CHAPTER 5

Using PL/SQL from .NET

In a sense, this chapter represents a slight shift in focus from the previous chapters; here, you step out of the .NET environment, to a degree, and delve more deeply into the database itself. The capabilities afforded by Oracle PL/SQL are often overlooked, especially when you're using a development environment outside of the database. This chapter isn't about becoming a PL/SQL expert or dissecting advanced PL/SQL techniques; it gives you enough information to enable you to start using PL/SQL with your .NET programs. The primary focus isn't on *creating* PL/SQL, but on *using* it. It isn't possible to provide a comprehensive examination of everything PL/SQL has to offer in a single chapter—entire books are devoted to that subject. Therefore, your goal in this chapter is to understand how to effectively use PL/SQL to perform common tasks from a .NET program. Specifically, you examine the following topics:

Why use PL/SQL?: You aren't required to use PL/SQL per se, though there are benefits to using it. In this section, you learn why you may find using PL/SQL beneficial.

PL/SQL Packages: Packages and package bodies are an important aspect of PL/SQL. You get to explore how to use packages and package bodies when creating stored PL/SQL code.

Anonymous PL/SQL Blocks: It's possible to use PL/SQL code that isn't stored in the database. Using anonymous blocks is the mechanism used to accomplish this.

Parameters and Return Values: As with plain SQL statements, you can pass parameters to as well as receive return values from PL/SQL. You'll work through a sample using the `OracleParameter` class that illustrates how to do this.

The `Ref Cursor` PL/SQL Data Type: You use a `Ref Cursor` to pass a pointer to a result set on the server to a client. In this section, you develop a familiarity with what `Ref Cursors` are and how to declare them.

Returning Result Sets from PL/SQL: In this section, I use the `Ref Cursor` PL/SQL data type and the `OracleRefCursor` data provider class to illustrate how to pass a pointer that points to a server-based result set to a .NET client program.

Using PL/SQL Associative Arrays: PL/SQL Associative Arrays are arrays PL/SQL can use. In this section, you create a sample application that uses PL/SQL Associative Arrays rather than the host language arrays you used in Chapter 3.

■ **NOTE** A number of PL/SQL components are part of a database installation and are available for general use. These components are published in the "PL/SQL Packages and Types" reference manual included in the Oracle documentation set. In addition, the "PL/SQL User's Guide and Reference" is valuable. If you need to brush up on your PL/SQL skills, I recommend consulting these guides. In addition, *Mastering Oracle PL/SQL: Practical Solutions* (Apress, 2004) is an excellent resource.

Why Use PL/SQL?

This question may immediately come to mind. After all, it's certainly possible to write entire programs or systems without ever writing or calling a single line of PL/SQL code. However, as is often said, just because you *can* do it that way doesn't mean that you *should* do it that way.

PL/SQL is tightly integrated with both the Oracle database and the SQL language. For example, Oracle column types are, in general, PL/SQL data types and vice versa. This means you can work with a single variable that is of the correct data type for both the table and the language. In fact, the PL/SQL language has constructs that enable this seamlessly.

■ **NOTE** Consult the PL/SQL manuals for information on the `%type` and `%ROWtype` attributes, which automatically allow declaration of a PL/SQL variable with the correct data type based on the table structure. When you use this method of declaring variables and the underlying length or type of the column changes, this change is transparent to your PL/SQL code and requires no (or at least reduced) code rework.

To declare a variable of the same type and length as a table column, declare the variable of type *`tablename.column`* and append `%type`. For example, say you have a table named EMPLOYEES and that table has a column called `LAST_NAME` that holds an employee's last name. To declare the variable *`l_emp_last_name`* of this type, you simply declare it as *`l_emp_last_name`* `EMPLOYEES.LAST_NAME%type`. In PL/SQL, the variable type follows the variable name rather than preceding it. If you increase the length of the `LAST_NAME` column from 30 to 48 characters, this change can be transparent to your PL/SQL code that uses this table. In this case, the *`l_emp_last_name`* variable automatically increases in length to 48. You'll see an example of this as you work through the code in this chapter.

You can use PL/SQL as a security tool as well. It's possible to create a PL/SQL procedure that returns data from a database table to which the user has no direct access. Because the user has no access to the table directly, they aren't able to browse the table using an external tool. Of course, the user needs the appropriate permissions to execute the PL/SQL procedure for this scenario to work correctly, but this is easily accomplished by the administrator of the database.

As you saw when we discussed the `FetchSize` property in Chapter 2, working with data in sets rather than as individual rows has some performance benefits. PL/SQL offers an alternative way to work with data in batches rather than as discrete rows known as *associative arrays*. Using this functionality, you can perform array-based operations with PL/SQL instead of host language array operations. You learn more about this later in the chapter.

4258_Ch05_CMP3 10/5/04 7:54 AM Page 205

Although there are many reasons to use PL/SQL, as a .NET programmer, you may find that it feels somewhat unnatural to move code out of the .NET environment and into the database. However, PL/SQL was created for the purpose of working with data in an Oracle database, and it does a very good job. Recent releases of the database include enhancements to the PL/SQL compiler and optimizer, which make it even more appealing from a pure performance perspective. The following sections illustrate how using PL/SQL from the .NET environment is made simple by the data provider.

■ **TIP** When working with PL/SQL procedures, functions, packages, and so forth, you should coordinate appropriate object permissions with your administrator. Although this applies in general, of course, it is particularly important when you're working with PL/SQL code. Database roles, by default, are not active when a stored procedure is executing; this can be very confusing for you to troubleshoot if the object privileges are granted to a database role rather than directly to a user. This is fully documented in the "PL/SQL User's Guide and Reference" in the supplied documentation.

PL/SQL Packages

A *package* is a construct that PL/SQL inherited from Ada—the language upon which PL/SQL is based. In a nutshell, a PL/SQL package is a mechanism for storing or bundling related items together as a single logical entity. A package is composed of two distinct pieces:

The package specification: Defines what is contained in the package and is akin to a header file in a language such as C++. Items defined in the specification are *public*. That is, code outside of the specification and body can see these items. The *specification* is the published interface to a package.

The package body: Contains the code for the procedures and functions defined in the specification. The body may also contain code not declared in the specification; in this case, this code is *private*. Other code in the same body can see and invoke this code, but code outside of the body can't.

These two pieces are stored as separate objects in the data dictionary, and they are visible in the `user_source` view among others. The specification is stored as the `PACKAGE` type, and the body is stored as the `PACKAGE BODY` type. You see an example of this shortly.

It's possible to have a specification with no body. It isn't possible, however, to have a body with no specification. For example, you can use a specification with no body to declare a set of public constants; because a set of constants doesn't need an implementation, a body isn't necessary.

Creating a Package Specification and Body

Here is the syntax for creating a package specification:

```
create [or replace] package <package name> {is | as}
<package specification contents>
end [<package name>];
```

When you create a package specification, the optional `or replace` clause indicates to Oracle that it should replace an existing package specification if one exists. This is a shorthand method that avoids having to first drop a package specification when you're recreating it. You must specify one of the `is` or `as` clauses, but not both. I tend to use the `as` clause when I'm creating package specifications and bodies and the `is` clause when I'm creating the procedures and functions that reside in the body. I do this solely because it makes the text flow better when I'm reading it (you'll see an example of this when you create a package body). The package specification is terminated with the `end` keyword, and optionally, the package specification name may follow it.

When you're creating the package body, the syntax is virtually identical to that of the package specification. The only difference in the syntax between the two is the addition of the `body` keyword to the `create` clause. Here is the syntax for creating a package body:

```
create [or replace] package body <package name> {is | as}
<package body contents>
end [<package name>];
```

Of course, an empty specification and body aren't of much use. Therefore, I'll briefly examine the syntax you'd use to create a procedure or function within the package body. As you may imagine, the syntax to accomplish this is fairly similar to that of the specification and body:

```
procedure <procedure name> ([<procedure parameters>]) [is | as]
  <variable declarations>
begin
  <procedure body>
[exception]
  <exception block if exception keyword used>
end [procedure name];
```

In order to create a function, you use a similar syntax:

```
function <function name> ([<function parameters>]) return <return type> [is | as]
  <variable declarations>
begin
  <function body>
[exception]
  <exception block if exception keywork used>
end [function name];
```

You may notice that the `exception` block is an optional component of a procedure or function. The PL/SQL code you develop in this book doesn't make use of this block. Instead you'll let any exceptions raised inside a PL/SQL block of code propagate back to your .NET programs. For additional information on this block, see the Oracle-supplied documentation or *Mastering Oracle PL/SQL: Practical Solutions,* which I mentioned previously.

Now that you've seen the syntax you need to create a package specification, a package body, and a procedure or function, you can create a simple specification and body. In addition, you can query the data dictionary view `user_source` to see some information related to the specification and body. This simple package contains a public variable and a procedure that displays the value of that variable in the SQL*Plus command window. Listing 5-1 contains the code to create the package specification.

Listing 5-1. *Creating a Simple Package Specification*

```
C:\>sqlplus oranetuser@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Tue Aug 24 21:12:25 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, Oracle Label Security, OLAP and Data Mining options

SQL>  create or replace package simple_pkg as
  2    l_hello_text varchar2(40) := 'Hello from a simple PL/SQL package!';
  3    procedure display_message;
  4  end simple_pkg;
  5  /

Package created.
```

As you can see in Listing 5-1, this code creates a simple package specification that contains one variable and declares one procedure that will be in the package body. The code in Listing 5-2 is used to create the package body.

Listing 5-2. *Creating the Package Body*

```
SQL> create or replace package body simple_pkg as
  2    procedure display_message is
  3    begin
  4      dbms_output.put_line(l_hello_text);
  5    end display_message;
  6  end simple_pkg;
  7  /

Package body created.
```

As you can see, Listing 5-2 ends with the package body being successfully created. This procedure does nothing more than invoke an Oracle-supplied PL/SQL procedure called `put_line`, which resides in the `dbms_output` package. The *`l_hello_text`* variable is passed as a parameter to this procedure. This package is fully documented in the "PL/SQL Packages and Types" manual in the Oracle documentation set.

Now that you've created the specification and body, you can invoke the procedure from your SQL*Plus session, as illustrated in Listing 5-3. Here is the syntax you use to call a packaged procedure: *<owner>.<package>.<procedure or function name>*. The *<owner>* is optional if the user making the call owns the package, as is the case in Listing 5-3.

4258_Ch05_CMP3 10/5/04 7:54 AM Page 208

Listing 5-3: *Invoking the Simple Procedure*

```
SQL> set serveroutput on size 1000000 format word_wrapped
SQL> begin
  2    simple_pkg.display_message;
  3  end;
  4  /
Hello from a simple PL/SQL package!

PL/SQL procedure successfully completed.
```

In Listing 5-3, you first enable output in your SQL*Plus session so that you can see the output of the procedure. If you execute the procedure and only see the `PL/SQL procedure successfully completed.` text, it is likely that `serveroutput` hasn't been enabled in SQL*Plus. After enabling output, invoke your procedure and see the results of displaying the value of the *`l_hello_text`* variable.

Querying the user_source View

The `user_source` data dictionary view contains the definition of the package specification and body for the `simple_pkg` you just created. The structure of this view is presented in Listing 5-4.

Listing 5-4. *The user_source View Structure*

```
SQL> desc user_source
 Name        Null?    Type
 -------------------- --------------
 NAME                 VARCHAR2(30)
 TYPE                 VARCHAR2(12)
 LINE                 NUMBER
 TEXT                 VARCHAR2(4000)
```

You may recall that I mentioned earlier that the package specification is stored as type `PACKAGE` in the data dictionary. Listing 5-5 is where you can clearly see this. If you issue a query similar to that in Listing 5-5, you'll display the code that makes up the package specification.

Listing 5-5. *Displaying the Package Specification as Stored in the user_source View*

```
SQL> select   text
  2  from     user_source
  3  where    name = 'SIMPLE_PKG'
  4  and      type = 'PACKAGE'
  5  order by line;

TEXT
----------------------------------------------------------------------
package simple_pkg as
  l_hello_text varchar2(40) := 'Hello from a simple PL/SQL package!';
  procedure display_message;
end simple_pkg;

4 rows selected.
```

In order to see the contents of the package body, you simply need to change the type to PACKAGE BODY as illustrated in Listing 5-6.

Listing 5-6. *Displaying the Package Body as Stored in the user_source View*

```
SQL> select   text
  2  from     user_source
  3  where    name = 'SIMPLE_PKG'
  4  and      type = 'PACKAGE BODY'
  5  order by line;

TEXT
---------------------------------------
package body simple_pkg as
  procedure display_message is
  begin
    dbms_output.put_line(l_hello_text);
  end display_message;
end simple_pkg;

6 rows selected.
```

Procedure and Function Overloading

Like other languages, PL/SQL allows for procedure and function overloading. If you aren't familiar with this concept, it basically means that a package may contain multiple versions of a procedure or a function. The different versions are distinguished by the signature. This means that each version of a procedure or function must have a distinct set of parameter types.

In order to take advantage of overloading, you must use packages. You can't use overloading for stand-alone procedures or functions. The following code snippet illustrates basic overloading.

```
create package overloadtest as
  procedure tproc;
  procedure tproc(p_in_value varchar2);
end;
```

As you can see, two procedures are declared in the specification, and both are named tproc. Of course, both versions of the procedure need to be implemented in the package body. When the PL/SQL run-time engine encounters a call to tproc with no parameters, the first version of the procedure is invoked. If the PL/SQL run-time engine encounters a call to tproc with a varchar2 parameter, the second version of the procedure is invoked. You'll see overloading in action in the complete samples later in the chapter.

Anonymous PL/SQL Blocks

In Listing 5-3, you invoked your simple package procedure by wrapping the call with the begin and end keywords. In doing this, you created what is known as an *anonymous PL/SQL block*. This block is so named because it isn't a named block as your specification, body, and procedure are. Typically, an anonymous block is constructed and executed a single time. Good coding practices dictate that if a block is called multiple times, generally, it's converted to a packaged procedure or function where feasible. If, for example, you're creating an application that executes an anonymous block against a database that the user chooses at run-time, it may not be feasible to convert that block to a stored procedure or function. In addition, as is the case here, you generally use anonymous blocks to invoke stored code in the database. However, this isn't a requirement, as you'll see shortly.

Here's how to construct an anonymous block:

```
[declare]
  <variable declarations>
begin
  <block body>
end;
```

The declare keyword is optional, but you must specify it if you're going to use variable declarations. If the block uses no variables (as was the case in Listing 5-3) this keyword is omitted. You can see an anonymous block by implementing the simple_pkg as an anonymous block. Listing 5-7 illustrates this process and its output.

Listing 5-7. *Implementing the simple_pkg as an Anonymous Block*

```
SQL> set serveroutput on size 1000000 format word_wrapped
SQL> declare
  2    l_hello_text varchar2(48) := 'Hello from a simple PL/SQL↵
 anonymous block!';
  3  begin
  4    dbms_output.put_line(l_hello_text);
  5  end;
  6  /
Hello from a simple PL/SQL anonymous block!

PL/SQL procedure successfully completed.
```

Although anonymous blocks are probably most frequently used to invoke other procedures or functions, you can also use them to batch SQL statements as a single group. You can see this in Listing 5-8.

Listing 5-8. *Using an Anonymous Block to Batch Statements*

```
SQL> create table anon_test
  2  (
  3    name_id number(4) primary key,
  4    name varchar2(32)
  5  );
```

```
Table created.

SQL> begin
  2    insert into anon_test (name_id, name) values (1, 'John');
  3    insert into anon_test (name_id, name) values (2, 'Paul');
  4    insert into anon_test (name_id, name) values (3, 'George');
  5    insert into anon_test (name_id, name) values (4, 'Ringo');
  6  end;
  7  /

PL/SQL procedure successfully completed.

SQL> commit;

Commit complete.

SQL> select   name_id,
  2           name
  3  from     anon_test
  4  order by name_id;

   NAME_ID NAME
---------- ------------
         1 John
         2 Paul
         3 George
         4 Ringo

4 rows selected.
```

Here is a question that comes up frequently: "What would happen if one of the batched statements violated the primary key?" As you discovered in Chapter 3, Oracle transactions either successfully complete or fail as a whole. Therefore, if one of the batched statements violates the primary key constraint, the entire anonymous block is rolled back. You can see this in Listing 5-9.

Listing 5-9. *A Single Statement Causes the Entire Block to Be Rolled Back*

```
  1  begin
  2    insert into anon_test (name_id, name) values (5, 'Micky');
  3    insert into anon_test (name_id, name) values (6, 'Michael');
  4    insert into anon_test (name_id, name) values (7, 'Peter');
  5    insert into anon_test (name_id, name) values (1, 'David');
  6* end;
SQL> /
begin
*
ERROR at line 1:
```

```
ORA-00001: unique constraint (ORANETUSER.SYS_C006173) violated
ORA-06512: at line 5

SQL> select   name_id,
  2           name
  3  from     anon_test
  4  order by name_id;

   NAME_ID NAME
---------- --------------------------------
         1 John
         2 Paul
         3 George
         4 Ringo

4 rows selected.

SQL>
```

Even though only one of the `name_id` values in the anonymous block violated the primary key (the statement on line 5 in the anonymous block), the entire block is rolled back.

Parameters and Return Values

Back in Chapter 2, I discussed the `OracleParameter` class and the `Direction` property exposed by that class. You can use this class to pass parameters to and from your PL/SQL code. When you're working with PL/SQL, your parameters may be input only parameters, input/output parameters, output only parameters, or return values from stored functions. Of course, it's also possible to have a procedure that takes no parameters.

The different modes or directions that a parameter may take are consistent with parameters in other programming languages. You use an input parameter to pass a value to a procedure or function, and such a parameter is read-only within the body of the procedure or function. In contrast, you use an output parameter to return a value to the calling program. When you're using an output parameter, the PL/SQL code changes the value of the variable representing the parameter. The input/output parameter is a hybrid of the input and output parameters. You can use it to pass a value to a procedure; the procedure may then modify it to return a value to the calling program. The return value of a function is assigned to a parameter that you declare using the `ParameterDirection.ReturnValue` enumeration.

In order to demonstrate using PL/SQL code and the different parameter directions from .NET, I'll show you how to create a table, LEAGUE_RESULTS, using a standard user. You'll use this table for the remaining sample code in this chapter. Listing 5-10 illustrates how to create the table using SQL*Plus.

Listing 5-10. *Creating the Sample Table*

```
C:\>sqlplus oranetuser@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jun 2 11:13:50 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 table league_results
  2  (
  3    position      number(2) primary key,
  4    team          varchar2(32),
  5    played        number(2),
  6    wins          number(2),
  7    draws         number(2),
  8    losses        number(2),
  9    goals_for     number(3),
 10    goals_against number(3)
 11  );

Table created.
```

In order to illustrate how to use the different parameter directions and return a value from a function, I'll show you how to create a PL/SQL package, called `league_test`, which contains the following three procedures and one function:

`Insert_row`: Allows you to populate the sample table using input parameters.

`Retrieve_row`: Retrieves data from the table using output parameters.

`Calculate_points`: Accepts an input/output parameter to identify a particular team in your table, and then modifies the value of this parameter to reflect the number of points that the specified team has accumulated.

`get_team`: Retrieve a value from the table using a function.

Listing 5-11 shows the package specification where you declare the procedures and function and the package body that contains them. I created this code in the same SQL*Plus session as the table in Listing 5-10.

Listing 5-11. *The league_test PL/SQL Package and Package Body*

```
SQL> create or replace package league_test as
  2    procedure insert_row(p_position in number,
  3      p_team in varchar2,
  4      p_played in number,
  5      p_wins in number,
  6      p_draws in number,
  7      p_losses in number,
```

```
  8      p_goals_for in number,
  9      p_goals_against in number);
 10
 11    procedure retrieve_row(p_position in number,
 12      p_team out varchar2,
 13      p_played out number,
 14      p_wins out number,
 15      p_draws out number,
 16      p_losses out number,
 17      p_goals_for out number,
 18      p_goals_against out number);
 19
 20    procedure calculate_points(p_inout in out number);
 21
 22    function get_team(p_position in number) return varchar2;
 23  end league_test;
 24  /

Package created.

SQL> create or replace package body league_test as
  2    procedure insert_row(p_position in number,
  3      p_team in varchar2,
  4      p_played in number,
  5      p_wins in number,
  6      p_draws in number,
  7      p_losses in number,
  8      p_goals_for in number,
  9      p_goals_against in number) is
 10    begin
 11      -- insert a row into the table
 12      insert into league_results (position,
 13        team,
 14        played,
 15        wins,
 16        draws,
 17        losses,
 18        goals_for,
 19        goals_against)
 20      values (p_position,
 21        p_team,
 22        p_played,
 23        p_wins,
 24        p_draws,
 25        p_losses,
 26        p_goals_for,
 27        p_goals_against);
 28    end insert_row;
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 215


```
29
30   procedure retrieve_row(p_position in number,
31     p_team out varchar2,
32     p_played out number,
33     p_wins out number,
34     p_draws out number,
35     p_losses out number,
36     p_goals_for out number,
37     p_goals_against out number) is
38   begin
39     -- this returns the columns for a given position in the table
40     select   team,
41              played,
42              wins,
43              draws,
44              losses,
45              goals_for,
46              goals_against
47     into     p_team,
48              p_played,
49              p_wins,
50              p_draws,
51              p_losses,
52              p_goals_for,
53              p_goals_against
54     from     league_results
55     where    position = p_position;
56   end retrieve_row;
57
58   procedure calculate_points(p_inout in out number) is
59   begin
60     -- this returns the number of points for a given position
61     -- in the table
62     -- points are calculated as:
63     --   3 points for a win
64     --   1 point for a draw
65     --   0 points for a loss
66     select   (wins * 3) + (draws)
67     into     p_inout
68     from     league_results
69     where    position = p_inout;
70   end calculate_points;
71
72   function get_team(p_position in number) return varchar2 is
73     l_team league_results.team%type;
74   begin
75     -- simply get the team for a given position
76     select   team
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 216


```
 77      into     l_team
 78      from     league_results
 79      where    position = p_position;
 80
 81      return l_team;
 82    end get_team;
 83  end league_test;
 84  /

Package body created.
```

At this point, you're ready to create your .NET code that instantiates the `Main` class and invokes a series of methods to utilize the procedures and the function that you created. You'll create a console application to accomplish this, and, as with your other code, you'll use some helper methods to separate and modularize the code. This sample (Parameters) is available in this chapter's folder in the Downloads section of the Apress website (`www.apress.com`).

The `Main` procedure for your sample is presented in Listing 5-12. This procedure simply creates a connection to the database and calls your helper methods.

Listing 5-12. *The Main Procedure*

```
static void Main(string[] args)
{
  // for using our helpers
  Class1 theClass = new Class1();

  // create our standard connection
  string connStr = "User Id=oranetuser; Password=demo; Data Source=oranet";
  OracleConnection oraConn = new OracleConnection(connStr);

  oraConn.Open();

  // call the helper methods
  Console.WriteLine("Executing input parameter sample...");
  theClass.load_table(oraConn);

  Console.WriteLine("Executing ouput parameter sample...");
  theClass.retrieve_row(oraConn, 4);

  Console.WriteLine("Executing input/output parameter sample...");
  theClass.calculate_points(oraConn, 4);

  Console.WriteLine("Executing return value parameter sample...");
  theClass.get_team(oraConn, 4);

  oraConn.Close();

  oraConn.Dispose();
}
```

The load_table Method

In this sample, the code for the `load_table` method is very simple. It serves as a wrapper for calling the `do_insert` method. The code in Listing 5-13 loads the sample table with the results of the 2003–2004 English Premier League season final standings. As you can see in Listing 5-13, this code performs a series of single row inserts. You learn how to perform array operations using PL/SQL a little later in the chapter.

Listing 5-13. *The load_table Code*

```
private void load_table(OracleConnection con)
{
  insert_row(con, 1, "Arsenal", 38, 26, 12, 0, 73, 26);
  insert_row(con, 2, "Chelsea", 38, 24, 7, 7, 67, 30);
  insert_row(con, 3, "Manchester United", 38, 23, 6, 9, 64, 35);
  insert_row(con, 4, "Liverpool", 38, 16, 12, 10, 55, 37);
  insert_row(con, 5, "Newcastle United", 38, 13, 17, 8, 52, 40);
  insert_row(con, 6, "Aston Villa", 38, 15, 11, 12, 48, 44);
  insert_row(con, 7, "Charlton Athletic", 38, 14, 11, 13, 51, 51);
  insert_row(con, 8, "Bolton Wanderers", 38, 14, 11, 12, 48, 56);
  insert_row(con, 9, "Fulham", 38, 14, 10, 14, 52, 46);
  insert_row(con, 10, "Birmingham City", 38, 12, 14, 12, 43, 48);
  insert_row(con, 11, "Middlesbrough", 38, 13, 9, 16, 44, 52);
  insert_row(con, 12, "Southampton", 38, 12, 11, 15, 44, 45);
  insert_row(con, 13, "Portsmouth", 38, 12, 9, 17, 47, 54);
  insert_row(con, 14, "Tottenham Hotspur", 38, 13, 6, 19, 47, 57);
  insert_row(con, 15, "Blackburn Rovers", 38, 12, 8, 18, 51, 59);
  insert_row(con, 16, "Manchester City", 38, 9, 14, 15, 55, 54);
  insert_row(con, 17, "Everton", 38, 9, 12, 17, 45, 57);
  insert_row(con, 18, "Leicester City", 38, 6, 15, 17, 48, 65);
  insert_row(con, 19, "Leeds United", 38, 8, 9, 21, 40, 79);
  insert_row(con, 20, "Wolverhampton Wanderers", 38, 7, 12, 19, 38, 77);

  Console.WriteLine("Table successfully loaded.");
  Console.WriteLine();
}
```

The insert_row Code

The code for the `insert_row` helper method is where you really start to see how to utilize PL/SQL code from your .NET code. You create an input parameter object for each parameter and assign attributes and values as you've done in previous samples. The important differences between the code in Listing 5-14 and previous code is that you're specifying your package and stored procedure name (`league_test.insert_row`) instead of literal SQL text for the `CommandText` property, and you're specifying `CommandType.StoredProcedure` instead of `CommandType.Text`. You may also notice that my coding preference is to name my .NET methods the same as the PL/SQL procedure or function they'll invoke. This standard or coding preference isn't a requirement. Other than these two primary differences, this code is similar to what you used up to now.

Listing 5-14. *The insert_row Code*

```
private void insert_row(OracleConnection con,
  decimal position,
  string team,
  decimal played,
  decimal wins,
  decimal draws,
  decimal losses,
  decimal goals_for,
  decimal goals_against)
{
  // create parameter objects for each parameter
  OracleParameter p_position = new OracleParameter();
  OracleParameter p_team = new OracleParameter();
  OracleParameter p_played = new OracleParameter();
  OracleParameter p_wins = new OracleParameter();
  OracleParameter p_draws = new OracleParameter();
  OracleParameter p_losses = new OracleParameter();
  OracleParameter p_goals_for = new OracleParameter();
  OracleParameter p_goals_against = new OracleParameter();

  // set non-default attribute values
  p_position.OracleDbType = OracleDbType.Decimal;
  p_played.OracleDbType = OracleDbType.Decimal;
  p_wins.OracleDbType = OracleDbType.Decimal;
  p_draws.OracleDbType = OracleDbType.Decimal;
  p_losses.OracleDbType = OracleDbType.Decimal;
  p_goals_for.OracleDbType = OracleDbType.Decimal;
  p_goals_against.OracleDbType = OracleDbType.Decimal;

  // assign values
  p_position.Value = position;
  p_team.Value = team;
  p_played.Value = played;
  p_wins.Value = wins;
  p_draws.Value = draws;
  p_losses.Value = losses;
  p_goals_for.Value = goals_for;
  p_goals_against.Value = goals_against;

  // create the command object and set attributes
  OracleCommand cmd = new OracleCommand("league_test.insert_row", con);
  cmd.CommandType = CommandType.StoredProcedure;

  // add parameters to collection
  cmd.Parameters.Add(p_position);
  cmd.Parameters.Add(p_team);
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 219


```
  cmd.Parameters.Add(p_played);
  cmd.Parameters.Add(p_wins);
  cmd.Parameters.Add(p_draws);
  cmd.Parameters.Add(p_losses);
  cmd.Parameters.Add(p_goals_for);
  cmd.Parameters.Add(p_goals_against);

  // execute the command
  cmd.ExecuteNonQuery();
}
```

The retrieve_row Code

In contrast to the `insert_row` code, which uses input parameters exclusively, the `retrieve_row` code in Listing 5-15 uses output parameters for the majority of its functionality. However, the code is again similar in nature. You're creating parameters, setting attributes, executing a call to a stored procedure, and displaying the results to the console window. The comments embedded in the code indicate what task each particular section of the code is performing.

Listing 5-15. *The retrieve_row Code*

```
private void retrieve_row(OracleConnection con, decimal position)
{
  // this retrieves a row and displays to the console
  // it uses the position column to determine which row
  // to retrieve and display

  // create parameter objects for each parameter
  OracleParameter p_position = new OracleParameter();
  OracleParameter p_team = new OracleParameter();
  OracleParameter p_played = new OracleParameter();
  OracleParameter p_wins = new OracleParameter();
  OracleParameter p_draws = new OracleParameter();
  OracleParameter p_losses = new OracleParameter();
  OracleParameter p_goals_for = new OracleParameter();
  OracleParameter p_goals_against = new OracleParameter();

  // set non-default attribute values
  p_position.OracleDbType = OracleDbType.Decimal;
  p_played.OracleDbType = OracleDbType.Decimal;
  p_wins.OracleDbType = OracleDbType.Decimal;
  p_draws.OracleDbType = OracleDbType.Decimal;
  p_losses.OracleDbType = OracleDbType.Decimal;
  p_goals_for.OracleDbType = OracleDbType.Decimal;
  p_goals_against.OracleDbType = OracleDbType.Decimal;

  p_team.Direction = ParameterDirection.Output;
  p_played.Direction = ParameterDirection.Output;
```

```
  p_wins.Direction = ParameterDirection.Output;
  p_draws.Direction = ParameterDirection.Output;
  p_losses.Direction = ParameterDirection.Output;
  p_goals_for.Direction = ParameterDirection.Output;
  p_goals_against.Direction = ParameterDirection.Output;

  p_team.Size = 32;

  // assign values for input parameter
  p_position.Value = position;

  // create the command object and set attributes
  OracleCommand cmd = new OracleCommand("league_test.retrieve_row", con);
  cmd.CommandType = CommandType.StoredProcedure;

  // add parameters to collection
  cmd.Parameters.Add(p_position);
  cmd.Parameters.Add(p_team);
  cmd.Parameters.Add(p_played);
  cmd.Parameters.Add(p_wins);
  cmd.Parameters.Add(p_draws);
  cmd.Parameters.Add(p_losses);
  cmd.Parameters.Add(p_goals_for);
  cmd.Parameters.Add(p_goals_against);

  // execute the command
  cmd.ExecuteNonQuery();

  // output the row to the console window
  Console.WriteLine("     Position: " + position.ToString());
  Console.WriteLine("         Team: " + p_team.Value);
  Console.WriteLine("       Played: " + p_played.Value.ToString());
  Console.WriteLine("         Wins: " + p_wins.Value.ToString());
  Console.WriteLine("        Draws: " + p_draws.Value.ToString());
  Console.WriteLine("       Losses: " + p_losses.Value.ToString());
  Console.WriteLine("    Goals For: " + p_goals_for.Value.ToString());
  Console.WriteLine("Goals Against: " + p_goals_against.Value.ToString());
  Console.WriteLine();
}
```

The calculate_points Code

The code for the `calculate_points` method uses an input/output parameter to pass a position value to the database. The database uses that same parameter to return the total points achieved for that team. Input/output parameters are analogous to a pointer in that they allow the method to change the value of the parameter. Although I am illustrating how to use an

4258_Ch05_CMP3 10/5/04 7:54 AM Page 221

input/output parameter with a numeric value in Listing 5-16, you can also use them for text transformation. For example, you can pass a text value to a procedure, and the procedure can transform the text in some manner, such as by performing encryption or creating a checksum value. A numeric accumulator is also something that you can easily implement using an input/output parameter.

Listing 5-16. *The calculate_points Code*

```
private void calculate_points(OracleConnection con, decimal inout)
{
  // this gets the total points for a team in position inout
  // and returns the value using the inout parameter

  // create parameter object and set attributes
  OracleParameter p_inout = new OracleParameter();
  p_inout.OracleDbType = OracleDbType.Decimal;
  p_inout.Direction = ParameterDirection.InputOutput;
  p_inout.Value = inout;

  // create the command object and set attributes
  OracleCommand cmd = new OracleCommand("league_test.calculate_points", con);
  cmd.CommandType = CommandType.StoredProcedure;

  // add parameter to the collection
  cmd.Parameters.Add(p_inout);

  // execute the command
  cmd.ExecuteNonQuery();

  // output the result to the console window
  Console.WriteLine("Total Points for position {0}: {1}",
    inout.ToString(), p_inout.Value.ToString());

  Console.WriteLine();
}
```

The get_team Code

The code in Listing 5-17 utilizes the `get_team` function and illustrates how to use a return value from a function. Using a return value from a function is not especially different from using an output parameter in a procedure. In many cases, it's a matter of semantics or established coding standards. Both accomplish the same task. Because you're retrieving a single value and are using a single input parameter, the code is shorter than that in the previous examples.

4258_Ch05_CMP3 10/5/04 7:54 AM Page 222

Listing 5-17. *The get_team Code*

```
private void get_team(OracleConnection con, decimal position)
{
  // gets the name of the team in position

  // create parameter objects and set attributes
  OracleParameter p_position = new OracleParameter();
  p_position.OracleDbType = OracleDbType.Decimal;
  p_position.Value = position;

  OracleParameter p_retval = new OracleParameter();
  p_retval.Direction = ParameterDirection.ReturnValue;
  p_retval.Size = 32;

  // create the command object and set attributes
  OracleCommand cmd = new OracleCommand("league_test.get_team", con);
  cmd.CommandType = CommandType.StoredProcedure;

  // add parameters to the collection
  cmd.Parameters.Add(p_retval);
  cmd.Parameters.Add(p_position);

  // execute the command
  cmd.ExecuteNonQuery();

  // output the result to the console window
  Console.WriteLine("Team in position {0}: {1}",
    position.ToString(), p_retval.Value.ToString());
}
```

Running the Parameters Sample Application

The sample application doesn't provide any user interactivity, but you can clearly see the results of running the application in a console window. Listing 5-18 illustrates the results of running the application.

Listing 5-18. *The Sample Application Output*

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter05\Parameters\bin\Debug>⏎
Parameters.exe
Executing input parameter sample...
Table successfully loaded.

Executing ouput parameter sample...
     Position: 4
         Team: Liverpool
```

```
        Played: 38
          Wins: 16
         Draws: 12
        Losses: 10
     Goals For: 55
 Goals Against: 37

Executing input/output parameter sample...
Total Points for position 4: 60

Executing return value parameter sample...
Team in position 4: Liverpool

C:\My Projects\ProOraNet\Oracle\C#\Chapter05\Parameters\bin\Debug>
```

In the output in Listing 5-19, you can clearly see the results of your procedures and function that return values from the database. However, you don't see the full results of the input parameter code that loads the table. In this listing, I've slightly reformatted the output to prevent line wrapping.

Listing 5-19. *The league_results Table*

```
SQL> select * from league_results order by position;

 POS TEAM                      PLAYED    WINS DRAWS LOSSES   FOR   AGAINST
---- ------------------------ ------- ------- ----- ------ ----- ---------
   1 Arsenal                       38      26    12      0    73        26
   2 Chelsea                       38      24     7      7    67        30
   3 Manchester United             38      23     6      9    64        35
   4 Liverpool                     38      16    12     10    55        37
   5 Newcastle United              38      13    17      8    52        40
   6 Aston Villa                   38      15    11     12    48        44
   7 Charlton Athletic             38      14    11     13    51        51
   8 Bolton Wanderers              38      14    11     12    48        56
   9 Fulham                        38      14    10     14    52        46
  10 Birmingham City               38      12    14     12    43        48
  11 Middlesbrough                 38      13     9     16    44        52
  12 Southampton                   38      12    11     15    44        45
  13 Portsmouth                    38      12     9     17    47        54
  14 Tottenham Hotspur             38      13     6     19    47        57
  15 Blackburn Rovers              38      12     8     18    51        59
  16 Manchester City               38       9    14     15    55        54
  17 Everton                       38       9    12     17    45        57
  18 Leicester City                38       6    15     17    48        65
  19 Leeds United                  38       8     9     21    40        79
  20 Wolverhampton Wanderers       38       7    12     19    38        77

20 rows selected.
SQL>
```

The Ref Cursor PL/SQL Data Type

One of the primary purposes of the Ref Cursor data type is that it allows PL/SQL to return a result set to a program written in another language. You can also use Ref Cursors within PL/SQL; however, in this section, you'll examine them from the perspective of returning results to an external program. By *external*, I mean a program you wrote in a language other than PL/SQL—one that doesn't reside in the database.

Although I typically refer to "returning a result set," a key attribute of using a Ref Cursor is that the result set actually remains on the server. What is returned to the client is a pointer to the result set on the server. This has an important ramification; when you're using Ref Cursors, the cursor is only valid for an open database connection—if the connection is closed, the Ref Cursor is no longer valid. This makes sense because the Ref Cursor is a pointer to a result set on the server; with no underlying connection, the pointer is invalid. Once you close a Ref Cursor, the resources held on the server are also released.

■ **NOTE** The OracleRefCursor class is used with Ref Cursors in the Oracle data provider. You can access the Ref Cursors via the OracleDataReader class in the Microsoft data provider.

Declaring a Ref Cursor Variable in PL/SQL

Before you can return a Ref Cursor to a calling program, you must declare a variable of the appropriate type. Unfortunately, you can't directly declare a variable to be a Ref Cursor. Instead, you must create a user-defined type and declare your variable to be of that type. Although this sounds confusing, it's really quite simple. The following code snippet illustrates this process:

```
-- create the type
type ref_cursor is ref cursor;
-- create the variable of type ref_cursor
l_cursor ref_cursor;
```

A Ref Cursor can be of two types:

Weakly typed: No return type is specified when the type is defined.

Strongly typed: A return type is specified when the type is defined.

The preceding code creates a weakly typed cursor type since no return type was defined. In order to create a strongly typed cursor type, you'll need to use code similar to the following:

```
-- create the type
type ref_cursor is ref cursor return league_results%rowtype;
-- create the variable of type ref_cursor
l_cursor ref_cursor;
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 225

In this code, a strongly typed cursor type is created. This `Ref Cursor` can only return a result set that has the structure of the LEAGUE_RESULTS table. The primary advantage of a weakly typed cursor is that you can use it to return any type of result set since no return type was defined. On the other hand, the PL/SQL compiler can check that a strongly typed cursor is associated only with queries that return the correct columns and column types.

■ **NOTE** If you know the type that a cursor needs to return, use a strongly typed cursor. If you don't, use a weakly typed cursor. For example, if the cursor returns data from a table whose structure you know at design time, you can use a strongly typed cursor. If the cursor returns data from a table that a user chooses dynamically at run-time, use a weakly typed cursor.

Returning Result Sets from PL/SQL

Earlier in the chapter, you returned values from the database using PL/SQL output, input/output, and function return value parameters. This approach works well when you're dealing with single row or single value results. However, when you're returning multiple rows, a better approach is to return a set of data rather than using individual parameters and values.

As you've just seen, the `Ref Cursor` is designed for this purpose. Recall the sample code from Chapter 2 where I illustrated the results of single-row fetching versus multiple-row fetching. This is the same principle you employ when you're using PL/SQL to return results from the database. Generally, you can return (or fetch) data in sets more successfully than implementing the same activity using single-row operations. This doesn't mean, however, that the techniques you utilized in the previous section no longer apply when you're returning sets of data. The techniques that you used to return values as parameters are still valid when you're returning sets of data—you still return the data as an output parameter or as a function return value. In this case, the output parameter or return value is a `Ref Cursor` rather than a scalar value.

You acquire the `OracleRefCursor` object as an output parameter or a function return value of type `OracleDbType.RefCursor`. There are no constructors for the `OracleRefCursor` class. Once you've acquired a PL/SQL `Ref Cursor` as an `OracleRefCursor`, you can populate an `OracleDataReader` or a `DataSet` object in your code. Although you can update data in a `DataSet`, a `Ref Cursor` itself isn't updateable. Therefore, if you choose to implement the capability to update data you retrieve via a `Ref Cursor`, you must provide a custom SQL statement for the `OracleDataAdapter` object. I address returning data rather than updating data from a `Ref Cursor` in this section.

Because you've already seen how to fill a `DataGrid` using a `DataSet` in Chapter 2, here, you'll create a console application that displays, as a comma-separated list, the contents of the LEAGUE_RESULTS table that you created and populated in the previous section.

Once you've acquired the `Ref Cursor` and populated an `OracleDataReader`, you'll see that it is no different working with the data than what you do if you use a SQL statement in your .NET code. However, because the code that returns your set of data resides in the database, it receives all the benefits that I mentioned earlier in the chapter, that is, it is compiled and available for the database server to reuse it.

To return the Ref Cursor from the database, you create a new package and package body as illustrated in Listing 5-20. You can also return a Ref Cursor using an anonymous block. The technique is the same as I illustrate here. The only difference is that the code resides in an anonymous block instead of a package. Here, you're using a get_table procedure and get_table function to illustrate the two different means of returning your cursor. The get_table routine is overloaded, and when the package code executes, the PL/SQL run-time determines (based on the different signatures of the two routines) the appropriate routine to invoke. I discussed overloading earlier in the chapter.

Listing 5-20. *Creating the league_rc PL/SQL Code*

```
C:\>sqlplus oranetuser@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Aug 25 11:50:46 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, Oracle Label Security, OLAP and Data Mining options

SQL> create or replace package league_rc as
  2    type ref_cursor is ref cursor return league_results%rowtype;
  3
  4    function get_table return ref_cursor;
  5    procedure get_table (p_cursor out ref_cursor);
  6  end league_rc;
  7  /

Package created.

SQL> create or replace package body league_rc as
  2    function get_table return ref_cursor is
  3      l_cursor ref_cursor;
  4    begin
  5      open l_cursor for
  6      select   position,
  7               team,
  8               played,
  9               wins,
 10               draws,
 11               losses,
 12               goals_for,
 13               goals_against
 14      from     league_results
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 227


```
15     order by position;
16
17     return l_cursor;
18   end get_table;
19
20   procedure get_table (p_cursor out ref_cursor) is
21   begin
22     open p_cursor for
23     select   position,
24              team,
25              played,
26              wins,
27              draws,
28              losses,
29              goals_for,
30              goals_against
31     from     league_results
32     order by position;
33   end get_table;
34 end league_rc;
35 /

Package body created.
```

In the PL/SQL code, you can see that you're declaring `Ref_Cursor` as a strongly typed `Ref Cursor` type. This is how you return the cursor to your .NET client code. In each routine, the cursor opens using a basic `SELECT` statement that retrieves and orders all of the rows in the LEAGUE_RESULTS table. This is an area where PL/SQL excels. Due to the tight integration of SQL and PL/SQL, you're able to embed SQL statements in your PL/SQL code with no modifications.

The `Main` method for your console application, which is the RefCursor solution in this chapter's folder in the Downloads section of the Apress website (`www.apress.com`), is presented in Listing 5-21. As you can see, it is similar to the previous code you've examined.

Listing 5-21. *The Main Method Code*

```
static void Main(string[] args)
{
  // for using our helpers
  Class1 theClass = new Class1();

  // create our standard connection
  string connStr = "User Id=oranetuser; Password=demo; Data Source=oranet";
  OracleConnection oraConn = new OracleConnection(connStr);

  oraConn.Open();

  // call the helper methods
```

```
  Console.WriteLine("Invoking ref cursor function...");
  theClass.call_function(oraConn);

  Console.WriteLine("Invoking ref cursor procedure...");
  theClass.call_procedure(oraConn);

  oraConn.Close();

  oraConn.Dispose();
}
```

The call_function Code

The `call_function` helper method in Listing 5-22 invokes the `get_table` function rather than the procedure in your PL/SQL package. Because you've declared the parameter direction to be a return value (`ParameterDirection.ReturnValue`) for your parameter, the code the data provider sent to the database causes the PL/SQL engine to invoke your function rather than your procedure. (Refer to the discussion of the `Direction` property in Chapter 2 if you need a refresher on this property.)

As I indicate with the code comments, it's important to assign the `OracleDbType` property correctly. By moving the code that deals with the data directly into the database, you simplify the .NET code you need.

Listing 5-22. *The call_function Code*

```
private void call_function(OracleConnection con)
{
  // create the command object and set attributes
  OracleCommand cmd = new OracleCommand("league_rc.get_table", con);
  cmd.CommandType = CommandType.StoredProcedure;

  // create parameter object for the cursor
  OracleParameter p_refcursor = new OracleParameter();

  // this is vital to set when using ref cursors
  p_refcursor.OracleDbType = OracleDbType.RefCursor;
  p_refcursor.Direction = ParameterDirection.ReturnValue;

  cmd.Parameters.Add(p_refcursor);

  OracleDataReader reader = cmd.ExecuteReader();

  while (reader.Read())
  {
    Console.Write(reader.GetDecimal(0).ToString() + ",");
    Console.Write(reader.GetString(1) + ",");
    Console.Write(reader.GetDecimal(2).ToString() + ",");
    Console.Write(reader.GetDecimal(3).ToString() + ",");
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 229


```
    Console.Write(reader.GetDecimal(4).ToString() + ",");
    Console.Write(reader.GetDecimal(5).ToString() + ",");
    Console.Write(reader.GetDecimal(6).ToString() + ",");
    Console.WriteLine(reader.GetDecimal(7).ToString());
  }

  Console.WriteLine();

  reader.Close();
  reader.Dispose();
  p_refcursor.Dispose();
  cmd.Dispose();
}
```

The call_procedure Code

The `call_procedure` code, as presented in Listing 5-23, is very similar to the code for the `call_function` method. You shouldn't be surprised by this. Because you elected to use an overloaded PL/SQL routine in the database, the primary difference between the two methods is that the parameter direction is specified differently. In order to correctly invoke your procedure in the database, you must declare the parameter direction to be an output parameter rather than a return value as you did in the `call_function` code. As with the function call code, you must correctly assign the `OracleDbType` as a `RefCursor`.

Listing 5-23. *The call_procedure Code*

```
private void call_procedure(OracleConnection con)
{
  // create the command object and set attributes
  OracleCommand cmd = new OracleCommand("league_rc.get_table", con);
  cmd.CommandType = CommandType.StoredProcedure;

  // create parameter object for the cursor
  OracleParameter p_refcursor = new OracleParameter();

  // this is vital to set when using ref cursors
  p_refcursor.OracleDbType = OracleDbType.RefCursor;
  p_refcursor.Direction = ParameterDirection.Output;

  cmd.Parameters.Add(p_refcursor);

  OracleDataReader reader = cmd.ExecuteReader();

  while (reader.Read())
  {
    Console.Write(reader.GetDecimal(0).ToString() + ",");
    Console.Write(reader.GetString(1) + ",");
    Console.Write(reader.GetDecimal(2).ToString() + ",");
```

```
    Console.Write(reader.GetDecimal(3).ToString() + ",");
    Console.Write(reader.GetDecimal(4).ToString() + ",");
    Console.Write(reader.GetDecimal(5).ToString() + ",");
    Console.Write(reader.GetDecimal(6).ToString() + ",");
    Console.WriteLine(reader.GetDecimal(7).ToString());
  }

  Console.WriteLine();

  reader.Close();
  reader.Dispose();
  p_refcursor.Dispose();
  cmd.Dispose();
}
```

Running the RefCursor Sample Application

When you run the sample application, it creates two sets of output to the console window. The output is the same for each method, as illustrated in Listing 5-24. As with the parameters sample from the previous section, whether you choose to use a procedure or a function often comes down to semantics. As you discovered in the `call_procedure` and `call_function` code, using either mechanism is very similar.

Listing 5-24. *The RefCursor Application Output*

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter05\RefCursor\bin\Debug>RefCursor.exe
Invoking ref cursor function...
1,Arsenal,38,26,12,0,73,26
2,Chelsea,38,24,7,7,67,30
3,Manchester United,38,23,6,9,64,35
4,Liverpool,38,16,12,10,55,37
5,Newcastle United,38,13,17,8,52,40
6,Aston Villa,38,15,11,12,48,44
7,Charlton Athletic,38,14,11,13,51,51
8,Bolton Wanderers,38,14,11,12,48,56
9,Fulham,38,14,10,14,52,46
10,Birmingham City,38,12,14,12,43,48
11,Middlesbrough,38,13,9,16,44,52
12,Southampton,38,12,11,15,44,45
13,Portsmouth,38,12,9,17,47,54
14,Tottenham Hotspur,38,13,6,19,47,57
15,Blackburn Rovers,38,12,8,18,51,59
16,Manchester City,38,9,14,15,55,54
17,Everton,38,9,12,17,45,57
18,Leicester City,38,6,15,17,48,65
19,Leeds United,38,8,9,21,40,79
20,Wolverhampton Wanderers,38,7,12,19,38,77
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 231


```
Invoking ref cursor procedure...
1,Arsenal,38,26,12,0,73,26
2,Chelsea,38,24,7,7,67,30
3,Manchester United,38,23,6,9,64,35
[duplicate output snipped]
20,Wolverhampton Wanderers,38,7,12,19,38,77

C:\My Projects\ProOraNet\Oracle\C#\Chapter05\RefCursor\bin\Debug>
```

Performing Bulk Operations

Earlier in the chapter, you used a routine in a PL/SQL package to insert rows into the LEAGUE_RESULTS table. That procedure inserted a single row at a time and was called repeatedly for each row that you inserted. As you saw in Chapters 2 and 3, performing operations in bulk can be a more efficient manner. The Oracle Data Provider for .NET allows you to use array binding to accomplish this when you're using arrays in .NET code. You can also use PL/SQL arrays, which are known as PL/SQL Associative Arrays. On major advantage of using these arrays, is that you can pass arrays to and from your PL/SQL code stored in the database, which means you can retrieve data in arrays, which you weren't able to do when using host language arrays.

■ **NOTE** The PL/SQL Associative Array feature isn't available in the current version of the Microsoft data provider.

PL/SQL Associative Arrays

A PL/SQL Associative Array is also known as a *PL/SQL Index-By Table*. The reason for this is that when you create a type to represent the array, you use the keywords `table` and `index by`. The syntax to declare such a type looks like this:

```
type <type name> is table of <data type> index by binary_integer
```

This creates an in-memory structure that is a set of key and value pairs. Specifying the `index by binary_integer` clause means that the key is an integer value. The value associated with each integer key is of type `<data type>`, which you specified when you created the type. For instance, to create an associative array that has integer key values and `varchar2` data values with a maximum length of 32, you use a statement such as the following:

■ **NOTE** In version 9i and later, you can also specify `varchar2` as the `index by` instead of `binary_integer`. This is documented in the PL/SQL User's Guide and Reference shipped with the Oracle documentation.

```
type t_assoc_array is table of varchar2(32) index by binary_integer;
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 232

Rather than specifying specific data types, you can also use the %type keyword we've discussed previously. This is illustrated by the following:

```
type t_assoc_array is table of league_results.team%type index by binary_integer;
```

The type created in here is of the same data type as the team column in the LEAGUE_RESULTS table.

You can access individual key/value pairs in the array by using a subscript operator and specifying the index value. For example, using the t_assoc_array type and an anonymous block, you can create a variable of this type and access individual elements as illustrated in Listing 5-25.

■**NOTE** Two dashes (--) are used to indicate a single line comment in PL/SQL. You can also use the C language convention of /* and */ for multiline comments.

Listing 5-25. *Creating an Associative Array Variable and Accessing Elements*

```
SQL> declare
  2    -- create the type
  3    type t_assoc_array is table of varchar2(32) index by binary_integer;
  4    -- create a variable of that type
  5    l_assoc_variable t_assoc_array;
  6    -- another variable to assign the value
  7    l_temp varchar2(32);
  8  begin
  9    -- add an element to the array
 10    l_assoc_variable(1) := 'An associative array value';
 11    -- get the element from the array
 12    l_temp := l_assoc_variable(1);
 13    -- display the value
 14    dbms_output.put_line(l_temp);
 15  end;
 16  /
An associative array value

PL/SQL procedure successfully completed.
```

In order to determine the index value for the first or last element in an associative array, you can use the first and last properties. These supply the equivalent of the lower and upper bounds of a .NET array. Listing 5-26 illustrates accessing these properties.

Listing 5-26. *Accessing the first and last Properties of an Associative Array*

```
SQL> declare
  2    -- create the type
  3    type t_assoc_array is table of varchar2(32) index by binary_integer;
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 233


```
  4    -- create a variable of that type
  5    l_assoc_variable t_assoc_array;
  6  begin
  7    -- add some elements to the array
  8    l_assoc_variable(1) := 'Element 1';
  9    l_assoc_variable(2) := 'Element 2';
 10    l_assoc_variable(3) := 'Element 3';
 11    l_assoc_variable(4) := 'Element 4';
 12    -- display the lower bound
 13    dbms_output.put_line(l_assoc_variable.first);
 14    -- display the upper bound
 15    dbms_output.put_line(l_assoc_variable.last);
 16  end;
 17  /
1
4

PL/SQL procedure successfully completed.
```

Using Bulk Binding

Using the `first` and `last` properties of an associative array does allow you to iterate over the array using a `for` loop. However, this isn't a very efficient method of performing this activity. Fortunately, PL/SQL allows you to perform operations on the array in bulk. Using bulk binding, you can populate an entire array in a single operation. You can also access all elements in an array in a single operation. To perform this operation when you're using an `UPDATE`, `INSERT`, or `DELETE` statement, you use the `forall` keyword. To perform this when doing a `SELECT`, specify the `bulk collect` keywords. Remember that these bulk operations take place on a PL/SQL Associative Array and not inside a plain SQL statement. You must use an anonymous block, a procedure, or a function to accomplish this task.

In order to use bulk binding, you must, of course, first create an associative array. The general syntax for using bulk binding with an `UPDATE`, `INSERT`, or `DELETE` statement is as follows:

```
forall <indexer> in <array.first>..<array.last>
  <update, insert, or delete statement> <array(<indexer>)>;
```

If you use the `t_assoc_array` and the `l_assoc_variable` from the previous section, an insert operation using bulk binding into a table named `t` would resemble the following code snippet:

```
forall x in l_assoc_variable.first..l_assoc_variable.last
  insert into t values (l_assoc_variable(x));
```

Using bulk binding with a select operation works in a similar manner. Again if you use the same associative array and variable, a bulk select would be similar to the following:

```
select   column
bulk collect into l_assoc_variable
from     t
order by t;
```

The Associative Array Bulk Sample Application

When you're working with PL/SQL arrays, you perform the same basic operations that you did when you were using host language arrays. Your sample console application (Associative in this chapter's folder of the Downloads section of the Apress website) ties together the concepts of PL/SQL Associative Arrays and bulk binding. This sample follows the pattern that you've developed of creating parameter objects, setting attribute values, adding the parameters to the command object parameter collection, and finally executing the code associated with the command object. In contrast to the code in Chapter 3, this code invokes stored procedures in the database to insert and select the sample data rather than utilize SQL statements embedded in the .NET code.

Listing 5-27 illustrates how to create the package and package body that contain the following two stored procedures that you use in this sample:

`bulk_insert`: This procedure performs a bulk insert (using an associative array) of data in the LEAGUE_RESULTS table.

`bulk_select`: This procedure performs a bulk select (using an associative array) of data from the LEAGUE_RESULTS table.

In the package declaration, you create a type for each column in the database table. This serves as your array inside of the PL/SQL code. You marshal data between the database and your .NET code using `OracleParameter` objects and these PL/SQL arrays. The PL/SQL procedures themselves are quite simple. They simply insert all of the values in the passed-in array into the LEAGUE_RESULTS table, or they return all of the data in the table to your client code.

Listing 5-27. *Creating the PL/SQL Package and Package Body*

```
C:\>sqlplus oranetuser@oranet

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Aug 25 12:57:04 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, Oracle Label Security, OLAP and Data Mining options

SQL> create or replace package league_associative as
  2    -- create a type for each column
  3    type t_position is table of league_results.position%type
  4      index by binary_integer;
  5    type t_team is table of league_results.team%type
  6      index by binary_integer;
  7    type t_played is table of league_results.played%type
  8      index by binary_integer;
  9    type t_wins is table of league_results.wins%type
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 235


```
 10      index by binary_integer;
 11    type t_draws is table of league_results.draws%type
 12      index by binary_integer;
 13    type t_losses is table of league_results.losses%type
 14      index by binary_integer;
 15    type t_goals_for is table of league_results.goals_for%type
 16      index by binary_integer;
 17    type t_goals_against is table of league_results.goals_against%type
 18      index by binary_integer;
 19
 20    -- the procedures that will perform our work
 21    procedure bulk_insert (p_position in t_position,
 22                           p_team in t_team,
 23                           p_played in t_played,
 24                           p_wins in t_wins,
 25                           p_draws in t_draws,
 26                           p_losses in t_losses,
 27                           p_goals_for in t_goals_for,
 28                           p_goals_against in t_goals_against);
 29
 30    procedure bulk_select (p_position out t_position,
 31                           p_team out t_team,
 32                           p_played out t_played,
 33                           p_wins out t_wins,
 34                           p_draws out t_draws,
 35                           p_losses out t_losses,
 36                           p_goals_for out t_goals_for,
 37                           p_goals_against out t_goals_against);
 38  end league_associative;
 39  /

Package created.

SQL>
SQL> create or replace package body league_associative as
  2    procedure bulk_insert (p_position in t_position,
  3                           p_team in t_team,
  4                           p_played in t_played,
  5                           p_wins in t_wins,
  6                           p_draws in t_draws,
  7                           p_losses in t_losses,
  8                           p_goals_for in t_goals_for,
  9                           p_goals_against in t_goals_against) is
 10    begin
 11      forall i in p_position.first..p_position.last
 12      insert into league_results (position,
 13                                  team,
 14                                  played,
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 236


```
 15                               wins,
 16                               draws,
 17                               losses,
 18                               goals_for,
 19                               goals_against)
 20                       values (p_position(i),
 21                               p_team(i),
 22                               p_played(i),
 23                               p_wins(i),
 24                               p_draws(i),
 25                               p_losses(i),
 26                               p_goals_for(i),
 27                               p_goals_against(i));
 28    end bulk_insert;
 29
 30    procedure bulk_select (p_position out t_position,
 31                           p_team out t_team,
 32                           p_played out t_played,
 33                           p_wins out t_wins,
 34                           p_draws out t_draws,
 35                           p_losses out t_losses,
 36                           p_goals_for out t_goals_for,
 37                           p_goals_against out t_goals_against) is
 38    begin
 39      select   position,
 40               team,
 41               played,
 42               wins,
 43               draws,
 44               losses,
 45               goals_for,
 46               goals_against
 47      bulk collect into p_position,
 48                        p_team,
 49                        p_played,
 50                        p_wins,
 51                        p_draws,
 52                        p_losses,
 53                        p_goals_for,
 54                        p_goals_against
 55      from     league_results
 56      order by position;
 57    end bulk_select;
 58  end league_associative;
 59  /

Package body created.

SQL>
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 237

Now that you have successfully created your PL/SQL code, you can create the familiar looking `Main` method as illustrated in Listing 5-28. Here, you'll use an addition to the source code file that you haven't seen before; you include `using Oracle.DataAccess.Types;` at the top of the source code file. Although I've included this namespace in the template applications in the code download, this is the first time you've expressly needed to use it. You need to determine what type of value your parameter represents in the code that displays the data in the console window, and including the `Types` enumeration in your source file makes this easier.

Listing 5-28. *The Main Method Code*

```
static void Main(string[] args)
{
  // for using our helpers
  Class1 theClass = new Class1();

  // create our standard connection
  string connStr = "User Id=oranetuser; Password=demo; Data Source=oranet";
  OracleConnection oraConn = new OracleConnection(connStr);

  oraConn.Open();

  // call the helper methods
  Console.WriteLine("Truncating table...");
  theClass.truncate_table(oraConn);
  Console.WriteLine("Completed truncating table...");
  Console.WriteLine();

  Console.WriteLine("Executing associative insert...");
  theClass.associative_insert(oraConn);
  Console.WriteLine("Completed associative insert...");
  Console.WriteLine();

  Console.WriteLine("Executing associative select...");
  theClass.associative_select(oraConn);
  Console.WriteLine("Completed associative select...");

  oraConn.Close();

  oraConn.Dispose();
}
```

The truncate_table Helper Method Code

Because you'll load the exact same data that already resides in the LEAGUE_RESULTS table shortly, you need to remove the existing data first. The `truncate_table` helper method in Listing 5-29 accomplishes this task. If you didn't remove the data, you'd get a constraint violation for the primary key column `position`.

Listing 5-29. *The truncate_table Method Code*

```
private void truncate_table(OracleConnection con)
{
  // a very simple helper method to truncate the
  // league_results table
  // since we will be inserting data that would
  // violate the primary key otherwise

  // create the command object and set attributes
  OracleCommand cmd = new OracleCommand("truncate table league_results", con);

  cmd.ExecuteNonQuery();
}
```

The associative_insert Method Code

The `associative_insert` method in Listing 5-30 is responsible for setting up all the parameter objects and invoking the PL/SQL procedure that you created earlier. Because you aren't using default values for most of the attributes in this code, the code may seem more verbose than the code you've seen up to now. However, the code repeats the same basic pattern that the other code you've created follows.

In the same way that you must set the `OracleDbType` correctly for code that uses a `Ref Cursor`, you must set the `CollectionType` property to `PLSQLAssociativeArray` for your code to function properly. Once you've set the attributes for your objects, assign values to the parameter objects. You're using the same data that you used earlier to populate the LEAGUE_RESULTS table. However, in this code, you're populating the table in a single database call and round-trip.

Listing 5-30. *The associative_insert Method Code*

```
private void associative_insert(OracleConnection con)
{
  // create the command object and set attributes
  OracleCommand cmd = new OracleCommand("league_associative.bulk_insert", con);
  cmd.CommandType = CommandType.StoredProcedure;

  // create parameter objects for each parameter
  OracleParameter p_position = new OracleParameter();
  OracleParameter p_team = new OracleParameter();
  OracleParameter p_played = new OracleParameter();
  OracleParameter p_wins = new OracleParameter();
  OracleParameter p_draws = new OracleParameter();
  OracleParameter p_losses = new OracleParameter();
  OracleParameter p_goals_for = new OracleParameter();
  OracleParameter p_goals_against = new OracleParameter();

  // set parameter type for each parameter
  p_position.OracleDbType = OracleDbType.Decimal;
```

```
p_team.OracleDbType = OracleDbType.Varchar2;
p_played.OracleDbType = OracleDbType.Decimal;
p_wins.OracleDbType = OracleDbType.Decimal;
p_draws.OracleDbType = OracleDbType.Decimal;
p_losses.OracleDbType = OracleDbType.Decimal;
p_goals_for.OracleDbType = OracleDbType.Decimal;
p_goals_against.OracleDbType = OracleDbType.Decimal;

// set the collection type for each parameter
p_position.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
p_team.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
p_played.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
p_wins.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
p_draws.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
p_losses.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
p_goals_for.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
p_goals_against.CollectionType = OracleCollectionType.PLSQLAssociativeArray;

// set the parameter values
p_position.Value = new decimal[20]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                                   11, 12, 13, 14, 15, 16, 17, 18, 19, 20};

p_team.Value = new string[20]{"Arsenal", "Chelsea",
                              "Manchester United", "Liverpool",
                              "Newcastle United", "Aston Villa",
                              "Charlton Athletic", "Bolton Wanderers",
                              "Fulham", "Birmingham City",
                              "Middlesbrough", "Southampton",
                              "Portsmouth", "Tottenham Hotspur",
                              "Blackburn Rovers", "Manchester City",
                              "Everton", "Leicester City",
                              "Leeds United", "Wolverhampton Wanderers"};

p_played.Value = new decimal[20]{38, 38, 38, 38, 38, 38, 38, 38, 38, 38,
                                 38, 38, 38, 38, 38, 38, 38, 38, 38, 38};

p_wins.Value = new decimal[20]{26, 24, 23, 16, 13, 15, 14, 14, 14, 12, 13,
                               12, 12, 13, 12, 9, 9, 6, 8, 7};

p_draws.Value = new decimal[20]{12, 7, 6, 12, 17, 11, 11, 11, 10, 14, 9,
                                11, 9, 6, 8, 14, 12, 15, 9, 12};

p_losses.Value = new decimal[20]{0, 7, 9, 10, 8, 12, 13, 12, 14, 12,
                                 16, 15, 17, 19, 18, 15, 17, 17, 21, 19};

p_goals_for.Value = new decimal[20]{73, 67, 64, 55, 52, 48, 51, 48, 52, 43,
                                    44, 44, 47, 47, 51, 55, 45, 48, 40, 38};
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 240


```
  p_goals_against.Value = new decimal[20]{26, 30, 35, 37, 40, 44, 51, 56, 46, 48,
                                          52, 45, 54, 57, 59, 54, 57, 65, 79,
                                          77};

  // set the size for each array
  p_position.Size = 20;
  p_team.Size = 20;
  p_played.Size = 20;
  p_wins.Size = 20;
  p_draws.Size = 20;
  p_losses.Size = 20;
  p_goals_for.Size = 20;
  p_goals_against.Size = 20;

  // set array bind size for the team column since it
  // is a variable size type (varchar2)
  p_team.ArrayBindSize = new int[20]{32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                                     32, 32, 32, 32, 32, 32, 32, 32, 32, 32};

  // add the parameters to the command object
  cmd.Parameters.Add(p_position);
  cmd.Parameters.Add(p_team);
  cmd.Parameters.Add(p_played);
  cmd.Parameters.Add(p_wins);
  cmd.Parameters.Add(p_draws);
  cmd.Parameters.Add(p_losses);
  cmd.Parameters.Add(p_goals_for);
  cmd.Parameters.Add(p_goals_against);

  // execute the insert
  cmd.ExecuteNonQuery();
}
```

The associative_select Method Code

Once you've inserted the data into the table, select it back out and display it to the console window. The `associative_select` method code performs these tasks for you. The code in Listing 5-31 is shorter than the code for the `associative_insert` method since you aren't required to establish parameter values. The important distinctions between this code and the code for the `associative_insert` method are that you declare the parameters as output parameters and that you assign the array values initially as `null`.

After performing the requisite setting of attribute values and so forth, you invoke the stored procedure that returns a PL/SQL Associative Array to your .NET code. The .NET code involved in this operation isn't materially different from the code that employed host language array binding.

In the loop that displays the parameter values to the console, you can see the reason for including `using Oracle.DataAccess.Types;` at the beginning of the source file. Your code

4258_Ch05_CMP3 10/5/04 7:54 AM Page 241

needs to determine what type of value a parameter object represents so that it may be output to the console window correctly. By including the namespace, your code is slightly more compact. The loop determines the number of iterations your code must make to retrieve the `size` attribute from the `p_position` parameter. All the parameters have the same number of elements. Once the number of overall iterations is determined, you iterate the parameter collection to display the parameter values for each row returned.

Listing 5-31. *The associative_select Method Code*

```
private void associative_select(OracleConnection con)
{
  // create the command object and set attributes
  OracleCommand cmd = new OracleCommand("league_associative.bulk_select", con);
  cmd.CommandType = CommandType.StoredProcedure;

  // create parameter objects for each parameter
  OracleParameter p_position = new OracleParameter();
  OracleParameter p_team = new OracleParameter();
  OracleParameter p_played = new OracleParameter();
  OracleParameter p_wins = new OracleParameter();
  OracleParameter p_draws = new OracleParameter();
  OracleParameter p_losses = new OracleParameter();
  OracleParameter p_goals_for = new OracleParameter();
  OracleParameter p_goals_against = new OracleParameter();

  // set parameter type for each parameter
  p_position.OracleDbType = OracleDbType.Decimal;
  p_team.OracleDbType = OracleDbType.Varchar2;
  p_played.OracleDbType = OracleDbType.Decimal;
  p_wins.OracleDbType = OracleDbType.Decimal;
  p_draws.OracleDbType = OracleDbType.Decimal;
  p_losses.OracleDbType = OracleDbType.Decimal;
  p_goals_for.OracleDbType = OracleDbType.Decimal;
  p_goals_against.OracleDbType = OracleDbType.Decimal;

  // set the collection type for each parameter
  p_position.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
  p_team.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
  p_played.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
  p_wins.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
  p_draws.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
  p_losses.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
  p_goals_for.CollectionType = OracleCollectionType.PLSQLAssociativeArray;
  p_goals_against.CollectionType = OracleCollectionType.PLSQLAssociativeArray;

  // set the direct for each parameter
  p_position.Direction = ParameterDirection.Output;
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 242


```
p_team.Direction = ParameterDirection.Output;
p_played.Direction = ParameterDirection.Output;
p_wins.Direction = ParameterDirection.Output;
p_draws.Direction = ParameterDirection.Output;
p_losses.Direction = ParameterDirection.Output;
p_goals_for.Direction = ParameterDirection.Output;
p_goals_against.Direction = ParameterDirection.Output;

// set the parameter values to null initially
p_position.Value = null;
p_team.Value = null;
p_played.Value = null;
p_wins.Value = null;
p_draws.Value = null;
p_losses.Value = null;
p_goals_for.Value = null;
p_goals_against.Value = null;

// set the size for each array
p_position.Size = 20;
p_team.Size = 20;
p_played.Size = 20;
p_wins.Size = 20;
p_draws.Size = 20;
p_losses.Size = 20;
p_goals_for.Size = 20;
p_goals_against.Size = 20;

// set array bind size for the team column since it
// is a variable size type (varchar2)
p_team.ArrayBindSize = new int[20]{32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32};

// add the parameters to the command object
cmd.Parameters.Add(p_position);
cmd.Parameters.Add(p_team);
cmd.Parameters.Add(p_played);
cmd.Parameters.Add(p_wins);
cmd.Parameters.Add(p_draws);
cmd.Parameters.Add(p_losses);
cmd.Parameters.Add(p_goals_for);
cmd.Parameters.Add(p_goals_against);

// execute the insert
cmd.ExecuteNonQuery();

// display as comma separated list
// field_count is used to determine when
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 243


```
  // we have completed a "line"
  int field_count = 1;
  for (int i = 0; i < p_position.Size; i++)
  {
    foreach (OracleParameter p in cmd.Parameters)
    {
      if (p.Value is OracleDecimal[])
      {
        Console.Write((p.Value as OracleDecimal[])[i]);
      }
      if (p.Value is OracleString[])
      {
        Console.Write((p.Value as OracleString[])[i]);
      }

      if (field_count < 8)
      {
        Console.Write(",");
      }
      else
      {
        field_count = 0;
      }

      field_count++;
    }
    Console.WriteLine();
  }
}
```

Running the Associative Sample

Like the previous samples, the Associative sample doesn't require any user input. You place a few lines of code to indicate where you are in the execution of the code. You can see these informational messages in the output of the sample in Listing 5-32. The program output is the same as the output from the `Ref Cursor` sample as you might expect—after all, you used the same data for both samples.

Listing 5-32. *The Associative Sample Output*

```
C:\My Projects\ProOraNet\Oracle\C#\Chapter05\Associative\bin\Debug>⏎
Associative.exe
Truncating table...
Completed truncating table...

Executing associative insert...
Completed associative insert...
```

4258_Ch05_CMP3 10/5/04 7:54 AM Page 244


```
Executing associative select...
1,Arsenal,38,26,12,0,73,26
2,Chelsea,38,24,7,7,67,30
3,Manchester United,38,23,6,9,64,35
4,Liverpool,38,16,12,10,55,37
5,Newcastle United,38,13,17,8,52,40
6,Aston Villa,38,15,11,12,48,44
7,Charlton Athletic,38,14,11,13,51,51
8,Bolton Wanderers,38,14,11,12,48,56
9,Fulham,38,14,10,14,52,46
10,Birmingham City,38,12,14,12,43,48
11,Middlesbrough,38,13,9,16,44,52
12,Southampton,38,12,11,15,44,45
13,Portsmouth,38,12,9,17,47,54
14,Tottenham Hotspur,38,13,6,19,47,57
15,Blackburn Rovers,38,12,8,18,51,59
16,Manchester City,38,9,14,15,55,54
17,Everton,38,9,12,17,45,57
18,Leicester City,38,6,15,17,48,65
19,Leeds United,38,8,9,21,40,79
20,Wolverhampton Wanderers,38,7,12,19,38,77
Completed associative select...

C:\My Projects\ProOraNet\Oracle\C#\Chapter05\Associative\bin\Debug>
```

Chapter 5 Wrap-Up

As I indicated at the beginning of the chapter, this chapter isn't about becoming a PL/SQL expert. However, I designed it to illustrate a topic that is sometimes (or often?) overlooked, yet one that provides a programming technique that is no more difficult than embedding SQL directly into your client code. I showed you some areas that you'll most often encounter when you use PL/SQL in conjunction with .NET code. Specifically I addressed the following:

Reasons for using PL/SQL: Even though you aren't required to use PL/SQL, it has its benefits. We examined these.

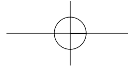
Using PL/SQL packages: Packages and package bodies are an important aspect of PL/SQL. You saw how to use these when you're creating stored PL/SQL code.

Working with anonymous PL/SQL blocks: You learned how to use anonymous blocks to invoke code stored in the database and to batch SQL statements.

Applying parameters and return values: You learned to pass parameters to and receive return values from PL/SQL. You built a sample using the `OracleParameter` class that illustrates how to do this.

Employing the `Ref Cursor` PL/SQL data type: You learned that using this data type is a way to pass a pointer to a result set on the server to a client. In this section, you became familiar with and learned to understand `Ref Cursors` and how you declare them.

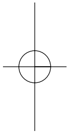
4258_Ch05_CMP3 10/5/04 7:54 AM Page 245

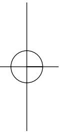


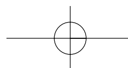
Returning result sets from PL/SQL: You learned how to apply your newfound knowledge of the Ref Cursor PL/SQL data type and the `OracleRefCursor` data provider class to pass a pointer to a server-based result set to a .NET client program.

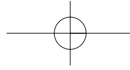
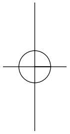
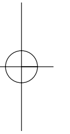
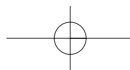
Using PL/SQL Associative Arrays: You saw how to create a sample application that uses PL/SQL Associative Arrays rather than using host language arrays. You also became familiar with performing bulk binding operations.

As you saw in Chapter 4, pragmatism and context sense are important aspects of a development effort. Using PL/SQL appropriately can serve to separate the logic and processes that deal directly with the data in a database from the logic and processes that deal with client-side issues. In addition to this separation of duties, the PL/SQL compiler compiles and optimizes code stored in PL/SQL packages once rather than possibly submitting and optimizing it multiple times, as may happen with embedded code that resides in the client. As you'll see in Chapter 6, sometimes using a supplied PL/SQL package is the easiest way to accomplish a task.







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 shortcut 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_OraDatabase101` 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.

4258_Ch07_CMP2 10/4/04 8:28 PM Page 287

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;
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 289


```
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.

4258_Ch07_CMP2 10/4/04 8:28 PM Page 290

■ **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
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 292


```
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.

4258_Ch07_CMP2 10/4/04 8:28 PM Page 293

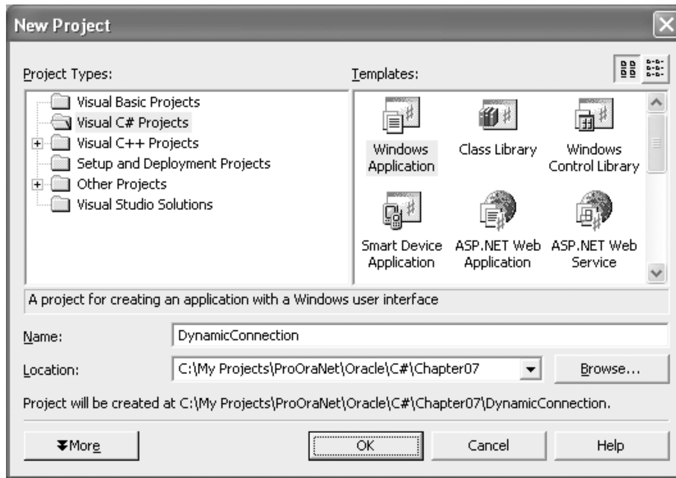


Figure 7-1. *The New Project dialog for the tnsnames-less sample*

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.

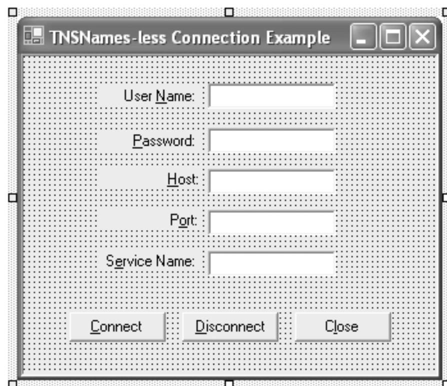


Figure 7-2. *The main form used in the tnsnames-less sample*

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.

4258_Ch07_CMP2 10/4/04 8:28 PM Page 294

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

4258_Ch07_CMP2 10/4/04 8:28 PM Page 296

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
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 297


```
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↵
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 298


```
 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.
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 300


```
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
              *
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 301


```
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.

4258_Ch07_CMP2 10/4/04 8:28 PM Page 305

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. *An 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 an 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;
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 307


```
User created.

SQL> grant connect to "OPS$RIDMRWILLIM1801\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 connection 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
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 308


```
------------------------------- ------------------------
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 connection 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 connection 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 (see 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();
    }
  }
}
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 310

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.

4258_Ch07_CMP2 10/4/04 8:28 PM Page 311

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.

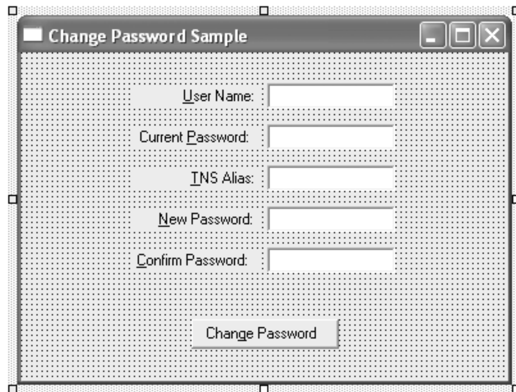


Figure 7-5. *The design-time representation of the form*

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

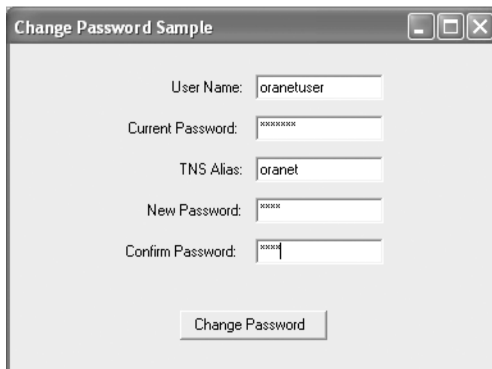


Figure 7-6. *The run-time representation of the form*

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

Figure 7-7. *The confirmation dialog*

4258_Ch07_CMP2 10/4/04 8:28 PM Page 314

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;
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 315


```
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.

4258_Ch07_CMP2 10/4/04 8:28 PM Page 316

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);
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 317


```
      // 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.

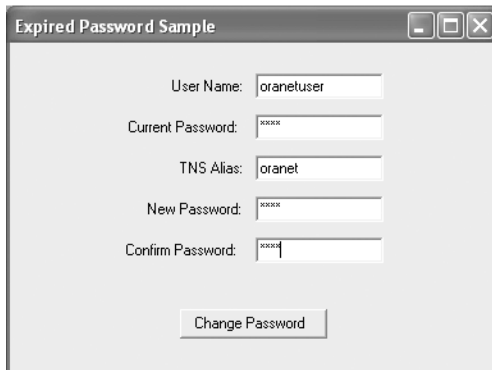


Figure 7-8. *The Expired Password Sample form*

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.

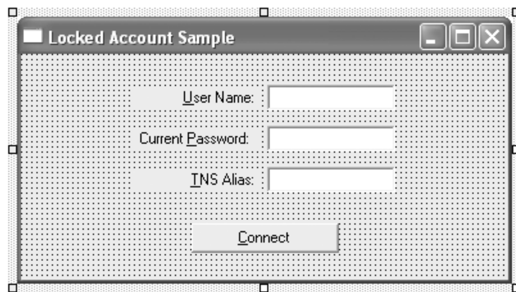


Figure 7-11. *The design time representation of the locked account form*

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.

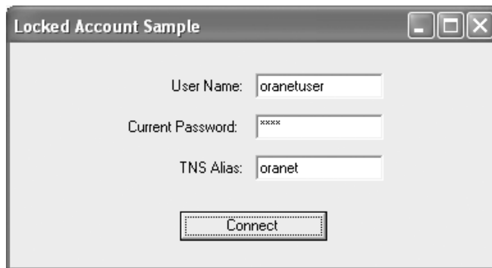


Figure 7-12. *The locked account test form at run-time*

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.

4258_Ch07_CMP2 10/4/04 8:28 PM Page 323

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();
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 324


```
  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 in 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 an 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,
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 325


```
  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.

4258_Ch07_CMP2 10/4/04 8:28 PM Page 326

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();
}
```

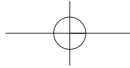
You perform the same steps as you did with the no connection pooling example. However, the results of your connection monitoring query are slightly different.

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 an 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:

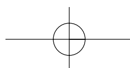
```
SQL> /

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
```

4258_Ch07_CMP2 10/4/04 8:28 PM Page 329



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:

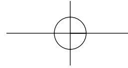
```
SQL> /

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.

4258_Ch07_CMP2 10/4/04 8:28 PM Page 330



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.