectionString** property value, details of, 70–71
File Table manifest component of assemblies, description of, 10
Fill method
  of DataAdapter class, 173
  populating **DataSet** objects with, 203–206
fill operation errors, handling in ADO.NET connection layer, 184–185
FillError DataAdapter class event, description of, 175
FillSchema DataAdapter class method, description of, 173
filter handlers, role in exception handling, 345
finally blocks, role in exception handling, 344–345
finally handlers, role in exception handling, 345
Find DataView class method, 240
FindRows DataView class method, 240
FirstName column in tblUser column for CUser class, details of, 514
folders
  creating for database projects, 298–299
  deleting for database projects, 299
foreign keys
  in databases, definition of, 42
  role in databases, 42–43
ForeignKeyConstraint objects, advisory when using **DataSet** objects, 215
format names, binding to existing message queues with, 460–461
formatters, using with message queues, 464–467
forms, creating for data binding, 551–557
forms for data-bound controls, creating with Data Form Wizard, 528–536
forward-only cursors, explanation of, 263–264
friendly names, binding to existing message queues with, 459–460
frmUser form, opening with Data Form Wizard, 536
frmUser.cs file, examining in Data Form Wizard, 537

G

garbage collection, role in .NET Framework, 7–8
Generate Dataset dialog box, displaying, 559
Get* methods of DataReader class, 152–159
GetAttribute XmlReader class method, 164
GetBaseException method
  of **OdbcException** class, 123
  of **OleDbException** class, 119
GetBoolean method of DataReader class, 152
GetByte method of DataReader class, 152
GetBytes method of DataReader class, 152
GetChanges method
  of **DataSet** class, 200
  of **DataTable** class, 222
GetChar method of DataReader class, 153
GetChars method of DataReader class, 153
GetChildRows DataRow class property, description of, 247
GetColumnError DataRow class property, description of, 247
GetColumnsInError DataRow class property, description of, 247
GetDataTypeName method of DataReader class, 153
GetDateTime method of DataReader class, 153
GetDecimal method of DataReader class, 153
GetDouble method of DataReader class, 154
GetEnumerator DataView class method, 240
GetErrors DataTable class method, description of, 222
GetFieldType method of DataReader class, 154
GetFillParameters DataAdapter class method, description of, 174

GetFloat method of DataReader class, 154
GetGuid method of DataReader class, 154
GetHashCode method
of **OdbcConnection** class, 83
of **SqlConnection** class, 80
GetInt16 method of DataReader class, 154
GetInt32 method of DataReader class, 154
GetInt64 method of DataReader class, 155
GetName method of DataReader class, 155
GetOleDbSchemaTable
OleDbConnection class method, details of, 82
GetOrdinal method of DataReader class, 155
GetParentRow DataRow class property, description of, 247
GetParentRows DataRow class property, description of, 248
GetSchemaTable method of DataReader class, 155
GetSqlBinary method of DataReader class, 155
GetSqlBoolean method of DataReader class, 156
GetSqlByte method of DataReader class, 156
GetSqlDateTime method of DataReader class, 156
GetSqlDecimal method of DataReader class, 156
GetSqlDouble method of DataReader class, 156
GetSqlGuid method of DataReader class, 156
GetSqlInt16 method of DataReader class, 157
GetSqlInt32 method of DataReader class, 157
GetSqlInt64 method of DataReader class, 157
GetSqlMoney method of DataReader class, 157
GetSqlSingle method of DataReader class, 157
GetSqlString method of DataReader class, 157
GetSqlValue method of DataReader class, 158
GetSqlValue method of DataReader class, 158
GetSqlValues method of DataReader class, 158
GetString method of DataReader class, 158
GetTimeSpan method of DataReader class, 158
GetType method
of **OdbcConnection** class, 83
of **OleDbConnection** class, 82
of **SqlConnection** class, 80
GetValue method of DataReader class, 158
GetValues method of DataReader class, 159
GetXml DataSet class method, description of, 200
GetXmlSchema DataSet class method, description of, 200
greater than (>) column comparison symbol, role in Query Designer join types, 323
greater than or equal to (>=) column comparison symbol, role in Query Designer join types, 323
grid pane of Query Designer, functionality of, 325–327
Group By grid pane column in Query Designer, 326
guarded blocks of code, using exception handlers with, 345
Guid property of **DirectoryEntry** class, description of, 417

H

Has . . . a inheritance, explanation of, 507
HasAttributes XmlReader class property, 162
HasChanges DataSet class method, description of, 200
HasErrors property
of **DataRow** class, 245
of **DataSet** class, 198
of **DataTable** class, 219
HasValue XmlReader class property, 162
HasVersion DataRow class property, description of, 248
Help system, integration in VS .NET IDE, 21–22
HelpLink property
of **Exception** class, 347
of **OdbcException** class, 121
of **OleDbException** class, 117
hierarchical databases, description of, 33–34

hierarchical versus relational databases, 33–35
HResult property
of **OdbcException** class, 121
of **OleDbException** class, 118

I
ICT Database Designer Tool, Web site for, 37
Id column in tblUser column for CUser class, details of, 514–515
Identity manifest component of assemblies, description of, 10
IDEs (Integrated Development Environments). *See also* VS .NET IDE
running scripts in, 302
running simple stored procedures from, 379–380
running stored procedures with arguments from, 384
running views from, 400–401
IDL (Interface Definition Language) files, role in language compilation, 8
IDs, retrieving for message queues, 458–459, 468–469
immediate mode, role in VS .NET IDE, 18–19
implementation inheritance, role in OOP, 509–510
ImportRow method, using with **DataTable** objects, 232
ImportRow DataTable class method, description of, 222
IMS (Information Management System), Web site for, 34
indexes
adding with Table Designer, 317–319
in databases, definition of, 43
role in databases, 43
IndexOutOfRangeException exception, 350, 352, 355
InferXmlSchema DataSet class method, description of, 200
InfoMessage Connection class event, details of, 85
information hiding, role in OOP, 510
inheritance, role in OOP, 507–510
Initial Catalog **ConnectionString** property value, details of, 70–71
Initial File Name **ConnectionString** property value, details of, 70–71
Initialized member of **MessageQueueTransactionStatus** Enum, description of, 483
InnerException property
of **Exception** class, 347
of **OdbcException** class, 122
of **OleDbException** class, 118
INSERT queries, using Query Designer for, 333
INSERT triggers, functionality of, 406
InsertCommand DataAdapter class property, description of, 172
integrated debugger in VS .NET IDE, explanation of, 20–21
Integrated Security **ConnectionString** property value, details of, 70–71
interface inheritance, role in OOP, 507–509
interface keyword in C#, relationship to OOP, 511
interface modes for VS .NET IDE, explanation of, 16
interfaces, creating and implementing with OOP, 508–509
internal transactions, role in message queues, 480–481
Interop Assemblies, location of, 278
InvalidCastException exception
handling for DataReader class, 160
thrown by **XmlReader** methods, 164
InvalidConstraintException exception thrown by **DataRelation** class, 253
InvalidOperationException exception, 351–352
handling for DataReader class, 160
handling for **XmlReader** class, 170
throwing, 111–112, 115, 144–145
Invoke DirectoryEntry class method, description of, 419
Is a . . . inheritance, explanation of, 507, 509
ISAPI extension
configuring for SQLXML 2.0, 593–598
testing for SQLXML 2.0, 599
IsClosed property of DataReader class, description of, 151
IsDBNull method of DataReader class, 159
IsDefault XmlReader class property, 162
IsEmptyElement XmlReader class property, 162
IsName XmlReader class method, 164
IsNameToken XmlReader class method, 164
IsNull DataRow class property, description of, 248

isolation levels, determining for running transactions, 107–108
IsolationLevel Enum, members of, 102
IsolationLevel property of Transaction class, description of, 108
IsReadOnly public property and method of **BindingContext** class, description of, 538
IsStartElement XmlReader class method, 164
ItemArray DataRow class property, description of, 245

J
JIT (Just in Time) compiler, functionality of, 9
join dependence, role in 5NF, 51
join types, changing with diagram pane of Query Designer, 323–324
journal storage
  controlling size of, 489–490
  retrieving messages from, 489
  using with message queues, 486–490
JScript, implications of CLR for, 6–7
JVM (Java Virtual Machine)
  versus CLR, 9

K
keys
  adding with Table Designer, 317–319
  in databases, definition of, 42
  role in databases, 42
keyset cursors, explanation of, 265

L
labels, binding to existing message queues with, 461
LastName column in tblUser column for CUser class, details of, 514
LDAP display names for users, Web site for, 425
LDAP (Lightweight Directory Access Protocol), introduction to, 413–414
LDAP query filters, Web sites for, 425
LDAP syntax, examining, 420–421
less than (<) column comparison symbol, role in Query Designer join types, 323
less than or equal to (<=) column comparison symbol, role in Query Designer join types, 323
line numbers, using in VS .NET IDE text editors, 24
link tables, using with many-to-many relationships, 41
linked servers
  accessing Microsoft Exchange Server as, 442–447
  creating, 443–444
  creating views on, 447
  dropping, 446
  setting up, 442
List property of **CurrencyManager** class, description of, 540
LoadDataRow DataTable class method, description of, 222
local processing resources, performance resources optimization of, 369
Locale property
  of **DataSet** class, 198
  of **DataTable** class, 219
LocalName XmlReader class property, 162
locking types, explanations of, 265–276
LoginName column in tblUser column for CUser class, details of, 514, 518, 523
lookup keys, role in databases, 42
LookupNamespace XmlReader class method, 164

M
machine names, creating message queues with, 454–455
Macros IDE, accessing, 22–23
mail messages, sending with SMTP, 437
mail, retrieving from Active Directory with OLE DB .NET data providers, 432
Make Table queries, using Query Designer for, 332
Managed Classes in SQLXML 2.0, introduction to, 611
managed code
  definition of, 6
  role in .NET Framework, 7
manifest components of assemblies, list of, 10
manipulation of databases, understanding, 33
manual data hooking versus data-bound controls, 525–526
manual transactions
  aborting, 106
  ADO.NET connection layer support for, 99
  committing, 107
  summary of, 109–110
many forms, 506–507

many-to-many relationships, explanation of, 41
mapping schema for XPath queries, displaying, 629–630
Max Pool Size **ConnectionString** property value, details of, 70–71
Max Pool Size value names, role in pooling connections for **SqlConnection**, 92
MaxLength DataColumn class property, description of, 251
MDAC (Microsoft Data Access Components)
  2.6, required on your system, 61
  2.7, distributed with VS .NET, 61
MDI (multiple document interface) mode, explanation of, 16
merge **DataSet**, explanation of, 210
Merge DataSet class method, description of, 200
merge failures, handling with **DataSet** class, 202–203
MergeFailed DataSet class event, 200
Message Exception class property, description of, 347
message formatters, setting up for message queues, 464–467
message journaling, explanation of, 488–489
Message property
  of **OdbcError** class, 121
  of **OdbcException** class, 122
  of **OleDbError** class, 117
  of **OleDbException** class, 118
message queues
  aborting transactions for, 483
  assigning labels to, 458
  binding to existing types of, 459–461
  binding with labels, 461
  clearing messages from, 472–473
  committing transactions for, 482–483
  controlling storage size of, 479–480
  creating programmatically, 454–456
  creating with Server Explorer, 289–290
  deleting with Server Explorer, 290
  displaying or changing properties in, 456–459
  enabling authentication on, 491
  enabling journaling on, 487
  ending transactions for, 482–483
  locating, 475–477
  making transactional, 480–485
  peeking at messages in, 467
  picking specific messages from, 468–469
  prioritizing messages in, 473–475
  rejecting nonencrypted messages in, 496–497
  removing, 477–479
  removing messages after specified time elapse, 493
  retrieving all messages from, 470
  retrieving IDs of, 458–459
  retrieving messages from, 463–467
  sending and retrieving messages from, 466–467
  sending messages to, 461–463
  setting up message formatters for, 464–467
  starting transactions for, 481
  usage of, 449, 451–453
  using journal storage with, 486–490
  using system-generated type of, 485–490
  using system journals with, 486–487
  using transactions with, 484–485
message queuing
  using access control with, 499–502
  using auditing with, 502
  using authentication with, 490–495
  using encryption with, 495–499
MessageQueue class, examining, 450
MessageQueueTransaction class, using, 483–485
messages
  deleting messages from, 290
  enabling journaling on, 487–488
  rejecting nonauthenticated type of, 491
  removing from message queues, 472–473
  removing from message queues after specified time elapse, 493
  retrieving from journal storage, 489
  sending and retrieving asynchronously, 470–472
  sending and retrieving from message queues, 466–467
metadata, generating for COM types, 277–278
Microsoft Access, advisory about, 367–368
Microsoft Data Access Components. *See* MDAC.
Microsoft Exchange Server
  accessing, 436–447
  accessing as linked server from SQL Server, 442–447
  retrieving contacts for UserMan user from, 438–439
  reviewing content of sample folder with, 440–441

Microsoft OLE DB Exchange Server Provider (ExOLEDB), using, 438–439
Microsoft OLE DB Provider for Internet Publishing (MSDAIPP), using, 439–441
Microsoft Visual Studio .NET documentation, role in VS .NET IDE Help system, 22
Microsoft.Data.Odbc namespace, description of, 58
Microsoft.Jet.OLEDB.4.0 provider, description of, 61
Min Pool Size **ConnectionString** property value, details of, 72–73
Min Pool Size value names, role in pooling connections for **SqlConnection**, 92
MinimumCapacity DataTable class property, description of, 220
MissingMappingAction DataAdapter class property, description of, 172
MissingSchemaAction DataAdapter class property, description of, 172
monikers, role in LDAP syntax, 420–421
Move* **XmlReader** class methods, 165–166
MoveTo DirectoryEntry class method, description of, 419
MoveToAttribute XmlReader class method, 165
MoveToContent XmlReader class method, 165
MoveToElement XmlReader class method, 165
MoveToFirstAttribute XmlReader class method, 165
MoveToNextAttribute XmlReader class method, 166
MSDAIPP (Microsoft OLE DB Provider for Internet Publishing), using, 439–441
MSDAORA provider, description of, 61
MSIL (Microsoft intermediate language), compilation of code to, 8
MSMQ (Microsoft Message Queue Server), explanation of, 449–450
multivalued dependencies, role in 4NF, 49
MySQL, advisory about, 368

N

n-tier applications, relationship to ADO.NET, 56
Name property
  of **DirectoryEntry** class, 418
  of **SqlParameter** class, 612
  of **XmlReader** class, 162
named parameters, executing SQL queries with, 623–625
named SQL transactions, beginning, 103
named transactions, tip about, 107
namespace pollution, definition of, 12
Namespace property
  of **DataColumn** class, 251
  of **DataSet** class, 198
  of **DataTable** class, 220
namespaces. *See also* data-related namespaces
  determining for classes, 123
  functionality of, 11–13
  public status of, 13
NameSpaceURI XmlReader class property, 163
NameTable XmlReader class property, 163
NativeError property
  of **OdbcError** class, 121
  of **OleDbError** class, 117
NativeGuid property of **DirectoryEntry** class, description of, 418
NativeObject property of **DirectoryEntry** class, description of, 418
Nested DataRelation class property, description of, 254
nested OleDb transactions, beginning, 105–106
nesting, definition of, 397
.NET Data Providers
  Connection class in, 63–64
  explanation of, 62–63
.NET-enabled applications, connecting to, 64
.NET Framework
  components of, 4–9
  cross-language inheritance in, 8–9
  data-related namespaces in, 58
  description of, 3–4
  garbage collection in, 7–8
  managed code in, 7
  using COM components from within, 276–278

.NET Framework class library, contents of, 13–14
Net or Network Library **ConnectionString** property value, details of, 72–73
Network Address **ConnectionString** property value, details of, 72–73
network directory services, explanation of, 413
network resources, performance resources optimization of, 369
New Project dialog box, displaying, 296
New Value grid pane column in Query Designer, 326
NewRow DataTable class method, description of, 222
NextResult method of DataReader class, 159
nodes, separating in Active Directory hierarchy, 420
NodeType XmlReader class property, 163
non-row returning SQL statements, executing, 632
non-row-returning string templates, executing, 632–633
non-row-returning templates, executing, 633
normal forms, explanation of, 46–51
normalization in databases, explanation of, 46
not equal to (<>) column comparison symbol, role in Query Designer join types, 323
NotSupportedException exception, handling for DataReader, 160
null values
  advisory when using CommandBuilder class, 190
  checking in columns, 149–150
  in databases, definition of, 36
  role in databases, 36
NullReferenceException exception, explanation of, 350, 352

O

O (organization) moniker, role in LDAP syntax, 421
Object Browser, role in VS .NET IDE, 19–20
object property values, manipulating with Active Directory, 426–429
objects
  binding to in Active Directory, 421–422
  checking structural integrity with DBCC, 371–372
  determining for databases, 37
  querying with DBObject virtual name, 605–606
  searching in Active Directory, 422–425
objectSid retrieving from Active Directory with OLE DB .NET data providers, 432
objUser object declaration, examining in Data Form Wizard, 536–537
ODBC connection pooling, disabling, 94–96
ODBC connection strings, setting up, 61–62
ODBC object pooling, clearing, 96
ODBC (Open Data Base Connectivity), Web site for, 96
ODBC (Open Data Base Connectivity) data sources, accessing in ADO, 59
ODBC standard escape sequences, Web sites for, 137
OdbcCommand class
  instantiating, 126–127
  properties of, 79, 135–137
  usage of, 124
OdbcConnection class, **BeginTransaction** method of, 101
OdbcConnection data type, pooling connections of, 94–96
OdbcConnection exceptions, handling, 119–123
OdbcConnection managed connection, explanation of, 63–64
OdbcDataAdapter class, setting command properties in, 182–184
OdbcDataReader class, use of, 147
OdbcError class, examining, 119–120
OdbcException class, examining, 121–123
ODBC .NET data provider, downloading, 58
OdbcTransaction classes, nesting transactions with, 106
OLE DB connection pooling, disabling, 93
OLE DB drivers
  in ADO.NET connected layer, 59
  specifying when connecting, 61–62
OLE DB .NET data provider, accessing Active Directory with, 429–436

OLE DB (Object Linking and Embedding Data Base) specification, explanation of, 59
OLE DB object pooling, clearing, 93–94
OLE DB providers
  compatibility with OLE DB.NET providers, 61
  selecting for data sources with Server Explorer, 283
  specifying for Active Directory connections, 430–431
  specifying when connecting, 61–62
OLE DB.NET providers compatible with OLE DB providers, list of, 61
OleDbCommand class
  properties of, 132–134
  usage of, 124
OleDbCommand object, instantiating, 124–126
OleDbConnection class
  methods of, 81–84
  properties of, 78
OleDbConnection data type, pooling connections of, 93–94
OleDbConnection exceptions, throwing, 115–119
OleDbConnection managed connection, explanation of, 63–64
OleDbDataAdapter class, setting command properties in, 181–182
OleDbDataReader class, use of, 147
OleDbError class, examining, 116–117
OleDbException class
  examining, 117–119
  methods of, 119
  traversing, 116
OleDbTransaction class, nesting transactions with, 105–106
one-to-many relationships, explanation of, 40
one-to-one relationships, explanation of, 39–40
OOP (Object Oriented Programming)
  C# keywords related to, 510–511
  overriding properties in derived classes with, 512–513
  overview of, 506–513
  resources for, 506
  role of encapsulation in, 510
  role of implementation inheritance in, 509–510
  role of inheritance in, 507–510
  role of polymorphism in, 506–507
open connection, explanation of, 85–86
Open member of **ConnectionState** Enum, description of, 89
Open method
  of **OdbcConnection** class, 83
  of **OleDbConnection** class, 82
  of **SqlConnection** class, 81
  throwing exceptions, 115
optimistic locking, explanation of, 265, 268–276
optimization issues, explanation of, 368–373
Option setting for message queue encryption, explanation of, 498
Options dialog box, accessing, 24–25
Options dialog box in SQL Editor, displaying, 334
Or . . . grid pane column in Query Designer, 326
Oracle, advisory about, 368
Oracle Database Project, downloading, 392
Oracle stored functions, running, 391–392
Oracle stored procedures
  retrieving result set from, 396
  running, 392–397
order systems, typical objects for, 38
Ordinal DataColumn class property, description of, 251
organization of databases, understanding, 32
orphaned child tables, preventing with referential integrity, 45
OU (organizational unit) moniker, role in LDAP syntax, 420–421
Output grid pane column in Query Designer, 326
override keyword in C#, relationship to OOP, 511
ownership chain, definition of, 391

P

Packet Size **ConnectionString** property value, details of, 72–73
PacketSize SqlConnection class property, details of, 77
Page_Load event procedure, displaying, 556–557
Parameters property
  of **OdbcCommand** class, 136
  of **OleDbCommand** class, 133
  of **SqlCommand** class, 130
parent classes, specifying in CUser class, 578
Parent property of **DirectoryEntry** class, description of, 418
parent tables, explanation of, 39

ParentColumns DataRelation class property, description of, 254
ParentKeyConstraint DataRelation class property, description of, 254
ParentRelations DataTable class property, description of, 220
ParentTable DataRelation class property, description of, 254
Password column in tblUser column for CUser class, details of, 514
Password or Pwd **ConnectionString** property value, details of, 72–73
Password property of **DirectoryEntry** class, description of, 418
Path property of **DirectoryEntry** class, description of, 418
PE (portable executable) files, storage of compiled code in, 8
Peek and **Receive** methods of **MessageQueue** class, explanation of, 464
Pending member of **MessageQueueTransactionStatus** Enum, description of, 483
perfmon (Performance Monitor), functionality of, 372
performance degradation, troubleshooting, 371–373
performance resources optimization table, 369
Permissions manifest component, description of, 10
Persist Security Info **ConnectionString** property value, details of, 72–73, 76
pessimistic locking, explanation of, 266–268
polymorphism, role in OOP, 506–507
pooling connections
  in ADO.NET connected layer, 90–96
  of data type **OdbcConnection**, 94–96
  of data type **OleDbConnection**, 93–94
  of data type **SqlConnection**, 92–93
Pooling **ConnectionString** property value, details of, 74–75
pools
  removing connections from, 92
  testing for, 92
populate **DataSet** with DataAdapter class, 203–206
populate **DataTable**, explanation of, 227
Position property of **CurrencyManager** class, description of, 540
positional parameters, executing SQL queries with, 625–627
Prefix property
  of **DataColumn** class, 251
  of **DataSet** class, 198
  of **DataTable** class, 220
  of **XmlReader** class, 163
Prepare Command class method, description of, 139
PRI (procedural referential integrity) in databases, explanation of, 45
primary keys
  in databases, definition of, 42
  role in databases, 42–43
  setting with Table Designer, 316–317
PrimaryKey DataTable class property, description of, 220
privacy for message queue encryption, explanation of, 497
private message queues
  creating on local machines, 454–456
  versus public message queues, 453–454
  retrieving, 475–476
  using with Server Explorer, 288
private variables, creating for CUser class, 515–516
procedures, using multiple exception handlers in, 345–346
processing resources, determining for optimization, 370
programming concepts, review of, 4–14
properties
  adding to Active Directory, 427–429
  checking existence with Active Directory, 426
  displaying and changing for message queues, 456–459
  editing with Active Directory, 427
  overriding in derived classes with OOP, 512–513
  using local cache for, 426
Properties property of **DirectoryEntry** class, description of, 418
Property Pages dialog box in Table Designer, displaying, 318–319
PropertyCollection class in System.DirectoryServices namespace, description of, 416
PropertyValueCollection class in System.DirectoryServices namespace, description of, 416
Provider **ConnectionString** property value, details of, 74–75

provider messages, handling in ADO.NET connection layer, 97–98
Provider OleDbConnection class property, details of, 78
providers. *See* OLE DB providers
public message queues
  creating, 455
  versus private message queues, 453–454
  retrieving, 476–477
  using with Server Explorer, 288
public properties, hooking up to data sets in UserMan example application, 576–578
public status of namespaces, advisory about, 13
PWD **ConnectionString** property value, details of, 72–73

Q

queries
  executing from browsers using file-based templates, 606–610
  executing with Query Designer, 330
Query Analyzer, functionality of, 372
Query Builder, accessing, 335
Query Designer
  changing join types with, 323–324
  creating select query with, 321
  creating SQL statements with, 325
  diagram pane in, 322–325
  executing queries with, 330
  grid pane in, 325–327
  grouping and sorting output in, 324–325
  hiding and showing panes in, 328–329
  removing tables with, 323
  Results pane in, 328
  sorting columns with, 324
  SQL pane in, 327–328
  using DELETE queries with, 331–332
  using INSERT queries with, 333
  using Make Table queries with, 332
  using SELECT queries with, 331
  using UPDATE queries with, 331
  using Verify SQL Syntax facility in, 329–330
Query Editor, producing SQL statements with, 335–336
query types, examining in Query Designer, 330–334
QuoteChar XmlReader class property, 163

R

RAW SELECT . . . FOR XML statement keyword, description of, 588
RCWs (Runtime Callable Wrappers), using with COM types, 277
RDBMSes (relational database management systems), planning move to, 375
RDN (relative distinguished name), role in Active Directory, 420
Read method of DataReader class, 159
Read method of **XmlReader** class, 166
read-only data, retrieving from views in code, 401–402
read-only mode, generating data-bound Web forms in, 554
Read* **XmlReader** class methods, 166–167
ReadAttributeValue method of **XmlReader** class, 166
ReadCommitted member of **IsolationLevel** Enum, description of, 102
ReadElementString method of **XmlReader** class, 166
ReadEndElement method of **XmlReader** class, 166
ReadInnerXml method of **XmlReader** class, 166
ReadOnly DataColumn class property, description of, 251
ReadOuterXml method of **XmlReader** class, 167
ReadStartElement method of **XmlReader** class, 167
ReadState XmlReader class property, 163
ReadString method of **XmlReader** class, 167
ReadUncommitted member of **IsolationLevel** Enum, description of, 102
ReadXml DataSet class method, description of, 200
ReadXmlSchema DataSet class method, description of, 201
Receive and **Peek** methods of **MessageQueue** class, explanation of, 464
records, role in databases, 36
RecordsAffected property of DataReader class, description of, 151
Recordset class versus **DataSet** class, 194–196
reference points, saving in transactions, 104–105
Referenced Assemblies manifest component, description of, 10
ReferenceEquals method, comparing with, 88

referential integrity in databases, explanation of, 44–45
Refresh CurrencyManager class method, description of, 541
RefreshCache DirectoryEntry class method, description of, 419
Register SQL Server Instance dialog box, displaying, 291
Registry, explaining absence of, 4
RejectChanges method
of **DataRow** class, 248
of **DataSet** class, 201
of **DataTable** class, 223
using with **DataSet** objects, 216–218
relational database design, explanation of, 37–38
Relational Database Model, explanation of, 36
relational databases, key aspects of, 35–51
relational versus hierarchical databases, 33–35
RelationName DataRelation class property, description of, 254
Relations DataSet class property, description of, 199
relationship names, showing in database diagrams, 312
relationships
adding with Database Designer, 311
creating with Table Designer, 320
deleting with Database Designer, 311
explanation of, 39
ReleaseObjectPool method
clearing ODBC object pooling with, 96
clearing OLE DB object pooling with, 93–94
of **OdbcConnection** class, 84
of **OleDbConnection** class, 82
RemoveAt CurrencyManager class method, description of, 541
Rename DirectoryEntry class method, description of, 420
RepeatableRead member of **IsolationLevel** Enum, description of, 102
Reset method
of **DataSet** class, 201
of **DataTable** class, 223
ResetCommandTimeout Command class method, description of, 139
ResolveEntity method of **XmlReader** class, 167
result sets
appending to streams, 621–622
representation in ADO.NET, 261
retrieving from Oracle stored procedures, 396
retrieving in **XmlReader** class, 620
saving to streams, 620–621
ResultPropertyCollection class in System.DirectoryServices namespace, description of, 416
ResultPropertyValueCollection class in System.DirectoryServices namespace, description of, 416
Results pane of Query Designer, functionality of, 328
ResumeBinding CurrencyManager class method, description of, 541
return values and arguments, running stored procedures with, 385–387
RETURN_VALUE, retrieving from stored procedures, 390–391
Rollback method
role in aborting manual transactions, 106
of Transaction class, 109
row and column change events, examining order of, 232–233
row changes, handling in **DataTable** objects, 235–236
row deletions, handling in **DataTable** objects, 236–238
row updates, handling in ADO.NET connection layer, 185–188
RowChanged DataTable class event, description of, 224
RowChanging DataTable class event, description of, 224
RowDeleted DataTable class event, description of, 224
RowDeleting DataTable class event, description of, 224
RowError DataRow class property, description of, 245
RowFilter DataView class property, description of, 239
rows
appending to tables with INSERT Results queries, 333
copying in **DataTable** objects, 230–232
reading in **XmlReader** class, 168–169
retrieving from stored procedures, 382
retrieving from stored procedures with input arguments, 384–385

rows *(continued)*
retrieving in views, 402
role in databases, 36
in database tables, definition of, 36
Rows DataTable class property, description of, 220
rows of data sources
navigating in bound controls, 543
retrieving for data-bound controls, 542
RowState DataRow class property, description of, 245
RowStateFilter DataView class property, description of, 239
RowUpdated DataAdapter class event, description of, 175
RowUpdating DataAdapter class event, description of, 175
Run On dialog box, displaying, 302
running transactions, determining isolation levels of, 107–108

S

sa (system administrator), role in SQL Servers, 293
samAccountName retrieving from Active Directory with OLE DB .NET data providers, 432
Save method of Transaction class, description of, 109
save points, using with **SqlTransaction** class, 104–105
scalar values, returning with stored procedures, 389–391
SchemaClassName property of **DirectoryEntry** class, description of, 418
SchemaEntry property of **DirectoryEntry** class, description of, 418
SchemaNameCollection class in System.DirectoryServices namespace, description of, 416
SchemaPath property, role in executing XPath queries, 630
script editing with SQL Editor, performing, 334–337
script templates, editing and using with SQL Editor, 336
scripts
placing in command files, 303–304
running in IDEs, 302
saving with SQL Editor, 336
sealed keyword in C#, relationship to OOP, 511
search Active Directory, explanation of, 422–424
search **DataTable**, explanation of, 229–230
search **DataView**, explanation of, 243
SearchResult class in **System.DirectoryServices** namespace, description of, 416
SearchResultCollection class in **System.DirectoryServices** namespace, description of, 416
security, using views for, 398
Select DataTable class method, description of, 223
SELECT . . . FOR XML AUTO, ELEMENTS output, 589–590
Select method, finding rows in **DataTable** objects with, 229–230
SELECT queries, using Query Designer for, 331
SelectCommand DataAdapter class property, description of, 172
SelectCommand property, setting, 178, 189
SELECT . . . FOR XML AUTO, XMLDATA output, 590–591
SELECT . . . FOR XML RAW output, 590
SELECT . . . FOR XML statement keywords, list of, 588–589
Serializable member of **IsolationLevel** Enum, description of, 102
Server **ConnectionString** property value, details of, 74–75
SERVER **ConnectionString** property value, details of, 74–75
Server Explorer
adding servers with, 286–287
Connect As dialog box options in, 287
connecting servers as different users with, 286–287
creating database objects with, 285
creating SQL Server databases with, 292–295
creating triggers with, 407–412
deleting and dropping SQL Server databases with, 295
deleting data connections with, 285
deleting servers with, 295–296
explanation of, 25–26
handling data connections with, 282–285
introduction to, 281–282

registering SQL Server instances with, 291
selecting message queues in, 478
unregistering SQL Server instances with, 292
using message queues with, 288–290
using server resources with, 288–295
using with SQL Server databases, 290–295
server processing resources, performance resources optimization of, 369
server resources, using with Server Explorer, 288–295
server-side cursors, explanation of, 262–263
server-side processing, definition of, 367
servers
adding with Server Explorer, 286–287
connecting as different users with Server Explorer, 286–287
deleting with Server Explorer, 295–296
ServerVersion property
of **OdbcConnection** class, 79
of **OleDbConnection** class, 78
of **SqlConnection** class, 77
SET ANSI_DEFAULTS T-SQL statement, role in RDBMS migration from SQL Server, 375
set, property arguments, checking length of, in CUser class, 517
SetColumnError DataRow class property, description of, 248
SetParentRow DataRow class property, description of, 248
SetPermissions method, using with access control for message queues, 501–502
SimpleStoredProcedure, displaying output from, 380
Skip XmlReader class method, 167
SMTP (Simple Mail Transfer Protocol), sending mail messages with, 437
SOAP (Simple Object Access Protocol), based on XML, 5
uses HTTP protocol, 5
XML Web services based on, 5
Solution Explorer
finding database projects in, 298–299
locating Show All Files button in, 535
running SQL scripts with, 336–337
switching to Class View in, 507
sort **DataView**, explanation of, 244–245
Sort DataView class property, description of, 239
Sort Order grid pane column in Query Designer, 326
Sort Type grid pane column in Query Designer, 326
SortOption class in **System.DirectoryServices** namespace, description of, 416
Source property
of **Exception** class, 347
of **OdbcError** class, 121
of **OdbcException** class, 122
of **OleDbError** class, 117
of **OleDbException** class, 118
sp_addlinkedserver system stored procedure, resource for, 442–443
sp_dropserver system stored procedures, resource for, 446
SQL Dialect, Web site for, 434
SQL Editor
editing and using script templates with, 336
running SQL scripts with, 336–337
saving scripts with, 336
using, 334–337
SQL pane of Query Designer, functionality of, 327–328
SQL queries
executing with named parameters, 623–625
executing with positional parameters, 625–627
executing with **SqlXmlCommand** class, 620–624
SQL Script database object template, description of, 301
SQL scripts, running with SQL Editor, 336–337
SQL Server 7.0, advisory about, 367
SQL Server databases. *See also* databases
accessing Microsoft Server as linked server from, 442–447
creating nondefault values and properties for, 294
creating simple stored procedures with, 376
creating with Server Explorer, 292–295
deleting and dropping with Server Explorer, 295
functionality of stored procedures in, 392
resources for, 371

SQL Server databases *(continued)*
retrieving from, with **SqlXmlAdapter** class, 614–619
updating with **SqlXmlAdapter** class, 618–619
using Server Explorer with, 290–295, 290–295
using stored procedures with, 375–379
SQL Server Enterprise Manager, viewing stored procedure dependencies with, 391
SQL Server instances
registering with Server Explorer, 291
unregistering with Server Explorer, 292
SQL Server .NET Data Provider versus SQLXML 2.0 Managed classes, 611
SQL standards, Web site for, 327
SQL statements
creating in SQL pane of Query Designer, 325
executing non-row-returning type of, 632
producing with Query Editor, 335–336
SQL syntax, verifying with Query Designer, 329–330
SqlCommand class
properties of, 130–131
usage of, 124
SqlCommand object, instantiating, 128–129
SqlCommandBuilder class, using, 190
SqlConnection class, **BeginTransaction** method for, 100–101
SqlConnection class exceptions
catching, 357
handling, 110–115
methods of, 80–81
properties of, 77
SqlConnection data type, pooling connections for, 92–93
SqlConnection managed connection, explanation of, 63–64
SqlDataAdapter class
instantiating, 176–178
setting command properties of, 179–180
SqlDataReader class and object, instantiating, 147–148
SqlInfoMessageEventArgs class
argument, 97–98
Message property, 98
Source property, 98
SQLOLEDB provider, description of, 61
SQLState property
of **OdbcError** class, 121
of **OleDbError** class, 117
SqlTransaction class
nesting transactions with, 104–105
using transaction save points with, 104–105
SQLXML 2.0
advisory about direct URL queries in, 606
advisory about using template string method with, 617
configuring ISAPI extension for, 593–598
executing directory queries from browser in, 601–606
executing file-based XML templates from code in, 617
executing queries from browsers with file-based templates, 606–610
installing, 592–593
introduction to, 591
introduction to Managed Classes in, 611
querying tables in, 601–603
sample connection string in, 614
sample template query with parameters in, 608–609
SQLXML 2.0 Managed Classes, using DiffGrams and UpdateGrams with, 619
SqlXmlAdapter class
examining, 612–619
retrieving SQL Server data with, 614–619
updating SQL Server data with, 618–619
SqlXmlAdapter objects, instantiating, 613–614
SqlXmlCommand class
executing SQL queries with, 620–623
introduction to, 619
populating **DataSet**s with, 615–616
updating data sources with, 631–635
SqlXmlCommand objects, instantiating, 619
SqlXmlParameter class, examining, 612
StackTrace property
of **Exception** class, 347
of **OdbcException** class, 122
of **OleDbException** class, 118
stand-alone applications versus distributed applications, 368
standard HTML programming model for ASP.NET, explanation of, 5

State, comparing to **ConnectionState**, 89
State property
of OdbcConnection class, 79
of OleDbConnection class, 78
of **SqlConnection** class, 77
StateChange Connection class event, details of, 85
static cursors, explanation of, 264
storage capability of databases, understanding, 32
Stored Procedure Script database object template, description of, 301
stored procedures, 372–373
creating, 375–379
creating with arguments, 382–384
creating with arguments and return values, 385–387
introduction to, 373
reasons for use of, 374–375
renaming, 391
retrieving rows and output values from, 388–389
retrieving rows from, 382
running from code, 380–382
running from IDE, 379–380
running with arguments from IDE, 384
running with Oracle, 392–397
syntax testing of, 387
using RETURN statement with, 389–391
using with arguments, 384–385
viewing dependencies for, 391
streams
appending result sets to, 621–622
saving result sets to, 620–621
strong typing versus weak typing, example of, 196
SuspendBinding CurrencyManager class method, description of, 541
syntax for creating namespaces, 12
system-generated message queues, examining, 485–490
system journals, using with message queues, 486–487
system message queues, using with Server Explorer, 288
System .NET root namespace, explanation of, 11–12
System Stored Procedures. *See* stored procedures, 372–373
System* data-related namespaces, list of, 58
System.Data namespace, description of, 58
System.Data.OleDb namespace, description of, 58
System.Data.SqlClient namespace, description of, 58
System.DirectoryServices namespace, examining in Active Directory, 415–417
SystemException exception type, 350

T

*.tlb (type libraries), role in generating metadata for COM types, 277
T-SQL, adding linked Microsoft Exchange Server with, 442–443
tabbed documents mode, explanation of, 16
Table Designer
adding columns to tables with, 315–316
adding constraints with, 319–320
adding indexes and keys with, 317–319
creating relationships with, 320
creating tables with, 314–315
setting primary keys with, 316–317
using triggers with, 320
Table grid pane column in Query Designer, 326
Table property
of **DataColumn** class, 251
of **DataRow** class property, description of, 245
of **DataView** class, 239
Table Script database object template, description of, 301
table views, changing for database diagrams, 313
TableMappings DataAdapter class property, description of, 172
TableName DataTable class property, description of, 220
tables
adding and showing in tables with Query Designer, 322
adding with Database Designer, 309
creating with Database Designer, 309–313
creating with Table Designer, 314–315
deleting and removing with Database Designer, 309
in databases, definition of, 36
querying with SQLXML 2.0, 601–603
removing with diagram pane of Query Designer, 323
role in databases, 36
sizing automatically in database diagrams, 313

tables and columns, choosing in Data Form Wizard, 532
tables and views, choosing in Data Form Wizard, 531
Tables DataSet class property, description of, 199
TargetSite Exception class property, description of, 347
TargetSite property of **OdbcException** class, details of, 122
TargetSite property of **OleDbException** class, details of, 118
Task List in VS .NET IDE, explanation of, 27–28
tblLog table in UserMan database schema, explanation of, 52
tblRights table in UserMan database schema, explanation of, 52
tblUser columns for CUser class, list of, 514
tblUser table in UserMan database schema, explanation of, 52
tblUser_Update trigger, displaying, 409
tblUserRights table in UserMan database schema, explanation of, 52
TCP/IP (Transmission Control Protocol/Internet Protocol), role in connectionless and connection oriented programming, 450
TDS (tabular data stream) protocol, role in .NET Data Provider for SQL Server, 59
template string method, advisory in SQLXML 2.0, 617
templates
  advisory about, 606
  executing as string templates, 632–633
  specifying with SQLXML 2.0, 602–603
text editors
  using in VS .NET IDE, 24
  using SQL Editor, 334–337
throw statements, using, 356–357
Toolbox in VS .NET IDE, explanation of, 27
tools, modifying in VS .NET IDE, 24–25
ToString method
  of **OdbcException** class, 123
  of **OdbcConnection** class, 84
  of **OleDbConnection** class, 82
  of **OleDbException** class, 119
  of **SqlConnection** class, 81
transaction boundaries, defining in ADO.NET connected layer, 99
Transaction class
  methods of, 109
  properties of, 108
Transaction property
  of **OdbcCommand** class, 136
  of **OleDbCommand** class, 133
  of **SqlCommand** class, 131
transaction save points, using with **SqlTransaction** class, 104–105
transactional message queues, creating, 481
transactional private message queues, creating, 456
transactions
  aborting for message queues, 483
  beginning with nondefault isolation levels, 102
  committing for message queues, 482–483
  ending for message queues, 482–483
  functionality of, 98
  locking data at data sources with, 266–268
  nesting with **OdbcTransaction** class, 106
  nesting with **OleDbTransaction** class, 105–106
  nesting with **SqlTransaction** class, 104–105
  saving reference points in, 104–105
  starting for message queues, 481
  using with message queues, 484–485
trigger and catch concurrency violation, example of, 269–270
Trigger Script database object template, description of, 301
triggers
  advisory about, 406
  creating, 407–412
  introduction to, 403–404
  invoking and catching exception raised example, 410–411
  locating after saving, 409
  reasons for use of, 406
  using with Table Designer, 320
  viewing source for, 412
Trusted_Connection **ConnectionString** property value, details of, 74–75
try blocks, role in filtering exceptions, 355
try statements, role in exception handling, 344
Type Reference manifest component, description of, 10

typed data sets
in ADO.NET disconnected layer, 196
creating with DataSet Designer, 338–340
creating with XML Designer, 338

U

UID **ConnectionString** property value, details of, 74–75
Unique DataColumn class property, description of, 251
unmanaged code, definition of, 6
Unspecified member of **IsolationLevel** Enum, description of, 102
untyped data sets in ADO.NET disconnected layer, explanation of, 196
update data source using DataAdapter class, 206–209
Update DataAdapter class method, description of, 174
UPDATE queries, using Query Designer for, 331
UpdateCommand DataAdapter class property, description of, 172
UpdatedRowSource property
of **OdbcCommand** class, 137
of **OleDbCommand** class, 134
of **SqlCommand** class, 131
throws **ArgumentException** exception, 145–146
UpdateGrams, advisory about using with SQLXML 2.0 Managed classes, 619
URLs (uniform resource locators), advisory about querying with, 606
UsePropertyCache property of **DirectoryEntry** class, description of, 418
User ID **ConnectionString** property value, details of, 74–75
user permissions, setting programmatically, 501
user properties
adding to Active Directory, 428
editing with Active Directory, 427
manipulating with Active Directory, 428–429
UserMan Properties dialog box, displaying, 433–434
UserMan database schema, 51–52
UserMan example application
adding Active Directory objects to, 579–581
adding command object parameters to, 575–576
building as **DataSet** object, 256–261
calling Active Directory class in, 581
conformity to 5NF, 53
creating additional objects for, 581–582
creating select query in, 321
creating stored procedures for accessing database tables in, 584
creating triggers with, 407–412
creating views in, 399–400
creating Web client for, 583
creating Windows client for, 582
downloading classes for, 567
exception handling in, 586
exposing functionality with Web services, 585–586
filling data sets in, 576
hooking up public properties to data sets in, 576–578
identifying information and objects in, 565–566
instantiating and initializing data adapters in, 574–575
instantiating command objects in, 573–574
instantiating **DataSet** objects in, 574
introduction to, 51–53
logging to event log in, 583–584
opening and closing connection to, 570–573
optimizing, 583–586
passing connection object to various classes in, 584
retrieving contacts from Exchange Server 2000, 438–439
schema for, 51–52
securing Password column in user table of, 583
setting up database security for, 584–585
setting up triggers for enforcing business rules in, 584
specifying parent classes in, 578
using constants for table and column names in, 585
using local transactions with, 585
Web site for, 57
Username property of **DirectoryEntry** class, description of, 418
userPrincipalName retrieving from Active Directory with OLE DB .NET data providers, 432
users, retrieving SID with Active Directory, 435–436

using statement
importing namespaces into classes with, 58
uspGetRETURN_VALUE stored procedure, displaying, 390
uspGetUsers stored procedures, displaying, 381
uspGetUsersAndRights stored procedure, displaying, 386–388
uspGetUsersByLastName stored procedure, displaying, 383

V

Value property
of **SqlParameter** class, 612
of **XmlReader** class, 163
VBA (Visual Basic for Applications), implications of CLR for, 7
VBScript, implications of CLR for, 7
view ports, moving in database diagrams, 312–313
View Script database object template, description of, 301
views
choosing in Data Form Wizard, 531
creating, 399–400
creating on linked servers, 447
introduction to, 397
manipulating data from code in, 402–405
reasons for use of, 398
restrictions of, 398
running from IDE, 400–401
saving, 399
using from code, 401–405
virtual directory in SQLXML 2.0
managing with SQLXML 2.0, 593–599
restarting after making changes to, 600
virtual keyword in C#, relationship to OOP, 511
Virtual Name Configuration dialog box, displaying in SQLXML 2.0, 605
Visible Analyst DB Engineer, Web site for, 37
Visio for Enterprise Architect, designing databases with, 305
viwUser view, displaying, 403
VS .NET IDE (Integrated Development Environment). *See also* IDEs
built-in Object Browser for, 19–20
built-in Web browser functionality for, 17
Command Window modes for, 18–19
integrated debugger in, 20–21
integrated Help system in, 21–22
interface modes for, 16
macros in, 22–23
modifying, 24–25
Server Explorer in, 25–26
setting up data connections with, 26
sharing with all .NET languages, 13
Task List in, 27–28
text editors in, 24
Toolbox in, 27
upgraded deployment tools in, 23
VSA (Visual Studio for Applications), implications of CLR for, 7

W

weak typing versus strong typing, example of, 196
Web browser functionality feature of VS .NET IDE, explanation of, 17
Web clients, creating for UserMan example application, 583
Web Forms programming model for ASP.NET, explanation of, 5
Web forms, using data-bound controls with, 545–551
Web services, exposing functionality with, 585–586
Web sites
Active Directory, 415, 434
Antechinus C# Programming Editor, 15
Apress, 57
DiffGrams, 633
IMS (Information Management System), 34
LDAP display names for users, 425
LDAP query filters, 425
modeling relational databases, 37
ODBC (Open Data Base Connectivity), 96
ODBC standard escape sequences, 137
Oracle Database Project, 392
SQL Dialect, 434
SQL standards, 327
SQLXML 2.0 download, 592
UserMan, 57
UserMan Web client, 583
UserMan Windows client, 582
for VS .NET IDE Command Window, 19
X.500 directory standard, 413
XPath Web site, 605
Web user controls, data binding in, 557–561
Windows form controls, binding to data sources, 537–538

Windows forms
creating data-bound controls for, 544
examining binding context for, 527–528
role of **BindingContext** object in, 527, 538–539
role of **CurrencyManager** object in, 527
using data-bound controls with, 527
Workstation ID **ConnectionString** property value, details of, 74–75
WorkstationId SqlConnection class property, details of, 77
wrappers
definition of, 505
reasons for use of, 505
wrapping databases, 513–524
WriteXml DataSet class method, description of, 201
WriteXmlSchema DataSet class method, description of, 201

X

X.500 directory standard, Web site for, 413
XML Designer, creating typed data sets with, 338
XML documents, writing to disk, 617
XML (eXtensible Markup Language), using with **DataSet** class in ADO.NET disconnected layer, 195–196
XML Parser feature of SQLXML 2.0, resource for, 593
XML templates, sample for updating data sources, 632
XML Web services programming model for ASP.NET, explanation of, 5
XMLDATA SELECT . . . FOR XML statement keyword, description of, 588
XmlException, handling for **XmlReader** class, 170
XmlLang XmlReader class property, 163
XmlNodeReader class, usage of, 161
XmlReader class
closing, 169
exception handling, 170
methods of, 164–167
properties of, 161–163
reading rows in, 168–169
retrieving result sets in, 620
usage of, 161–168
XmlReader objects, declaring and instantiating, 168
XmlSpace XmlReader class property, 163
XmlTextReader class, usage of, 161
XmlValidatingReader class, usage of, 161
XPath queries, executing, 627–631
XPath, querying database tables with, 603
XPath Web site, 605
.xsd files, creation by Data Form Wizard, 534