CHAPTER 7

Advanced Connections and Authentication

In this chapter, you'll revisit the topic of connections, which I first addressed in Chapter 1, as well as look at some additional authentication methods above and beyond the basic username/password method that you've used up to this point. You'll examine the difference between privileged and non-privileged connections and learn how to connect to the database using a privileged connection.

A topic you may find confusing is how to use operating system authentication with the Oracle database. In this chapter, you work through the process of configuring the system to allow for Windows operating system authentication and create an operating system–authenticated connection. One of the underutilized features of the Oracle database is its ability to implement password expiration, password lockout and password changing. I'll address these topics and round out the chapter by examining connection pooling and multiplexing.

By this point, you're very familiar with the username/password method of authenticating to the database using the `Connection` object because that is the method all of the sample code has used thus far. One clear characteristic of this connection method is that the user must provide a username and password. In turn, the password is (ideally) maintained—that is, the password is changed on a regular basis. Sometimes some users see this maintenance as a chore or optional task. As a result, the administrator can implement password rules to enforce things such as required password changes. However, when you use operating system authentication, this activity becomes unnecessary in the database. When you use operating system authentication, you no longer need to maintain passwords in the database. In fact, you can't maintain them in the database because no password information is stored there.

Another type of connection that you haven't yet seen is what is known as a *privileged connection.* Of course, all connections have privileges of some sort. When I'm discussing privileged connections I mean a connection that has one of two system privileges:

The SYSDBA privilege: This is the highest administrative privilege an account can have. This privilege allows the user to perform all activities on a database.

The SYSOPER privilege: This privilege has slightly fewer abilities than the SYSDBA privilege. For example, a user connected with this privilege can't create a new database and may only perform certain types of database recovery operations.

In practice, typical .NET applications don't need to connect to a database with the SYSDBA or SYSOPER privileges. However, you may encounter times when you're creating an application that does need to connect with one of these privileges. For instance, an application that starts up or shuts down a database needs to use the SYSOPER privilege. Such an application can use the SYSDBA privilege, but that is an over-privileging if the application doesn't need to perform activities other than start up or shut down. If you're creating an application that creates an Oracle database from within the application, then you need to connect with the SYSDBA privilege because the SYSOPER privilege doesn't allow for this capability.

You'll also take a look at the ability to connect to the default database. I find this most useful in situations where I don't know the connection information for a database in advance. Another situation where this may come in handy is when databases are in different environments, such as a Sandbox, a Development, and a Test environment. When you allow the application to connect to the default database, you aren't required to configure the networking components as you did in Chapter 1 when you created your standard configuration. Of course, whether or not you want this is a question you and your database administrator need to discuss. I believe this is convenient or a short cut that you should use where appropriate in your environment.

The Default Database Connection

You can connect to a database without specifying the database in the connection string. When you connect in this manner, the database is known as a default database. The default database is determined by the value of the ORACLE_SID environment variable. The concept of the default database connection is only relevant on a server. As a result, the code you write in this section must run on the same machine as the default Oracle database. Recall that you may assign the value for the ORACLE_SID variable in several places:

The Registry: You can set the ORACLE_SID under the Oracle Home registry hive. On my system, the ORACLE_SID for my 10g installation is set under the `HKLM\SOFTWARE\ORACLE\KEY_OraDatabase10i` key.

The Environment Variables dialog: You can set the ORACLE_SID in the System Variables section.

The Environment Variables dialog: You can set the ORACLE_SID in the User Variables section.

In a Command Prompt window: Using the `set` command, you can set the ORACLE_SID value. For example, set `ORACLE_SID=LT10G`.

The locations where you may specify the value are listed in hierarchical order from lowest precedence to highest. For example, if the value is set in the registry, and also in the User Variables section of the Environment Variables dialog, the value specified in the User Variables section takes precedence over the value specified in the registry.

If you need to set values for the Oracle environment variables, I strongly recommend that you set them in the registry and then override them in a command prompt window if you need to. If you choose to set them using the Environment Variables dialog, be aware that it may cause unexpected behavior in the Oracle utilities that can be hard to debug. When multiple versions of the Oracle software are installed on a system, Oracle determines the environment or profile for each version based on the entries specified under the registry hive for each version of the software. This effectively segregates the environment configuration for each version into distinct profiles. When you specify a configuration variable using the Environment Variables dialog, that value overrides any entries in the registry for all versions of the software installed.

As a practical example, let's look at my system; I have Oracle versions 8i, 9i, and 10g installed. Under the registry keys for each of these versions, I've specified the ORACLE_SID for the corresponding version of the database: LT8I for the 8i database, LT9I for the 9i database, and LT10G for the 10g database. If I start the 9i version of SQL*Plus, Oracle uses the LT9I value for the ORACLE_SID because it's specified under the 9i home registry hive. If I specify the ORACLE_SID using the Environment Variables dialog as LT10G, when I start the 9i version of SQL*Plus, the ORACLE_SID is no longer LT9I; it is LT10G. This means that LT10G has become my default database for my 9i software. You'll probably find this confusing, if you are unaware of what has transpired.

If you need to examine the registry to determine the current value for the `ORACLE_SID` key, the key is located in the `HKLM\SOFTWARE\ORACLE\KEY_HomeName` hive for 10g and the `HKLM\SOFTWARE\ORACLE\HOME`*N* hive for previous releases. Refer to Chapter 1 if you need a refresher on the Oracle server architecture.

■ **TIP** Because the ORACLE_SID value is stored in the registry, you can't issue an `echo %ORACLE_SID%` command in a command prompt window to determine the value.

In order to illustrate how to connect to the default database on an Oracle host machine, I take the "HelloOracle" example I used in Chapter 1 and, with a very slight modification, enable it to connect to the default database. By doing so, I create a new Visual Studio solution named DefaultConnection.

■ **NOTE** Be sure to add a reference to the `Oracle.DataAccess.dll` assembly to the project and include the `using Oracle.DataAccess.Client;` directive at the top of the source file if you're creating a new solution.

Listing 7-1 contains the modified `Main` method from the HelloOracle example. You can find this sample (DefaultConnection) in this chapter's folder in the Downloads section of the Apress website (`www.apress.com`).

Listing 7-1. *The Modified Main Method from the HelloOracle Sample*

```
static void Main(string[] args)
{
  // Use the default database by omitting the Data Source connect
  // string attribute.
  String connString = "User Id=oranetuser;Password=demo";
  OracleConnection oraConn = new OracleConnection(connString);

  try
  {
    oraConn.Open();

    Console.WriteLine("\nHello, Default Oracle Database Here!\n");
    Console.WriteLine("Connection String: ");
    Console.WriteLine(oraConn.ConnectionString.ToString() + "\n");
    Console.WriteLine("Current Connection State: ");
    Console.WriteLine(oraConn.State.ToString() + "\n");
    Console.WriteLine("Oracle Database Server Version: ");
    Console.WriteLine(oraConn.ServerVersion.ToString());
  }
  catch (Exception ex)
  {
    Console.WriteLine("Error occured: " + ex.Message);
  }
  finally
  {
    if (oraConn.State == System.Data.ConnectionState.Open)
    {
      oraConn.Close();
    }
  }
}
```

To take advantage of connecting to the default database, the only change you need to make to the source code is in the definition of the connection string. By omitting the `Data Source` attribute from the connection string, you indicate to the Oracle Data Provider for .NET that you want to connect to the default database. Listing 7-2 contains the output that results from running this example.

Listing 7-2. *The Output of the DefaultConnection Sample*

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\DefaultConnection\bin\Debug>↵
DefaultConnection.exe

Hello, Default Oracle Database Here!

Connection String:
User Id=oranetuser;
```

```
Current Connection State:
Open

Oracle Database Server Version:
10g

C:\My Projects\ProOraNet\Oracle\C#\Chapter07\DefaultConnection\bin\Debug>
```

Notice in the output that the connection string that displays no longer contains the `Data Source` attribute. If you have more than one database on the machine you're using, it's easy to illustrate how you can set the ORACLE_SID environment variable in a command prompt window to override the value set in the registry. This also illustrates that you are, in fact, connecting to the default database as specified by the value of this variable. Listing 7-3 contains the output of the DefaultConnection sample after you change the value of the ORACLE_SID environment variable in the command prompt window.

Listing 7-3. *The Output of the DefaultConnection Example After Changing the ORACLE_SID*

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\DefaultConnection\bin\Debug>
set ORACLE_SID=LT8I

C:\My Projects\ProOraNet\Oracle\C#\Chapter07\DefaultConnection\bin\Debug>
DefaultConnection.exe

Hello, Default Oracle Database Here!

Connection String:
User Id=oranetuser;

Current Connection State:
Open

Oracle Database Server Version:
8.1.7.4.1

C:\My Projects\ProOraNet\Oracle\C#\Chapter07\DefaultConnection\bin\Debug>
```

In Listing 7-3, you see that I've overridden the value of the ORACLE_SID as specified in my registry (`LT10G`) with the value `LT8I`. The database specified by `LT8I` is, as I am sure you have guessed, an Oracle8i database that you can see in the Oracle Database Server Version informational output.

At this point, you may be asking yourself what would happen if the value for the `ORACLE_SID` was missing from the registry and wasn't specified elsewhere. In order to demonstrate what happens in this case, I've temporarily removed the `ORACLE_SID` key from my registry. Listing 7-4 illustrates what occurs when this is the case. In this listing, I also explicitly undefine the ORACLE_SID value in the command prompt window.

■ **CAUTION** If you remove the registry key or set it to no value, be sure you undo this operation immediately after testing.

Listing 7-4. *The Output of the DefaultConnection Sample with No ORACLE_SID Value Set*

```
C:\My Projects\ProOraNet\Chapter07\DefaultConnection\bin\Debug>set ORACLE_SID=

C:\My Projects\ProOraNet\Chapter07\DefaultConnection\bin\Debug>↵
DefaultConnection.exe
Error occured: ORA-12560: TNS:protocol adapter error

C:\My Projects\ProOraNet\Chapter07\DefaultConnection\bin\Debug>
```

As you can see in Listing 7-4, the Oracle client software immediately returns an error when it is unable to determine a value for the ORACLE_SID variable. To round out your exploration of connecting to the default database, I'll illustrate that the value specified for the ORACLE_SID must be a valid SID. This value can't be a TNS alias.

Listing 7-5 contains the results of running the sample code with a TNS alias specified in place of a valid SID. The error generated in this case is the same as when no value is present.

Listing 7-5. *The Output of the DefaultConnection Sample with an Invalid ORACLE_SID*

```
C:\My Projects\ProOraNet\Chapter07\DefaultConnection\bin\Debug>↵

set ORACLE_SID=ORANET

C:\My Projects\ProOraNet\Chapter07\DefaultConnection\bin\Debug>↵
DefaultConnection.exe
Error occured: ORA-12560: TNS:protocol adapter error

C:\My Projects\ProOraNet\Chapter07\DefaultConnection\bin\Debug>
```

Using tnsnames-less Connections

As the number of database installations increases and as systems become more diverse, maintaining a `tnsnames.ora` file for every database can become cumbersome. Oracle has several solutions to address these concerns including integrating with Microsoft Active Directory and its own Oracle Internet Directory product. However, a more simple, do-it-yourself solution that may be practical involves creating *tnsnames-less connections*.

What this means is that you connect to a database via the Oracle networking architecture but without using a TNS alias specified in a `tnsnames.ora` file. This is somewhat similar to the way a JDBC Thin driver connection string in Java works. Rather than placing the connection information into a file, you'll embed it into the connection string itself. Using the standard tnsnames method, you've been specifying a connection string such as

```
User Id=oranetuser; Password=demo; Data Source=oranet;
```

where `oranet` is a tnsnames alias. You can take the information that the `oranet` alias is used to represent from the tnsnames file and substitute it into the `Data Source` attribute of the connection string. This results in a more busy connection string for the `Connection` object, but I provide some abstraction for this in the sample code. If you use this technique, the connection string above would become

```
"User Id=oranetuser; Password=demo; Data Source=(DESCRIPTION = (ADDRESS_LIST =
 (ADDRESS = (PROTOCOL = tcp)(HOST = ridmrwillim1801)(PORT = 1521))
 (CONNECT_DATA = (SERVICE_NAME = LT10G.SAND)));"
```

In order to implement this connection technique, you need to know the host, the port, and the service name of the database to which you wish to connect. If you aren't using the service name to connect to a database, you could use the `SID` method instead. However, Oracle recommends the `service name` method for databases version 8i or later. The `service name` method supports Real Application Clusters and Grid systems whereas the `SID` method is directed toward single instance/host systems.

An end user can dynamically supply the `host`, `port`, and `service name` parameters at run-time, or you may specify them in an application-specific configuration file that you create. However, if you store the information in a configuration file, you are basically duplicating the creation of an entry in a `tnsnames.ora` file. As a result, I typically use dynamic values at run-time. How you get the values at run-time is up to you. An end user can dynamically supply them in an interactive application, or they you can read them from a database table—in a noninteractive, system-level application, for example. You'll implement this connection technique by creating a Windows Forms–based application, and you'll supply the required values interactively at run-time.

You can temporarily disable the `tnsnames.ora` file by renaming it to ensure that you're using your tnsnames-less connection. A new feature in the Oracle9i networking components (which carries on into 10g as well) is that the command line utilities tell you which file they used to carry out the request. One of these commands is the tnsping utility. If you've ever used the ping utility, this utility will be very familiar. Instead of pinging any network address, with the tnsping utility, you ping an Oracle listener service. The utility allows you to verify a pathway between Point A and Point B. Listing 7-6 illustrates the simple usage of the tnsping utility. Here, I simply ping the `oranet` TNS alias that you've been using as your standard.

Listing 7-6. *Using the tnsping Utility*

```
C:\>tnsping oranet

TNS Ping Utility for 32-bit Windows: Version 10.1.0.2.0 -
 Production on 12-JUL-2004 13:23:55

Copyright (c) 1997, 2003, Oracle.  All rights reserved.

Used parameter files:
c:\oracle\10.1\database\network\admin\sqlnet.ora
```

```
Used TNSNAMES adapter to resolve the alias
Attempting to contact (DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)
(HOST = ridmrwillim1801)(PORT = 1521))) (CONNECT_DATA =↵
 (SERVICE_NAME = LT10G.SAND)))
OK (20 msec)

C:\>
```

As you might recall from Chapter 1, the `sqlnet.ora` file is an optional file, but if it exists, it influences the Oracle networking environment. In Listing 7-6, you can see that the Oracle networking client has determined that my machine does have a `sqlnet.ora` file, and it reads it to extract configuration information. The `Used TNSNAMES adapter to resolve the alias` informational message indicates that my `sqlnet.ora` file contains a directive to use the `tnsnames.ora` file to resolve names.

Listing 7-7 shows what happens when the `tnsnames.ora` file is renamed, thus preventing the Oracle networking client from using it to resolve names. The `sqlnet.ora` file has been left intact.

Listing 7-7. *The Results of a tnsping After Renaming the tnsnames.ora File*

```
C:\oracle\10.1\database\network\admin>rename tnsnames.ora tnsnames.ora.bak

C:\oracle\10.1\database\network\admin>tnsping oranet

TNS Ping Utility for 32-bit Windows: Version 10.1.0.2.0 -↵
 Production on 12-JUL-2004 13:25:57

Copyright (c) 1997, 2003, Oracle.  All rights reserved.

Used parameter files:
c:\oracle\10.1\database\network\admin\sqlnet.ora

TNS-03505: Failed to resolve name

C:\oracle\10.1\database\network\admin>
```

Like the previous tnsping, this attempt reads the `sqlnet.ora` file to retrieve configuration information. However, the `tnsnames.ora` file no longer exists, so the networking client is unable to resolve the TNS alias `oranet`. Ordinarily you don't want this to happen. However, for this sample, you want to ensure that the Oracle networking client isn't able to resolve names. To create your tnsnames-less sample, you need to create a new Visual C# Windows application. I have called my Visual Studio solution DynamicConnection, as illustrated in Figure 7-1.

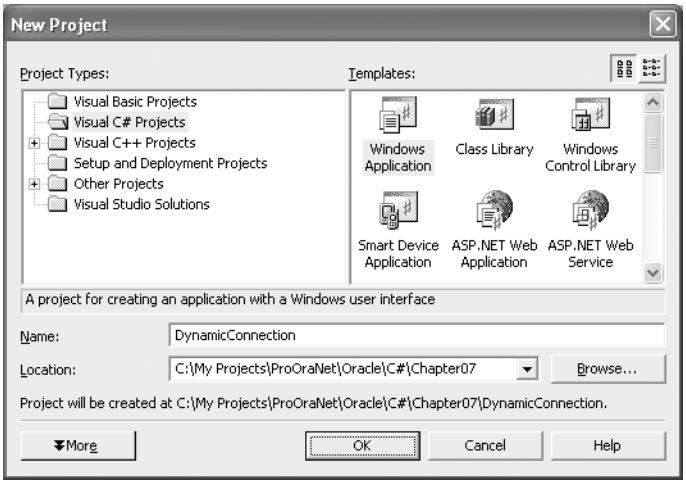

Figure 7-1. *The New Project dialog for the tnsnames-less sample*

Once you've created the project and generated the skeleton code, create the labels, text boxes, and buttons, as illustrated in Figure 7-2. (I've included relevant sections of the code here—for all the details, refer to the code in this chapter's folder on the Apress website.) Of course, you should add a reference to the Oracle Data Provider for .NET assembly to the project and include the relevant `using` directive in the source code for the form.

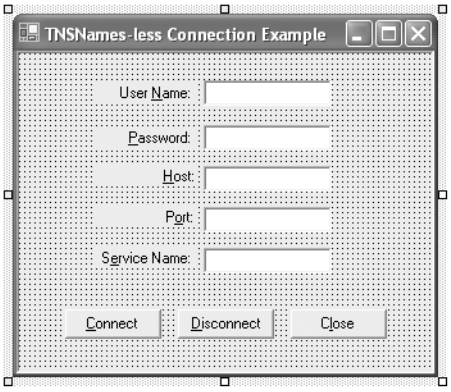

Figure 7-2. *The main form used in the tnsnames-less sample*

For this simple example, you create a single private method that takes the values it needs to generate a TNS connection string as parameters. You call this method by simply passing the values entered on the form. Listing 7-8 contains the private method and the code in the Connect button click event.

Listing 7-8. *The Main Code to Create a tnsnames-less Connection*

```
private void doDynamicConnection(
  string p_user,
  string p_password,
  string p_host,
  string p_port,
  string p_service_name)
{
  // build a tns connection string based on the inputs
  string l_data_source = "(DESCRIPTION=(ADDRESS_LIST=" +
    "(ADDRESS=(PROTOCOL=tcp)(HOST=" + p_host + ")" +
    "(PORT=" + p_port + ")))" +
    "(CONNECT_DATA=(SERVICE_NAME=" + p_service_name + ")))";

  // create the .NET provider connect string
  string l_connect_string = "User Id=" + p_user + ";" +
    "Password=" + p_password + ";" +
    "Data Source=" + l_data_source;

  // attempt to connect to the database
  OracleConnection oraConn = new OracleConnection(l_connect_string);

  try
  {
    oraConn.Open();

    // dislay a simple message box with our data source string
    MessageBox.Show(
      "Connected to data source: \n" + oraConn.DataSource,
      "Dynamic Connection Sample");
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message,"Error Occured");
  }
  finally
  {
    if (oraConn.State == System.Data.ConnectionState.Open)
    {
      oraConn.Close();
    }
  }
}

private void btnConnect_Click(object sender, System.EventArgs e)
{
  doDynamicConnection(
```

```
    txtUserName.Text,
    txtPassword.Text,
    txtHost.Text,
    txtPort.Text,
    txtServiceName.Text);
}
```

Although the code to create the `Data Source` may seem busy, basically, it dynamically creates the text that appears in a `tnsnames.ora` file based on the input values. Simply substituting this dynamically generated text in place of the TNS alias in the `Data Source` attribute of the connection string allows you to connect to a database based on values supplied at run-time. In Figure 7-3, I've completed the form with the same values I used to create the standard TNS alias.

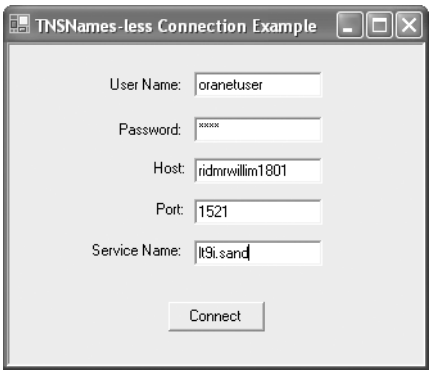

Figure 7-3. *The Dynamic Connection Sample form*

After you complete the form with the appropriate values, click the Connect button. A message box that displays the generated connection string appears. A simple dialog displays if an exception is trapped during the connection attempt. Figure 7-4 shows the dialog that displays given the input values supplied in Figure 7-2.

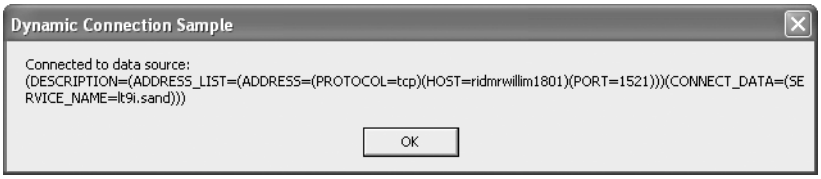

Figure 7-4. *The run-time results of the Dynamic Connection Sample*

The ability to create database connections dynamically at run-time is a powerful capability. You'll find this technique especially useful in situations such as in a system where maintaining a `tnsnames.ora` file isn't feasible or desirable. One example of such a system is a central repository that connects to various target databases. By creating a database table that contains all the information you need to connect to the target databases, you can easily create a database-driven

solution that connects dynamically at run-time to subscribed databases. If you move the functionality demonstrated in the private method in your sample form into a library, multiple applications can take advantage of this technique.

Privileged and Non-Privileged Connections

The terms *privileged* and *non-privileged,* when used in conjunction with a connection, don't refer to any specific privileges a user account may or may not have been granted in a database. This is because the privileged and non-privileged refer to *system-level* privileges and not *database-level* privileges. This is most evident by the fact that these privileges allow you to make a connection even if the database itself is not started or opened. Two system-level privileges may be granted: SYSDBA or SYSOPER. Any connection that doesn't utilize either of these privileges is said to be non-privileged. Even though these are system-level privileges, they are, from time to time, referred to as *connection types* or *connection privileges.* This is because when you connect using one of these system-level privileges in a utility such as SQL*Plus, the clause `as SYSDBA` or `as SYSOPER` is specified along with the connection string.

The SYSDBA and SYSOPER Privileges

There are a few differences between these two privileges. The SYSDBA privilege is the highest privilege an account may have. An account that has this SYSDBA privilege can perform *any* operation, and for this reason, you should connect to the database using this privilege sparingly and only when you need it. For example, you'll need this privilege when you create a new database or perform certain types of database recovery operations.

The SYSOPER privilege is very similar to the SYSDBA privilege. The only system-level functions the SYSDBA privilege provides that the SYSOPER privilege doesn't are those that let the user create a database and change the character set of a database. As a result, you should take the same care when you grant this privilege that you do with the SYSDBA privilege.

However, one subtle difference between these two is important. When you make a connection using either of these system-level privileges, the connection doesn't utilize the schema normally associated with the account making the connection. For connections you establish using the SYSDBA privilege, the schema used is the SYS schema. Connections that you establish using the SYSOPER privilege, on the other hand, use the PUBLIC schema. This effectively prevents connections that use SYSOPER from being able to view or manipulate data in private schemas.

If you have tried to connect as the user SYS, you may have received the error depicted in Listing 7-9.

Listing 7-9. *Attempting to Connect as SYS in SQL*Plus*

```
C:\>sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 13:31:35 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

SQL> connect sys
```

```
Enter password:
ERROR:
ORA-28009: connection to sys should be as sysdba or sysoper


SQL>
```

The default configuration of Oracle, as installed in the Appendix, is configured so that SYS must explicitly connect as either SYSDBA or SYSOPER. In order to connect as SYSDBA, simply specify `as SYSDBA` in the connection string.

■ **CAUTION** Exercise proper care when you're connecting as SYSDBA. This is a fully privileged connection and should never be used for a normal connection. In addition, make sure you coordinate with your DBA if you're creating an application that makes such connections to the database—any administrator would want to know about an application that makes connections of this type.

Because the SYSDBA and SYSOPER privileges are so powerful, you'll grant them to your administrative user for the samples in this section, and then you'll revoke them when you have completed the samples. Let's begin by connecting as SYS using the SYSDBA privilege. This is one of the few times that you use the SYS user. In this case, it is necessary to connect as SYS in order to grant the SYSDBA privilege. Listing 7-10 demonstrates the process of connecting as SYS and granting the SYSDBA and the SYSOPER privileges. Notice that you're using the default database connection as discussed in the last section.

Listing 7-10. *Connecting as SYS and Granting the SYSDBA and SYSOPER Privileges*

```
C:\>sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 13:33:12 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

SQL> connect sys as sysdba
Enter password:
Connected.
SQL> grant sysdba to oranetadmin;

Grant succeeded.

SQL> grant sysoper to oranetadmin;

Grant succeeded.

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
```

```
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

Now that your administrative user has the SYSDBA and the SYSOPER privileges, you can connect with each privilege. When you connect as a privileged connection, as with the SYS user, you must use the as clause to specify which privilege you wish to use for your connection. If you omit the clause, you'll simply connect with your normal, non-privileged account. Listing 7-11 demonstrates these concepts.

■ **NOTE** You must install and configure the database and networking components in the same fashion as the setup in the Appendix for this to work properly. This doesn't mean that you must follow the installation steps in the Appendix, only that your configuration must match it. This installation is a preconfigured installation type and it results in an installation configuration that is common.

Listing 7-11. *Connecting with the SYSDBA and SYSOPER Privileges*

```
C:\>sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 13:35:02 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

SQL> connect oranetadmin
Enter password:
Connected.
SQL> /* format username column as alphanumeric with width of 16 */
SQL> /* format database column as alphanumeric with width of 24 */
SQL> COL USERNAME FORMAT A16
SQL> COL DATABASE FORMAT A24
SQL> SELECT   A.USERNAME,
  2           B.GLOBAL_NAME DATABASE
  3  FROM     USER_USERS  A,
  4           GLOBAL_NAME B;

USERNAME         DATABASE
---------------- ------------------------
ORANETADMIN      LT10G.SAND

1 row selected.

SQL> connect oranetadmin as sysdba
Enter password:
Connected.
```

```
SQL> SELECT   A.USERNAME,
  2           B.GLOBAL_NAME DATABASE
  3  FROM     USER_USERS  A,
  4           GLOBAL_NAME B;

USERNAME         DATABASE
---------------- ------------------------
SYS              LT10G.SAND

1 row selected.

SQL> connect oranetadmin as sysoper
Enter password:
Connected.
SQL> SELECT   A.USERNAME,
  2           B.GLOBAL_NAME DATABASE
  3  FROM     USER_USERS  A,
  4           GLOBAL_NAME B;

no rows selected

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

As illustrated in Listing 7-11, when you connect with no system-level privilege specified, you connect to your normal schema of ORANETADMIN. However, when you connect with SYSDBA or SYSOPER, things begin to get interesting. When you connect with the SYSDBA privilege, your query to determine who you are returns SYS as the username. When you connect with the SYSOPER privilege, your query returns no rows. This is because you're connected to the special schema PUBLIC and not to a real user. To further illustrate this, you'll connect with no system-level privileges, create a table, insert a record, and attempt to query the table. Listing 7-12 contains the code to illustrate this.

Listing 7-12. *Schema Object Visibility When Connecting with System-Level Privileges*

```
C:\>sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 13:38:09 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

SQL> connect oranetadmin
Enter password:
Connected.
```

```
SQL> create table t
  2  (
  3    c varchar2(32)
  4  );

Table created.

SQL> insert into t values ('Can you see me?');

1 row created.

SQL> commit;

Commit complete.

SQL> select c from t;

C
--------------------------------
Can you see me?

1 row selected.

SQL> connect oranetadmin as sysdba
Enter password:
Connected.
SQL> select c from t;
select c from t
             *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> select c from oranetadmin.t;

C
--------------------------------
Can you see me?

1 row selected.

SQL> connect oranetadmin as sysoper
Enter password:
Connected.
SQL> select c from t;
select c from t
             *
```

```
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> select c from oranetadmin.t;
select c from oranetadmin.t
                          *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

As Listing 7-12 illustrates, all goes as expected when you connect with no system-level privileges. You operate as yourself in this connection. When you connect with the SYSDBA privilege, you get an error when you query your table. This is because you're now connected with the SYS schema. However, by specifying the table owner, you are able to successfully query your table. When connected with the SYSOPER privilege, you simply aren't able to see the table at all.

■ **NOTE** If you execute the `grant select on t to public` command, you can query your table while you're connected with the SYSOPER privilege.

Connecting as a Privileged User

Now that you understand the SYSDBA and SYSOPER privileges, you can examine how to connect from a .NET application using either of these privileges. Under normal circumstances, your applications don't need to connect with either of these system privileges, but if your application needs to shut down or start up a database, for example, it needs to connect with either of these privileges.

Continuing with the trend of keeping things as simple as possible to focus on the topic at hand, you'll create a console application that simply connects as a privileged user, issues the simple "Who am I?" query to verify that you have connected as a privileged user, displays the results, and exits. Listing 7-13 contains the major points in the code for the PrivilegedConnection sample (which you can find in this chapter's folder in the Downloads section of the Apress website). To connect with either privilege, your code needs to include a new attribute in the connection string. The attribute is the `DBA Privilege` attribute, and it must be assigned either SYSDBA or SYSOPER to be valid.

Listing 7-13. *The SYSDBA and SYSOPER Privilege Test Code*

```
static void Main(string[] args)
{
  Class1 theClass = new Class1();

  // our "basic" connection string
  string conn_1 = "User Id=oranetadmin;" +
    "Password=demo;" +
    "Data Source=oranet";

  // our "sysdba" connection string
  string conn_2 = "User Id=oranetadmin;" +
    "Password=demo;" +
    "Data Source=oranet;" +
    "DBA Privilege=SYSDBA";

  // our "sysoper" connection string
  string conn_3 = "User Id=oranetadmin;" +
    "Password=demo;" +
    "Data Source=oranet;" +
    "DBA Privilege=SYSOPER";

  // our "who am i?" query
  string l_sql = "select a.username, " +
    "b.global_name database " +
    "from user_users a, " +
    "global_name b";

  theClass.privilegeTest(conn_1, l_sql);
  theClass.privilegeTest(conn_2, l_sql);
  theClass.privilegeTest(conn_3, l_sql);
}

void privilegeTest(string p_connect, string p_sql)
{
  // a simple little helper method
  // gets a connection, executes the sql statement,
  // and prints the results (if any) to the console
  OracleCommand oraCmd;
  OracleDataReader oraReader;

  OracleConnection oraConn = new OracleConnection(p_connect);

  try
  {
```

```
    oraConn.Open();

    oraCmd = new OracleCommand(p_sql,oraConn);

    oraReader = oraCmd.ExecuteReader();

    while (oraReader.Read())
    {
      Console.WriteLine("User: ");
      Console.WriteLine("  " + oraReader.GetString(0));
      Console.WriteLine("Database: ");
      Console.WriteLine("  " + oraReader.GetString(1) + "\n");
    }
  }
  catch (Exception ex)
  {
    Console.WriteLine("Error occured: " + ex.Message);
  }
  finally
  {
    if (oraConn.State == System.Data.ConnectionState.Open)
    {
      oraConn.Close();
    }
  }
}
```

After you create the project and successfully compile the code, a sample test should yield results similar to those in Listing 7-14.

Listing 7-14. *The Output from the Privilege Test*

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\PrivilegedConnection\bin\Debug>↵
PrivilegedConnection.exe
User:
  ORANETADMIN
Database:
  LT10G.SAND

User:
  SYS
Database:
  LT10G.SAND


C:\My Projects\ProOraNet\Oracle\C#\Chapter07\PrivilegedConnection\bin\Debug>
```

Recall that when you connect with the SYSOPER privilege, you connect to the special PUBLIC schema. Because you connect to the PUBLIC schema, your "Who am I?" query returns no rows, and, therefore, there is output. Now that you've completed your exploration of the SYSDBA and SYSOPER privileges, you'll revoke them from our administrative user. Listing 7-15 illustrates this process.

Listing 7-15. *Revoking the SYSDBA and SYSOPER Privileges*

```
C:\>sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 13:43:24 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

SQL> connect sys as sysdba
Enter password:
Connected.
SQL> revoke sysdba from oranetadmin;

Revoke succeeded.

SQL> revoke sysoper from oranetadmin;

Revoke succeeded.

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

Connecting via Operating System Authentication

Connecting to a database via operating system authentication involves Oracle authenticating the user account using Windows authentication. Operating system authentication isn't limited to the Windows platform; however, Windows is the only platform I investigated in this section. When you connect via operating system authentication, you don't need the user ID and password. The convention you use to indicate that operating system authentication should be employed is a single forward slash (`/`) symbol. Rather than specifying a username in the connection string, such as `User Id=oranetuser`, specify the user ID as `User Id=/`.

If you aren't working in a stand-alone environment as I am, you may need to coordinate efforts between your database administrator and your operating system administrator in order to enable operating system authentication. There are two classes of operating system authentication you can use: enterprise users and external users. To employ the enterprise users method of authentication, you must have a directory server. Therefore, you need to create an external user authentication scheme.

In this section, you configure operating system authentication using the external user scheme and ensure that everything is working properly by using SQL*Plus as your litmus test. Once you verify that your configuration is working correctly, use your "Who am I?" console application to illustrate connecting via operating system authentication in a .NET application.

Configuring Oracle for Operating System Authentication

In order to properly connect via operating system authentication, you must ensure that Oracle is configured to allow such connections. By default, the `sqlnet.ora` file contains the entry that enables operating system authentication. As we discussed in Chapter 1, the `SQLNET.AUTHENTICATION_SERVICES = (NTS)` entry allows for authentication by the operating system. Listing 7-16 is a representative `sqlnet.ora` file.

Listing 7-16. *A sqlnet.ora File*

```
C:\oracle\10.1\database\network\admin>type sqlnet.ora
# SQLNET.ORA Network Configuration File:
 C:\oracle\10.1\database\network\admin\sqlnet.ora
# Generated by Oracle configuration tools.

SQLNET.AUTHENTICATION_SERVICES= (NTS)

NAMES.DIRECTORY_PATH= (TNSNAMES)

C:\oracle\10.1\database\network\admin>
```

If your `sqlnet.ora` file doesn't contain the entry for `SQLNET.AUTHENTICATION_SERVICES`, you must add it to the file. Although the `sqlnet.ora` file influences the behavior of the Oracle networking components, some parameters influence the behavior of the database itself. These parameters reside in what is known as the init file or the spfile. The spfile exists for Oracle9i and Oracle10g databases, whereas you must use the init file in earlier versions. It's still possible to use an init file in 9i and 10g if you manually configure your database and installation process, but the preconfigured install types and the Oracle tools such as the Database Creation Assistant create a spfile.

You won't make modifications to these parameters here; however, you do need to know the value of one of them, `os_authent_prefix`, to properly configure your authentication example. Oracle uses this parameter when it is authenticating external users. As the name of the parameter implies, the value of this parameter is prefixed to the operating system username. Listing 7-17 illustrates one method of determining the value of this parameter.

Listing 7-17. *Determining the Value of the os_authent_prefix Parameter*

```
C:\>sqlplus oranetadmin

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 13:46:03 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.
```

```
Enter password:

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> show parameter os_authent_prefix

NAME                           TYPE        VALUE
------------------------------ ----------- ------------------------------
os_authent_prefix              string      OPS$
SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

You can see that, on my system, the value of this parameter is `OPS$`. The value may be different on your system or it may not be specified at all. If the value isn't specified, it doesn't mean that operating system authentication won't work or isn't available, it just means that no value will be prefixed to an operating system username. The Windows username that I use on my system is `willim18`, and my host name is `ridmrwillim1801`. Therefore, when I authenticate using operating system authentication to my database, I authenticate as `OPS$RIDMRWILLIM1801\WILLIM18`.

Because I authenticate as `OPS$RIDMRWILLIM1801\WILLIM18`, I need to create that user in the Oracle database. Listing 7-18 illustrates the process of creating a user for operating system authentication.

Listing 7-18. *Creating a User for Operating System Authentication*

```
C:\>sqlplus oranetadmin

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 13:47:28 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

Enter password:

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> create user "OPS$RIDMRWILLIM1801\WILLIM18" identified externally
  2  default tablespace users
  3  temporary tablespace temp
  4  quota unlimited on users;
```

```
User created.

SQL> grant connect to "OPS$RIDMRWILLIM18O1\WILLIM18";

Grant succeeded.

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

As you can see, I used the administrative user to log in to SQL*Plus and manually create a new user.

■ **TIP** If you are part of a Windows domain, when you create the new user, you should include the domain name in the username. For example, if I was in a domain called Liverpool, my user would be OPS$LIVERPOOL\WILLIM18.

Testing Operating System Authentication

Now that you've successfully created your user to be authenticated by the operating system, you can test this in SQL*Plus quite easily. You only need to specify a forward slash (/) for your connect string. Then execute your "Who am I?" script to verify that you have connected as expected. Listing 7-19 illustrates this process.

Listing 7-19. *Testing Operating System Authentication*

```
C:\>sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 13:49:15 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

SQL> connect /
Connected.
SQL> COL USERNAME FORMAT A32
SQL> COL DATABASE FORMAT A24
SQL> SELECT   A.USERNAME,
  2           B.GLOBAL_NAME DATABASE
  3  FROM     USER_USERS  A,
  4           GLOBAL_NAME B;

USERNAME                         DATABASE
```

```
------------------------------ ------------------------
OPS$RIDMRWILLIM1801\WILLIM18     LT10G.SAND

1 row selected.

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

By specifying only a / for your connect string, you've successfully connected to your default database using operating system authentication. If you wish to connect to a database using its TNS alias and operating system authentication, you simply supply the TNS alias as part of the connect string as we discussed earlier. Of course, when you connect to a TNS alias, your account must be set up properly in the destination database as it was in Listing 7-19. The sample .NET code you develop in the following section (Listing 7-20) illustrates both methods of connecting.

Operating System Authentication in .NET

Implementing operating system authentication in .NET code is trivial. Most of the work is in the setup and verification to make sure that the authentication is working as you expect it to. Because you've already set up and verified that your External User account is working as expected, in this section, you implement your "Who am I?" sample query using operating system authentication to the default database as well as your standard TNS alias. Listing 7-20 contains the core code you need to implement your sample. Of course, a reference to the `Oracle.DataAccess.dll` assembly and a `using Oracle.DataAccess.Client;` are included in the OSAuthenticatedConnection project (which you can access from this chapter's folder in the Downloads section of the Apress website).

Listing 7-20. *The Core Code for Testing Operating System Authentication*

```
static void Main(string[] args)
{
  Class1 theClass = new Class1();

  // our "default" database connection string
  string conn_1 = "User Id=/";

  // our "tns alias" database connection string
  string conn_2 = "User Id=/;" +
    "Data Source=oranet";

  // our "who am i?" query
  string l_sql = "select a.username, " +
    "b.global_name database " +
```

```
    "from user_users a, " +
    "global_name b";

  Console.WriteLine("Using the default database...");
  theClass.authenticationTest(conn_1, l_sql);

  Console.WriteLine("Using the tns alias...");
  theClass.authenticationTest(conn_2, l_sql);
}

void authenticationTest(string p_connect, string p_sql)
{
  // a simple little helper method
  // gets a connection, executes the sql statement,
  // and prints the results to the console
  OracleCommand oraCmd;
  OracleDataReader oraReader;

  OracleConnection oraConn = new OracleConnection(p_connect);

  try
  {
    oraConn.Open();

    oraCmd = new OracleCommand(p_sql,oraConn);

    oraReader = oraCmd.ExecuteReader();

    while (oraReader.Read())
    {
      Console.WriteLine("User: ");
      Console.WriteLine("  " + oraReader.GetString(0));
      Console.WriteLine("Database: ");
      Console.WriteLine("  " + oraReader.GetString(1) + "\n");
    }
  }
  catch (Exception ex)
  {
    Console.WriteLine("Error occured: " + ex.Message);
  }
  finally
  {
    if (oraConn.State == System.Data.ConnectionState.Open)
    {
      oraConn.Close();
    }
  }
}
```

Running the sample code should produce results that are appropriate for your system and resemble those in Listing 7-21.

Listing 7-21. *The Output of the Operating System Authentication Test*

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\OSAuthenticatedConnection\bin\Debug>
OSAuthenticatedConnection.exe
Using the default database...
User:
  OPS$RIDMRWILLIM1801\WILLIM18
Database:
  LT10G.SAND

Using the tns alias...
User:
  OPS$RIDMRWILLIM1801\WILLIM18
Database:
  LT10G.SAND


C:\My Projects\ProOraNet\Oracle\C#\Chapter07\OSAuthenticatedConnection\bin\Debug>
```

As expected, the results of your "Who am I?" query display the same data when connecting via operating system authentication to both the default database and that same database specified as a TNS alias. Using operating system authentication is a viable method for not having to maintain a password for a database user as we discussed earlier. However, if you wish to have a finer grained control over the password, then Oracle can accommodate that as well when you're using database authentication.

Password Management

The topic of password management can be rather broad. I will, therefore, limit the discussion to three areas:

Changing a Database Password: You'll look at changing a password via SQL*Plus as well as in .NET code.

Dealing with Expired Database Passwords: Passwords can expire and thus need to be changed.

Locking Out a Database Password: Accounts can be locked if password rules created by the database administrator are violated.

These are the most common areas for dealing with password management. One common misconception regarding Oracle passwords is that what is stored in the database isn't the actual password. Passwords in Oracle are really one-way hash values that result from an internal algorithm that incorporates the password as defined by the user. It is, therefore, not possible to reverse-engineer an Oracle password—no password is stored in the database, only the result of the one-way hash algorithm.

Changing a Database Password

A database user can change their own password. If the database user has the appropriate database privilege (`alter user`), they may change the password for other users as well.

■ **NOTE** The `alter user` privilege allows for more than just changing another user's password. You can find the complete list of attributes that can be changed by `alter user` under the `alter user` SQL statement reference in the "Database SQL Reference" in the documentation set.

The command you use to change a password is simply `alter user <username> identified by <password>`. Listing 7-22 illustrates how to do this via SQL*Plus.

Listing 7-22. *Changing a Password in SQL*Plus*

```
C:\>sqlplus oranetuser@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 13:53:30 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

Enter password:

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> alter user oranetuser identified by newpass;

User altered.

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

The database user, oranetuser, now has the password newpass. You may wish to change the password back to what it was before by simply executing the command and specifying the old password. Listing 7-23 contains the core code for a Windows Forms application named PasswordChange that allows you to change the password back (or to any other value) if you wish. This sample doesn't allow you to change the password for any user in the database—only for the oranetuser user.

Listing 7-23. *Simple Code to Change a Password*

```
private void btnChangePassword_Click(object sender, System.EventArgs e)
{
  if (txtNewPassword.Text != txtConfirmPassword.Text)
  {
    MessageBox.Show("New passwords do not match.", "Password Mismatch");

    return;
  }

  string l_connect = "User Id=" + txtUserName.Text + ";" +
    "Password=" + txtCurrentPassword.Text + ";" +
    "Data Source=" + txtTNSAlias.Text;

  string l_sql = "alter user " + txtUserName.Text + " " +
    "identified by " + txtNewPassword.Text;

  OracleCommand cmd;
  OracleConnection oraConn = new OracleConnection(l_connect);

  try
  {
    oraConn.Open();

    cmd = new OracleCommand(l_sql,oraConn);

    cmd.ExecuteNonQuery();

    MessageBox.Show("Password changed successfully.", "Password Changed");
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message, "Error Occured");
  }
  finally
  {
    if (oraConn.State == System.Data.ConnectionState.Open)
    {
      oraConn.Close();
    }
  }
}
```

Figure 7-5 shows the simple form you'd use to capture the relevant information you need to change the database password.

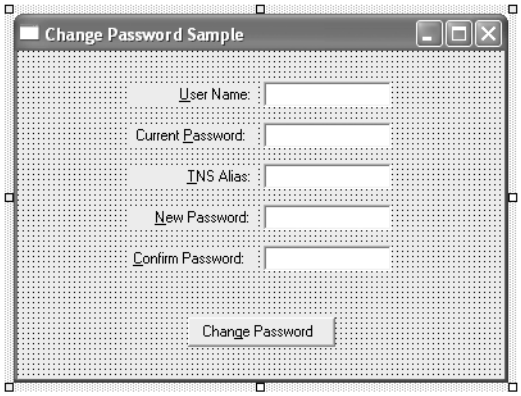

Figure 7-5. *The design-time representation of the form*

Figure 7-6 shows the form at run-time. Because you changed the password for the oranetuser in SQL*Plus earlier, I changed it back to the value of demo, as it was previously.

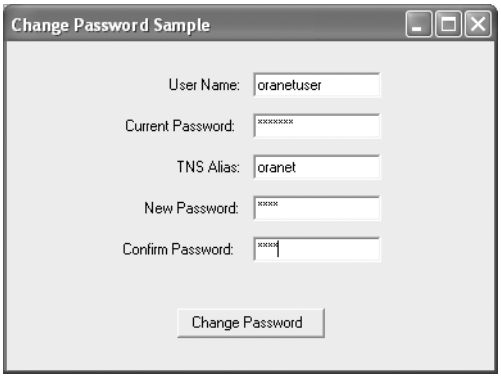

Figure 7-6. *The run-time representation of the form*

Figure 7-7 shows the confirmation message that states that the password was changed successfully.

Figure 7-7. *The confirmation dialog*

To verify that the password for oranetuser was successfully changed from newpass to demo, create a SQL*Plus session and specify the new password. Listing 7-24 illustrates this process (in order to make it explicit that the password was successfully changed, I specify the password as part of the connection string in SQL*Plus).

Listing 7-24. *Verifying That the Password Was Changed*

```
C:\>sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 13:56:07 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

SQL> connect oranetuser/demo@oranet
Connected.
SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

Dealing with Expired Database Passwords

If the database administrator for your system elects to implement password expiration, it's possible that the password for your user may expire. In this section, you simulate this situation by using SQL*Plus and manually expiring a password. After I demonstrate the concept in SQL*Plus, you implement the .NET code to catch the expired password exception and then allow the password to be changed. Listing 7-25 illustrates the process of manually expiring a password from SQL*Plus. Notice that you connect as your administrative user to expire the password for your typical-privileged user.

Listing 7-25. *Manually Expiring a Password*

```
C:\>sqlplus oranetadmin@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 14:01:20 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

Enter password:

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> alter user oranetuser password expire;
```

```
User altered.

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

The password for oranetuser has expired. When the user attempts their next connection, Oracle detects that the password has expired and triggers a prompt for a new password. Listing 7-26 illustrates this process.

Listing 7-26. *Detection and Change of the Expired Password*

```
C:\>sqlplus oranetuser@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 14:02:38 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

Enter password:
ERROR:
ORA-28001: the password has expired


Changing password for oranetuser
New password:
Retype new password:
Password changed

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

■ **NOTE** The password that is initially entered must be the correct expired password. If an invalid password is entered, you receive the standard "invalid username or password" error message instead of being prompted for a new password.

To demonstrate how to trap and process the expired password in .NET, I clone the simple Windows Form application used to change the password. The form itself doesn't change. I simply change the code inside the button click event. Listing 7-27 represents the code that detects the expired password condition and connects with a new password. The new project is called PasswordExpiration (see this chapter's folder on the Downloads section of the Apress website).

Listing 7-27. *Changing an Expired Password*

```
private void btnChangePassword_Click(object sender, System.EventArgs e)
{
  // display a simple message if the "new" and
  // the "confirm" passwords do not match
  if (txtNewPassword.Text != txtConfirmPassword.Text)
  {
    MessageBox.Show("New passwords do not match.", "Password Mismatch");

    return;
  }

  // build a connect string based on the user input
  string l_connect = "User Id=" + txtUserName.Text + ";" +
    "Password=" + txtCurrentPassword.Text + ";" +
    "Data Source=" + txtTNSAlias.Text;

  OracleConnection oraConn = new OracleConnection(l_connect);

  try
  {
    // attempt to open a connection
    // this should fail since we have expired the password
    oraConn.Open();
  }
  catch (OracleException ex)
  {
    // trap the "password is expired" error code
    if (ex.Number == 28001)
    {
      // dislay a simple marker to indicate we trapped the error
      MessageBox.Show("Trapped Expired Password", "Expired Password");

      // this method changes the expired password
      oraConn.OpenWithNewPassword(txtNewPassword.Text);
```

```
      // display a simple marker to indicate password changed
      MessageBox.Show("Changed Expired Password", "Expired Password");
    }
    else
    {
      MessageBox.Show(ex.Message, "Error Occured");
    }
  }
  finally
  {
    if (oraConn.State == System.Data.ConnectionState.Open)
    {
      oraConn.Close();
    }
  }
}
```

In Figure 7-8, you can see the form at run-time.

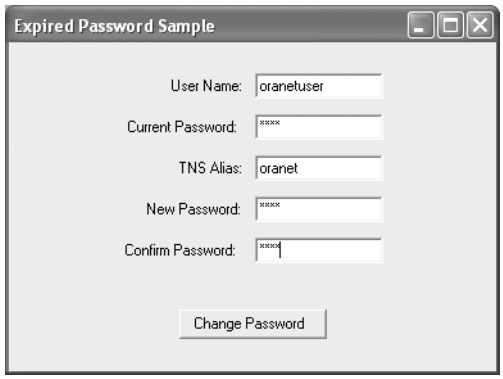

Figure 7-8. *The Expired Password Sample form*

By completing the text field in the form and clicking the Change Password button, you should see the dialog depicted in Figure 7-9.

Figure 7-9. *The Expired Password detected dialog*

After dismissing the dialog that informs you that the expired password condition was detected, you should see the dialog depicted in Figure 7-10.

Figure 7-10. *The Expired Password changed dialog*

To verify that the expired password was correctly changed, use SQL*Plus as illustrated in Listing 7-28.

Listing 7-28. *Verifying That the Expired Password Was Changed*

```
C:\>sqlplus oranetuser@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 14:05:11 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

Enter password:

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

The sample code, as presented here, doesn't verify that the new password you entered was different from the old one. You should coordinate any password rules such as no password reuse with your database administrator if you're using the password expiration feature. Oracle has the ability to enforce a variety of password rules; however, this is a subject outside the scope of this book because the creation of the password rules in the database relates to your database administrator more than it does to us as developers. The definition of those rules, on the other hand, is something that application developers and corporate security departments are frequently involved in creating. If your database administrator has implemented password rules, it's possible that, in addition to password expiration, you may encounter a user account that has been locked.

Locking Out a Database Account

In addition to an expired password, it's possible that an account can become locked out as a result of database password rules that may be implemented. For example, if the database administrator implements rules that dictate an account becoming locked after three unsuccessful login attempts, and a user inputs an incorrect password three times in succession, the account is then locked out. Unlike the case of an expired password, no method allows a simultaneous change of the password value as well as the password or account state. Once an account is locked, it must be explicitly unlocked. Typically, the database administrator (or similar privileged user) performs the unlocking. This is primarily due to the fact that if an account becomes locked, the database administrator is likely to know why. If an application simply allowed a user to get into a locked account state and then unlock the account, there is little point in implementing the rules that allow the lock to occur in the first place.

In this section, once again you use SQL*Plus to manually place your account into the desired state; then you implement the functionality in .NET code to detect the situation and notify the user. Here, you only detect the locked state and alert the user that the account is locked. If you wish to implement the ability to unlock an account in your application, I suggest you discuss this with your database administrator prior to doing so. Listing 7-29 illustrates how to lock an account using your administrative user in SQL*Plus.

■ **NOTE** The act of locking or unlocking an account doesn't affect the password per se. I included it in this section because, from a user perspective, a locked account may be interpreted as a password issue.

Listing 7-29. *Using SQL*Plus to Lock an Account*

```
C:\>sqlplus oranetadmin@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 14:06:29 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

Enter password:

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> alter user oranetuser account lock;

User altered.

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

At this point, your oranetuser account is locked. Any attempt to connect to the database generates a trappable error as illustrated in Listing 7-30.

Listing 7-30. *Attempting to Connect to a Locked Account*

```
C:\>sqlplus /nolog

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 14:07:30 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

SQL> connect oranetuser@oranet
Enter password:
ERROR:
ORA-28000: the account is locked


SQL> exit

C:\>
```

As you can see in Listing 7-30, Oracle generates an ORA-28000 error message when it detects an attempt to connect with a locked account. Your test code from the PasswordLocked project in Listing 7-31 (see this chapter's folder in the Downloads section of the Apress website for the complete sample code) takes advantage of this fact to trap the condition and display a simple dialog that indicates that the condition has been trapped.

Listing 7-31. *The Account Locked Test Code*

```
private void btnConnect_Click(object sender, System.EventArgs e)
{
  // build a connect string based on the user input
  string l_connect = "User Id=" + txtUserName.Text + ";" +
    "Password=" + txtCurrentPassword.Text + ";" +
    "Data Source=" + txtTNSAlias.Text;

  OracleConnection oraConn = new OracleConnection(l_connect);

  try
  {
    // attempt to open a connection
    // this should fail since we have locked the account
    oraConn.Open();
  }
  catch (OracleException ex)
  {
    // trap the "account is locked" error code
    if (ex.Number == 28000)
```

```
    {
      // display a simple marker to indicate we trapped the error
      MessageBox.Show("Trapped Locked Account", "Locked Account");
    }
    else
    {
      MessageBox.Show(ex.Message, "Error Occured");
    }
  }
  finally
  {
    if (oraConn.State == System.Data.ConnectionState.Open)
    {
      oraConn.Close();
    }
  }
}
```

Figure 7-11 shows the design time representation of the form. This is the same form I used in the previous samples except I've removed the New Password and Confirm Password labels and text boxes.

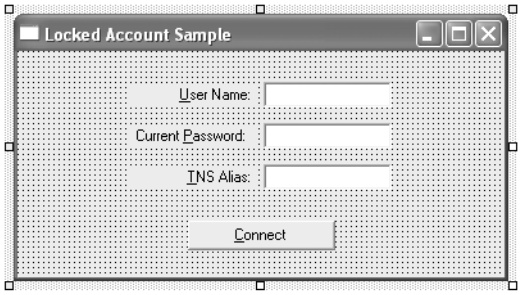

Figure 7-11. *The design time representation of the locked account form*

When you run the sample code and attempt to log in to the database as the oranetuser account as depicted in Figure 7-12, you should receive an error dialog.

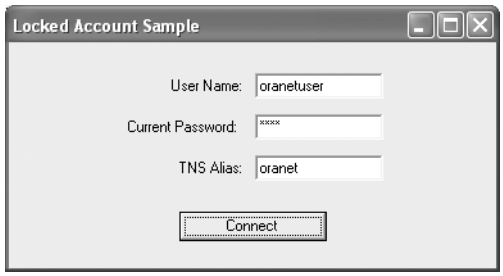

Figure 7-12. *The locked account test form at run-time*

When you click the Connect button, you're presented with the dialog in Figure 7-13.

Figure 7-13. *The dialog indicating you trapped the locked account condition*

Finally, to wrap-up your exploration of locked accounts, you'll unlock the oranetuser account in SQL*Plus using your administrative user. This is illustrated in Listing 7-32.

Listing 7-32. *Unlocking an Account*

```
C:\>sqlplus oranetadmin@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 14:09:09 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

Enter password:

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> alter user oranetuser account unlock;

User altered.

SQL> exit
Disconnected from Oracle Database 10g Enterprise Edition↵
 Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

C:\>
```

Connection Pooling and Multiplexing

One of the design features of the .NET Framework is increased scalability and more prudent resource usage. One way this design feature is exposed by the Oracle Data Provider for .NET is through the connection pooling mechanism.

■ **NOTE** The connection pooling mechanism isn't limited to the Oracle Data Provider for .NET. Other data providers can also expose the connection pooling feature.

The term *multiplexing* can be confused with the term *connection pooling*, though they represent distinct approaches to limiting resource usage. In a multiplexing configuration, a single connection to the resource (a database, in your case) is shared by multiple clients of the resource. The Oracle Data Provider for .NET doesn't expose a multiplexing method. An application (such as a middle-tier application) is responsible for creating the connection and brokering its usage among its clients. However, the database itself supports a multiplexing scheme through the shared server (formerly known as multi-threaded server) connection type. We examined shared server and dedicated server modes in Chapter 1.

On the other hand, since the connection pooling functionality is directly exposed by the data provider, and because it is trivial to implement, you use it as your resource saving method. In fact, you have to explicitly *not* use the connection pooling feature, because it is enabled by default.

In order to see how the connection pooling feature works, you'll create two applications. The first is a simple console application called NoConnectionPooling that explicitly disables connection pooling. The other, called ConnectionPooling, uses the default value of having connection pooling enabled. You'll then run these applications and examine the connections to your database in SQL*Plus. You incorporate a small pause function in your code that gives you time to examine the connections in SQL*Plus. In order to examine the effect of the connection pooling attribute, you have to do a bit of bouncing around between SQL*Plus and your application.

In order to verify that the connection pooling attribute has been disabled and to see the effects of this in SQL*Plus, create a new console application like the one in Listing 7-33.

Listing 7-33. *A Console Application That Disables Connection Pooling*

```
static void Main(string[] args)
{
  // create a connection string to our standard database
  // the pooling=false attribute disables connection pooling
  string l_connect = "User Id=oranetuser;" +
    "Password=demo;" +
    "Data Source=oranet;" +
    "pooling=false";

  OracleConnection conn_1 = new OracleConnection(l_connect);
  conn_1.Open();

  // pause so we can monitor connection in
  // SQL*Plus
  Console.WriteLine("Connection 1 created... Examine in SQL*Plus");
  Console.ReadLine();

  conn_1.Dispose();

  // pause so we can monitor connection in
  // SQL*Plus
  Console.WriteLine("Connection 1 disposed... Examine in SQL*Plus ");
  Console.ReadLine();
```

```
  OracleConnection conn_2 = new OracleConnection(l_connect);
  conn_2.Open();

  // pause so we can monitor connection in
  // SQL*Plus
  Console.WriteLine("Connection 2 created... Examine in SQL*Plus ");
  Console.ReadLine();

  conn_2.Dispose();

  // pause so we can monitor connection in
  // SQL*Plus
  Console.WriteLine("Connection 2 disposed... Examine in SQL*Plus ");
  Console.ReadLine();
}
```

Here, you repeatedly execute a query in SQL*Plus to monitor your connections. After each `Examine in SQL*Plus` message, you toggle over to the window where you're running SQL*Plus and execute the query. The query simply displays your user name, the program that is executing, and the time the user logged on to the database. The time that the user logged in to the database is important, because it is this that verifies that you are or aren't using connection pooling. If connection pooling isn't being used, the time that the user logged in to the database varies with the iterations of your query. If connection pooling is being used, the time the user logged in to the database remains constant because the single connection is being reused.

The steps you use to verify if connection pooling is or isn't being used are as follows:

1. Once you have created and compiled your console application, open a command prompt window, change to the directory that contains the executable for your test, and execute the binary, as shown here:

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\NoConnectionPooling\bin\Debug>↵
NoConnectionPooling.exe
Connection 1 created... Examine in SQL*Plus
```

2. Start a SQL*Plus session as illustrated here:

```
C:\>sqlplus oranetadmin@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Jul 12 18:18:56 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

Enter password:

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> COL PROGRAM FORMAT A24
SQL> SELECT   USERNAME,
```

```
  2           PROGRAM,
  3           TO_CHAR(LOGON_TIME,'HH24:MI:SS') LOGON_TIME
  4  FROM     V$SESSION
  5  WHERE    USERNAME = 'ORANETUSER';

USERNAME                       PROGRAM                  LOGON_TI
------------------------------ ------------------------ --------
ORANETUSER                     NoConnectionPooling.exe  18:18:56

1 row selected.
```

3. Press the Enter key to un-pause the application as illustrated in the following code:

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\NoConnectionPooling\bin\Debug>↵
NoConnectionPooling.exe
Connection 1 created... Examine in SQL*Plus

Connection 1 disposed... Examine in SQL*Plus
```

4. Toggle over to the SQL*Plus session and re-execute the query by entering a single forward slash (/) character and pressing Enter. The following code illustrates this:

```
SQL> /

no rows selected
```

Your query has returned no rows because the connection was disposed and you don't have connection pooling enabled.

5. Return to the application window and press the Enter key to un-pause the application as illustrated here:

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\NoConnectionPooling\bin\Debug>↵
NoConnectionPooling.exe
Connection 1 created... Examine in SQL*Plus

Connection 1 disposed... Examine in SQL*Plus

Connection 2 created... Examine in SQL*Plus
```

6. Return to the SQL*Plus session and re-execute the query as you did in step 4. The following code illustrates this.

```
SQL> /

USERNAME                       PROGRAM                  LOGON_TI
------------------------------ ------------------------ --------
ORANETUSER                     NoConnectionPooling.exe  18:28:58

1 row selected.
```

You can see that you've established a new connection and that this connection has a different logon time from the previous connection.

7. Return to the application window and press Enter to un-pause the application as illustrated here:

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\NoConnectionPooling\bin\Debug>
NoConnectionPooling.exe
Connection 1 created... Examine in SQL*Plus

Connection 1 disposed... Examine in SQL*Plus

Connection 2 created... Examine in SQL*Plus

Connection 2 disposed... Examine in SQL*Plus
```

8. Return to SQL*Plus and re-execute the query. The following code contains the results:

```
SQL> /

no rows selected
```

   Once again, your query doesn't return results because you disposed of your second connection.

9. Return to the application window and press Enter to un-pause the application. The application terminates at this point.

In order to demonstrate connection pooling, you'll use virtually the same code as you did in Listing 7-33. The only difference between this code and the following code to demonstrate connection pooling is that you remove the `pooling=false` attribute from the connection string. Since `pooling` is True by default, this enables your application to take advantage of connection pooling. Listing 7-34 contains the code you used in your connection pooling test application.

Listing 7-34. *A Console Application That Uses Connection Pooling*

```
static void Main(string[] args)
{
  // create a connection string to our standard database
  // the pooling attribute defaults to "true" so we
  // do not need to include it to enable pooling
  string l_connect = "User Id=oranetuser;" +
    "Password=demo;" +
    "Data Source=oranet";

  OracleConnection conn_1 = new OracleConnection(l_connect);
  conn_1.Open();

  // pause so we can monitor connection in
  // SQL*Plus
  Console.WriteLine("Connection 1 created... Examine in SQL*Plus");
  Console.ReadLine();
```

```
  conn_1.Dispose();

  // pause so we can monitor connection in
  // SQL*Plus
  Console.WriteLine("Connection 1 disposed... Examine in SQL*Plus ");
  Console.ReadLine();

  OracleConnection conn_2 = new OracleConnection(l_connect);
  conn_2.Open();

  // pause so we can monitor connection in
  // SQL*Plus
  Console.WriteLine("Connection 2 created... Examine in SQL*Plus ");
  Console.ReadLine();

  conn_2.Dispose();

  // pause so we can monitor connection in
  // SQL*Plus
  Console.WriteLine("Connection 2 disposed... Examine in SQL*Plus ");
  Console.ReadLine();
}
```

You perform the same steps as you did with the no connection pooling example. However, the results of your connection monitoring query are slightly different.

1. Once you've created and compiled your console application, open a command prompt window, change to the directory that contains the executable for your test, and execute the binary. The following code illustrates this process:

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\NoConnectionPooling\bin\Debug>↵
ConnectionPooling.exe
Connection 1 created... Hit enter key
```

2. Start a SQL*Plus session as illustrated here:

```
C:\>sqlplus oranetadmin@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Tue Jul 13 19:03:47 2004

Copyright (c) 1982, 2004, Oracle.  All rights reserved.

Enter password:

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options
```

```
SQL> COL PROGRAM FORMAT A24
SQL> SELECT   USERNAME,
  2           PROGRAM,
  3           TO_CHAR(LOGON_TIME,'HH24:MI:SS') LOGON_TIME
  4  FROM     V$SESSION
  5  WHERE    USERNAME = 'ORANETUSER';

USERNAME                       PROGRAM                  LOGON_TI
------------------------------ ------------------------ --------
ORANETUSER                     ConnectionPooling.exe    19:03:47

1 row selected.
```

3. Press the Enter key to un-pause the application as illustrated in the following code:

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\NoConnectionPooling\bin\Debug>↵
ConnectionPooling.exe
Connection 1 created... Examine in SQL*Plus

Connection 1 disposed... Examine in SQL*Plus
```

4. Toggle over to the SQL*Plus session and re-execute the query by entering a single forward slash (/) and pressing Enter as illustrated here:

```
SQL> /

USERNAME                       PROGRAM                  LOGON_TI
------------------------------ ------------------------ --------
ORANETUSER                     ConnectionPooling.exe    19:03:47

1 row selected.
```

   Your query has returned a row even though the connection was disposed of. This is the effect of connection pooling. Rather than terminate your connection, you've returned it to the pool.

5. Return to the application window and press the Enter key to un-pause the application as illustrated here:

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\NoConnectionPooling\bin\Debug>↵
ConnectionPooling.exe
Connection 1 created... Examine in SQL*Plus

Connection 1 disposed... Examine in SQL*Plus

Connection 2 created... Examine in SQL*Plus
```

6. Return to the SQL*Plus session and re-execute the query:

```
SQL> /

USERNAME                       PROGRAM                  LOGON_TI
------------------------------ ------------------------ --------
ORANETUSER                     ConnectionPooling.exe    19:03:47

1 row selected.
```

Even though your application created a new connection, you can see that you haven't created a new database connection. The logon time remains constant—the same as it was for the first connection.

7. Return to the application window and press Enter to un-pause the application:

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter07\NoConnectionPooling\bin\Debug>
ConnectionPooling.exe
Connection 1 created... Examine in SQL*Plus

Connection 1 disposed... Examine in SQL*Plus

Connection 2 created... Examine in SQL*Plus

Connection 2 disposed... Examine in SQL*Plus
```

8. Return to SQL*Plus and re-execute the query. Here are the results:

```
SQL> /

USERNAME                       PROGRAM                  LOGON_TI
------------------------------ ------------------------ --------
ORANETUSER                     ConnectionPooling.exe    19:03:47

1 row selected.
```

Once again, your query has returned the same result even though you disposed of your second connection.

9. Return to the application window and press Enter to un-pause the application.

The application terminates. If you executed your query at this point, it wouldn't return a row because the application terminated.

In the second application, you can clearly see the impact of the `pooling` attribute. By allowing connection pooling to take place, you were able to save resources because you didn't need to establish a second connection. You could reuse the existing connection, and thus bypass the overhead associated with starting up a new connection to the database. As an application developer, you have a great deal of control over the connection pooling environment. The complete set of attributes is available in the Oracle Data Provider for .NET documentation set. Connection pooling has a positive impact on web-based applications because they frequently follow a pattern of receiving the request, getting data from the database, and returning results. By maintaining a pool of readily available connections, a web-based application can reduce its service cycle time.

By storing the connection pooling attributes in the application's `web.config` (or `app.config`) file and reading them at application run-time, you can allow an application administrator to tune the connection pool without rewriting or recompiling the application. An example of this may look like the following:

```
<appSettings>
  <add key="Connection Pooling" value="false"/>
</appSettings>
```

If you used the `ConfigurationSettings.AppSettings.GetValues("Connection Pooling")` method to retrieve the value of the `"Connection Pooling"` key, the application would turn connection pooling on or off in a dynamic run-time fashion.

Chapter 7 Wrap-Up

You began this chapter by looking at the default database connection and how to implement the ability to connect to the default database in your code. We discussed how this ability can be a benefit when you don't know the TNS alias for a database in advance. After discussing a default database connection, we examined how to create a tnsnames-less connection. This type of connection affords you a great deal of flexibility, especially in a table-driven sort of application.

We then examined the difference between system-level privileged and non-privileged connections in a fair amount of detail. You created code to connect as a system-level privileged user and using SQL*Plus, you learned how these types of connections work with the database. In addition to privileged and non-privileged connections, you tackled the sometimes-confusing topic of operating system authentication. You saw how to connect to the default database and a database via the TNS alias using operating system authentication. After looking at the methods to employ authentication by the operating system, you examined a few areas that are common in environments where password rules and account lockout are in effect. And finally, you finished the chapter by taking in a fairly lengthy set of examples that illustrate how to use connection pooling. All in all, we covered a lot of ground in this chapter.