Returning an ADO Recordset to Visual Basic/ASP using REF CURSORS

Author: Alastair Vance
Created: 02-JUN-2000
Contact: **********

This example was created and tested using the following:

- Oracle 8.1.5 Database
- Oracle 8.1.6 Client (which contains the necessary 8.1.6 OLE DB Provider drivers)
- Demo Tables (see create and insert scripts below)
- Microsoft ADO 2.1 or greater
- Microsoft Visual Basic 6 SP3

For this example, I will use the following table structure:

```
CREATE TABLE site (
        site_id       number(2)      not null,
        location       varchar2(12))
/
```

Insert the following rows into the site table:

```
insert into site values (1, 'PARIS');
insert into site values (2, 'BOSTON');
insert into site values (3, 'LONDON');
insert into site values (4, 'STOCKHOLM');
insert into site values (5, 'OTTAWA');
insert into site values (6, 'WASHINGTON');
insert into site values (7, 'LA');
insert into site values (8, 'TORONTO');
```

Once the table has been created and populated, we can start to develop our oracle package/procedures to query the table. First, the package header…

```
CREATE OR REPLACE PACKAGE AL_PACKAGE AS

         TYPE mycursor IS REF CURSOR;

            PROCEDURE AL_PROCEDURE1 (
                        Pmyid IN NUMBER,
                        Pmycursor OUT mycursor, -- Use cursor
                        Perrorcode OUT NUMBER);

            PROCEDURE AL_PROCEDURE2 (
                         Pmyquery IN VARCHAR2,
                         Pmycursor OUT mycursor,
                         Perrorcode OUT NUMBER);

END AL_PACKAGE;
/
```

Type the above script into your editor and execute it to create the package header. In the package header, I have declared my REF CURSOR as mycursor (see above). There are two procedures declared within the package header, *AL_PROCEDURE1* and *AL_PROCEDURE2*.

AL_PROCEDURE1 accepts a number parameter as input and returns a variable of type mycursor (which is our REF CURSOR) and a number parameter (which will contain any SQL error codes).

The *AL_PROCEDURE2* declaration is similar to *AL_PROCEDURE1*, but accepts a string parameter as input in place of a number.

```
CREATE OR REPLACE PACKAGE BODY AL_PACKAGE AS


        PROCEDURE AL_PROCEDURE1 (
                        Pmyid IN NUMBER,
                        Pmycursor OUT mycursor, -- Use cursor
                        Perrorcode OUT NUMBER) IS

        BEGIN

                Perrorcode := 0;

                -- Open the REF CURSOR
                -- Use Input Variable "PmyID" as part of the query.
                OPEN Pmycursor FOR
                        SELECT location
                             FROM Site
                             WHERE site_id = Pmyid;

        EXCEPTION
                WHEN OTHERS THEN
                        Perrorcode := SQLCODE;


        END AL_PROCEDURE1;


        PROCEDURE AL_PROCEDURE2 (
                         Pmyquery IN VARCHAR2,
                         Pmycursor OUT mycursor,
                         Perrorcode OUT NUMBER) IS

           BEGIN

                       Perrorcode := 0;

                       -- Open the REF CURSOR
                       -- This procedure uses a query
                       -- which is passed in as a parameter.
                       OPEN Pmycursor FOR Pmyquery;


           EXCEPTION
                        WHEN OTHERS THEN
                               Perrorcode := SQLCODE;



        END AL_PROCEDURE2;



END AL_PACKAGE;
/
```

As you can see from above, *AL_PROCEDURE1* simply runs a pre-defined query, using a REF CURSOR. It uses the input variable to filter the query. In this example, the procedure expects an input parameter containing the *site_id*. The REF CURSOR (when opened) will contain the record with the corresponding site *location*.

AL_PROCEDURE2 is a more powerful procedure. It allows us to pass ANY valid SQL statement to be executed. The REF CURSOR will contain the results.

In both procedures, if there are any exceptions raised by Oracle, the SQL Error Code is passed back to the caller (Visual Basic) using the declared OUT parameter, *Perrorcode*.

We will now look at how we can use Visual Basic to call our procedures and read the records stored in the REF CURSOR.

- Start up VB and create a new project.
- Add in "Microsoft ActiveX Data Objects 2.1 Library" from the References dialog within the Project menu
- Add a text field to the form (name it "txtQuery")
- Add a command button to the form (name it "CmdExecute")
- Add a text box to the form (name it "txtResults")

We are going to use *AL_PROCEDURE1* first. Type the following code into your project under the click event for the button.

```
Private Sub CmdExecute_Click()
Dim Conn As New ADODB.Connection
Dim RS As New ADODB.Recordset
Dim Cmd As New ADODB.Command

TxtResults.Text = ""

‘ It’s important to use the correct connection string.  Especially the PLSQLRSet switch!
‘ Important:  I am using Oracle OLE DB Provider
‘ Remember to replace xxxxx with your database username/password, etc.
Conn.Open "PROVIDER=OraOLEDB.Oracle;DATA SOURCE=xxxxx;” & _
“USER ID=xxxxxx;PASSWORD=xxxxxx;PLSQLRSet=1"

Cmd.ActiveConnection = Conn
Cmd.CommandType = adCmdStoredProc
Cmd.CommandText = "AL_PACKAGE.AL_PROCEDURE1"

' Setup parameters.  Notice how we don't need one for the REF CURSOR.
' Also, notice how we pass the parameter in from the text field.
Cmd.Parameters.Append Cmd.CreateParameter("SiteID", adVarChar, adParamInput, 10,
TxtQuery.Text)
Cmd.Parameters.Append Cmd.CreateParameter("ErrCode", adVarChar, adParamOutput, 10)

' Execute the Procedure and populate our recordset
Set RS = Cmd.Execute

' A generic do while loop which will display
' all records contained in our recordset
Do While Not RS.EOF

   For i = 0 To RS.Fields.Count - 1

      TxtResults.Text = TxtResults.Text & RS.Fields(i).Value & Chr(9)

   Next
   TxtResults.Text = TxtResults.Text & vbCrLf
```

```
   RS.MoveNext

Loop

' Print error message if any
TxtResults.Text = TxtResults.Text & vbCrLf & "Error Code: " & Cmd.Parameters("ErrCode").Value
& vbCrLf

RS.Close
Set Cmd = Nothing
Conn.Close
Set Conn = Nothing
End Sub
```

Save and run the project. If you type 8 into the query text box and click the button, you should get returned "TORONTO". Try a few different *site_id*'s to see what happens (refer to the site table).

The VB code for executing *AL_PROCEDURE2* is virtually identical, apart from two lines.

Simply comment out the following line (which specifies which procedure to execute – the format is PACKAGENAME.PROCEDURENAME):

```
Cmd.CommandText = "AL_PACKAGE.AL_PROCEDURE1"
```

And insert the following line:

```
Cmd.CommandText = "AL_PACKAGE.AL_PROCEDURE2"
```

Also, comment out the following create parameter line:

```
Cmd.Parameters.Append Cmd.CreateParameter("SiteID", adVarChar, adParamInput, 10,
TxtQuery.Text)
```

And insert the following line in its place:

```
Cmd.Parameters.Append Cmd.CreateParameter("QueryStr", adVarChar, adParamInput, 2000,
TxtQuery.Text)
```

That's all there is to it! Resave the project and hit run. This time, type a valid SQL query (do not append a semi-colon to the end of the statement) into the query text box and click the button. You should see the results of your query in the results text box!

Here are some examples to try:

- Select count(*) from site
- Select * from site
- Select * from site where location like '%ON'
- Try querying another table that may be available in the same schema.