White Paper

Retrieving Result Sets from Oracle® Stored Procedures using Team Developer

By Suren Behari
Product Manager

November 2004

GUPTA

Table of Contents

Abstract

Let's start by saying this: "It is possible to retrieve Record Sets from an Oracle® stored procedure using SQLRouter/Oracle AND OLE DB from Team Developer."

With SQL Server and Sybase it is relatively straightforward to return a record set from a stored procedure. Oracle, on the other hand, does not allow the results of a SELECT statement to be passed straight back to the calling client. Such a construct can only be used with an INTO clause. This rule applies to both Anonymous PL/SQL blocks (a stored procedure without a name) and stored procedures.

This paper covers two samples in detail, showing you how to retrieve Oracle Stored Procedure result sets using both the SQLRouter/Oracle and the OLE DB Provider for Oracle.

Oracle Packages

A package is a way of grouping collections of objects together - such as datatypes, stored procedures, functions, variables, and constants.

Structure of an Oracle Package

For those not familiar with Oracle packages, I will just go through the basics, although the only essential part to remember is that the TYPE declaration for each PL/SQL table must appear in the *specification* section of the package.

A package consists of a specification (a.k.a. header) and a body section.

The specification section is the definition of all elements that can be referenced outside the package, i.e. the public elements. This is analogous to a COM type *library*. This is where the TYPE PL/SQL table declaration must be placed.

The body section then defines the actual code that makes up the objects within the package, i.e. the implementation of the stored procedures.

Package Specification Syntax

The Specification is the definition of all elements that can be referenced outside the package, i.e. the *public* elements. This is similar to a *type library* in COM. This is where the TYPE declaration will be placed.

```
PACKAGE package_name
IS
  {variable and type declarations}
  {cursor declarations}
  [module specifications]
END {package_name};
```

Package BODY Syntax

The Body section defines the code making up the objects within the package, i.e. the actual implementation of the stored procedures.

```
PACKAGE BODY package_name
IS
  {variable and type declarations}
  {cursor specifications - SELECT statements}
  [module specifications]
BEGIN
  [procedure bodies]
END {package_name};
```

Using SQLRouter/Oracle

SQLRouter/Oracle does not support Oracle Reference Cursors, however, you can retrieve Oracle result sets from Stored Procedures by using arrays. You can use dynamic arrays as input, output, and input/output arguments to Oracle PL/SQL stored procedures. The arrays can be of type NUMBER, STRING and DATE/TIME.

PL/SQL Tables

A PL/SQL table is an indexed, single-dimension, unbounded, sparse collection of homogenous elements. What? In mere mortal terms, it is a datatype similar to arrays as found in Team Developer and other programming languages.

It defines a structured (non-scalar) datatype capable of storing a single array of values. For those familiar with Team Developer, a stored procedure that uses PL/SQL table parameters can be thought of as a Team Developer function with array parameters that are passed as Receive type. These array parameters are populated one-by-one before being passed back to the calling function.

Each column that is to be returned by the stored procedure must be declared as a separate parameter to the procedure. However, it is perfectly legal to use the same PL/SQL table datatype for each parameter if that is applicable, such as in the case of a ***firstname*** and ***lastname*** field.

Note: Oracle8 also supports two new collection datatypes that can be used as parameters to a stored procedure: *Variable Arrays (VARRAYs)* and *Nested Tables*. These will not be covered here as PL/SQL tables were available from PL/SQL version 2 and suit our purposes ideally.

Overview of PL/SQL Tables

PL/SQL tables have the following characteristics:

- One Dimensional
  PL/SQL tables can contain only one column.

- Integer Indexed
  Each element is indexed by a single integer. Note that they do not have to be sequential; therefore you could use an existing foreign key to index the data such as an employee ID. At present Oracle only supports the use of BINARY_INTEGER as the indexing mode - which is fine for our purposes.

- Unbounded
  PL/SQL tables have no preset limit: as you add an element the table will grow to accommodate it.

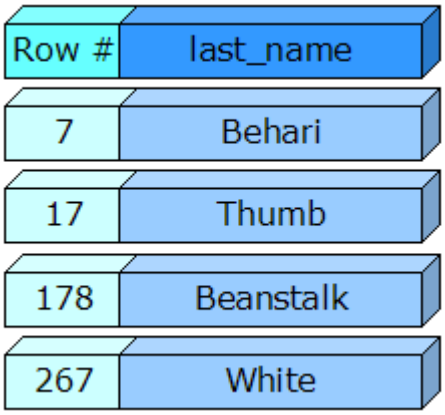
- Consistent Data Types
  Each and every element must be of the same datatype.

- Sparsely Populated
  The rows in a PL/SQL table do not need to be assigned in a sequential order, i.e. row five could contain the word 'blue' whereas row 505 could contain the word 'red' with no rows defined in between.

To define a PL/SQL table, the TYPE statement must be placed in the Specification section of an Oracle package. This can then be used as a parameter datatype for any stored procedures in that package.

Example

We may want to retrieve a list of last names from an employee table. To do this we must declare a PL/SQL table to contain all names.



Thus:

TYPE tblEmployeeName IS TABLE OF VARCHAR2(30) INDEX BY BINARY_ INTEGER;

This will define a structure capable of storing an array of VARCHAR2 values each to a length of 30 characters. With this statement there is no binding to a specific column. So, if the first and last names are declared as VARCHAR2(30), tblEmployeeName can be used to hold arrays of first and last names.

Sample Application

Step 1: Connecting
The sample application starts by prompted for Oracle connection information. I used the instance name "**TDORA9i**" as my data source, with username and password as "**scott**" and "**tiger**" respectively,

Step 2: Create Table

```
CREATE TABLE employee
        (
        em_id           NUMBER(3)   PRIMARY KEY,
        em_first_name       VARCHAR2(30),
        em_last_name        VARCHAR2(30)
        )'
```

Step 3: Create the Package Header

```
CREATE OR REPLACE PACKAGE Employee_pkg AS

TYPE em_id_tbl IS TABLE OF employee.em_id%TYPE INDEX BY BINARY_
INTEGER;
TYPE first_name_tbl IS TABLE OF employee.em_first_name%TYPE INDEX BY
BINARY_INTEGER;
TYPE last_name_tbl IS TABLE OF employee.em_last_name%TYPE INDEX BY
BINARY_INTEGER;


PROCEDURE insert_employee
        (em_id              IN      em_id_tbl,
         em_first_name      IN      first_name_tbl,
         em_last_name       IN      last_name_tbl,
         em_num_details     IN      NUMBER);

PROCEDURE RecordCount(nRecCount OUT NUMBER);

PROCEDURE GetEmployeeList(o_emID OUT em_id_tbl, o_FirstName OUT first_
name_tbl,
                          o_LastName OUT last_name_tbl);

END Employee_Pkg;
```

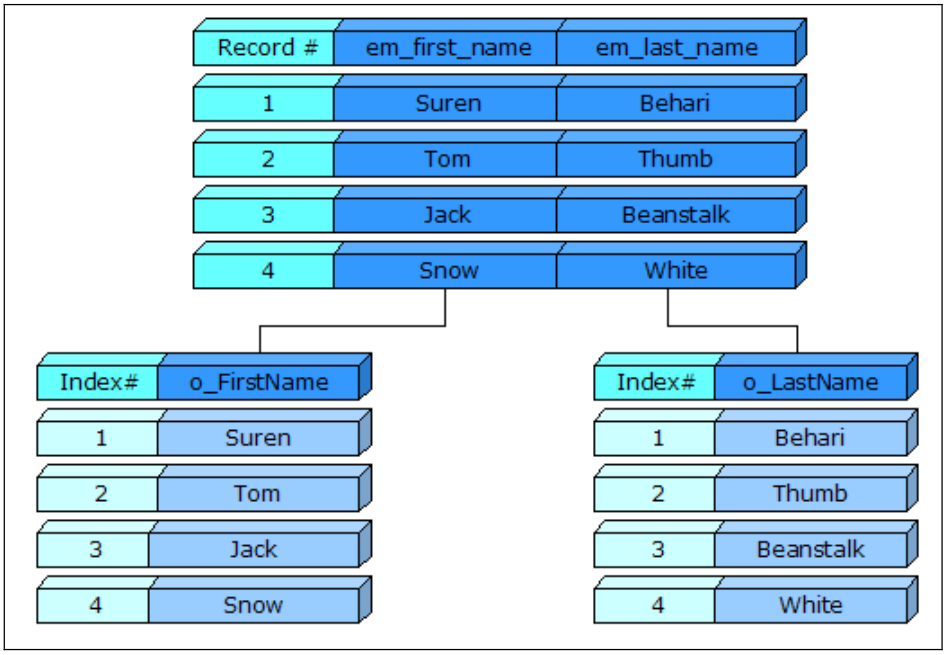
Step 4: Create the Package Body
Once you have defined your stored procedure's specification you can define the implementation in the body section.

The easiest and most efficient way to populate each of your parameter elements is to use a PL/SQL cursor For...Loop to loop through the results of your cursor SELECT transferring them into the corresponding positions; for each record found increment a record counter and transfer the first_name into o_FirstName(x)

and last_name into o_LastName(x).

```
/*************************************************
 * Open the cursor and populate each element for each record
 *************************************************/
FOR EmployeeRec IN employee_cur LOOP
    recCount:= recCount + 1;
    o_emID(recCount):= EmployeeRec.em_id;
    o_FirstName(recCount):= EmployeeRec.em_first_name;
    o_LastName(recCount):=  EmployeeRec.em_last_name;
END LOOP;
```

Example



For example, you may want to retrieve a list of employee first and last names for certain criteria from a table (em_first_name and em_last_name). The stored procedure would need two output parameters, e.g. o_FirstName and o_ LastName. Both parameters must be declared as PL/SQL table datatypes.

Once populated, there were 50 records that matched the criteria. You would find that both parameters had an upper boundary of 50 and the results of each record would appear in successive elements.

```
CREATE OR REPLACE PACKAGE BODY Employee_pkg AS

        PROCEDURE insert_employee
                (em_id                  IN      em_id_tbl,
                em_first_name           IN      first_name_tbl,
                em_last_name            IN      last_name_tbl,
                em_num_details          IN      NUMBER)
        IS
        BEGIN
                FOR n IN 1..em_num_details LOOP

                INSERT INTO EMPLOYEE
                        (em_id,em_first_name,em_last_name) VALUES
                        (em_id(n),em_first_name(n),em_last_name(n));
                END LOOP;
        END insert_employee;

        PROCEDURE RecordCount(nRecCount OUT NUMBER) IS
        BEGIN
                nRecCount :=0;
                SELECT count(*) into nRecCount FROM employee;
        END RecordCount;

        PROCEDURE GetEmployeeList(o_emID OUT em_id_tbl, o_
FirstName       OUT first_name_tbl,
                                        o_LastName OUT last_name_tbl)  IS
            CURSOR employee_cur IS
                SELECT em_id, em_first_name, em_last_name FROM
employee;
                recCount NUMBER DEFAULT 0;
                BEGIN
                FOR EmployeeRec IN employee_cur LOOP
                        recCount:= recCount + 1;
                        o_emID(recCount):= EmployeeRec.em_id;
                        o_FirstName(recCount):= EmployeeRec.em_first_
name;
                        o_LastName(recCount):=  EmployeeRec.em_last_
name;
                END LOOP;
        END GetEmployeeList;

END Employee_Pkg;'
```

Step 5: Insert Rows using Arrays

The sample application uses arrays as input arguments to the ***insert_employee*** procedure, to insert rows into the employee table, as illustrated below:

```
Set em_id[0] = 1
Set em_first_name[0] = 'Suren'
Set em_last_name[0] = 'Behari'

Set em_id[1] = 2
Set em_first_name[1] = 'Tom'
Set em_last_name[1] = 'Thumb'

Set em_id[2] = 3
Set em_first_name[2] = 'Jack'
Set em_last_name[2] = 'Beanstalk'

Set em_id[3] = 4
Set em_first_name[3] = 'Snow'
Set em_last_name[3] = 'White'

SqlPLSQLCommand( hSql, 'Employee_pkg.insert_employee(em_id,em_first_
name,em_last_name,nNum)
```

Step 6: Populate the Table Window

Finally we use SQLPLSQLCommand to populate the table window as follows:

```
Set strRecCount = "Employee_Pkg.RecordCount( nRecCount )"
Set nRecCount = 0
Call SqlPLSQLCommand( hSql, strRecCount )
Set strEmpList = "Employee_Pkg.GetEmployeeList( o_emID, o_FirstName,
o_LastName)"
Call SqlPLSQLCommand( hSql, strEmpList )
Set nCount = 0
While nCount < nRecCount
        Call SalTblInsertRow( tblEmployees, TBL_MaxRow )
        Set tblEmployees.colFirstName = o_FirstName[nCount]
        Set tblEmployees.colLastName = o_LastName[nCount]
        Set nCount = nCount +1
If bConnect
        Call SqlDisconnect( hSql )
```

Using OLE DB

The Oracle Provider for OLE DB has a class name of OraOLEDB.Oracle, and gives us pretty much the same level of functionality as Microsoft's Oracle provider (class name is MSDAORA), with the exception that it supports the use of Oracle reference cursors so that we can return back result set cursors from a stored procedure.

REF CURSORS

A reference cursor is a pointer to a memory location that can be passed between different PL/SQL clients, thus allowing query result sets to be passed back and forth between clients.

A reference cursor is a variable type defined using the PL/SQL TYPE statement within an Oracle package, much like a PL/SQL table:

```
TYPE ref_type_name IS REF CURSOR RETURN return_type;
```

Here, ref_type_name is the name given to the type and return_type represents a record in the database. You do not have to specify the return type as this could be used as a general catch-all reference cursor. Such *non-restrictive* types are known as weak, whereas specifying the return type is *restrictive*, or *strong*.
The following example uses %ROWTYPE to define a strong return type that represents the record structure of the emp table:

```
DECLARE TYPE EmpCurType IS REF CURSOR RETURN emp%ROWTYPE;
```

The Oracle Provider for OLE DB allows consumers to execute a PL/SQL stored procedure with an argument of Ref cursor type or a stored function returning a Ref cursor. OraOLEDB can return up to one rowset per PL/SQL stored procedure or function call.

OraOLEDB returns a rowset for the Ref cursor bind variable. Since there is no predefined datatype for Ref cursors in the OLE DB specification, the consumer must not bind this parameter.

If the PL/SQL stored procedure has one or more arguments of Ref cursor type, OraOLEDB binds these arguments appropriately and returns a rowset for the first argument of Ref cursor type.

If the PL/SQL stored function returns a Ref cursor or has an argument of Ref cursor type, OraOLEDB binds these appropriately and returns a rowset. If the stored function does not return a Ref cursor, the rowset is returned for the first Ref cursor argument in the stored function.

To use this feature, stored procedures or functions must be called in the ODBC SQL escape sequence. This feature also requires stored procedures and functions to be in a specific format:

- A package must be created to define all the cursors used in the procedure or function.
- The procedure or function is created using the defined cursors.

Team Developer 3.1 supports Oracle REF CURSORS. To achieve this, you need to use the OLE DB connection string attribute "PLSQLRSET=1" to specify that OraOLEDB must return a rowset from the PL/SQL stored procedure. This MUST be set up in an external UDL file that is pointed to by the System Variable "**SqlUDL**".

PLSQLRSet

The PLSQLRSet connection string attribute specifies whether OraOLEDB needs to parse the PL/SQL stored procedures to determine if a PL/SQL stored procedure returns a rowset. If any of the PL/SQL blocks that the consumer is going to execute returns a rowset, PLSQLRSet needs to be set to 'one'.

This parsing procedure incurs processing overhead of at least one round trip to the database. If none of the PL/SQL stored procedures return a rowset within a particular session, PLSQLRSet should be set to 0. The default value of this attribute is 0.

Oracle Provider for OLE DB does not support PL/SQL Tables (passing an array into a stored procedure). On the other hand, the Oracle Provider for OLE DB does allow consumers to execute a PL/SQL stored procedure with an argument of "Ref cursor" type or a stored function returning a Ref cursor.

Team Developer 2005 allows enabling PLSQLRSet at a statement level. This allows you to enable PLSQLRSet via connection string directly rather than force you to use UDL.

Sample Application

Step 1: Connecting

The sample application starts by prompted for Oracle connection information. I used the Universal Data Link file "**ora.udl**" as my data source, with username and password as "**scott**" and "**tiger**" respectively. The connection uses SQLConnect with the system variable SqlUDL set to the udl file specified as the data source.

Step 2: Create Table

```
CREATE TABLE employee
        (
        em_id           NUMBER(3)   PRIMARY KEY,
        em_first_name       VARCHAR2(30),
        em_last_name        VARCHAR2(30)
        )'
```

Step 3: Create the Package Header

```
CREATE OR REPLACE PACKAGE Employee_pkg AS

 TYPE empcur IS REF CURSOR;

 PROCEDURE insert_employee;

 PROCEDURE RefCurRetAndOutParam(q_cursor OUT empcur, p_cursor OUT
empcur);


END Employee_Pkg;
```

Step 4: Create the Package Body

Once you have defined your stored procedure's specification you can define the implementation in the body section. Because the Oracle OLE DB Provider does not support arrays as input parameters to Stored Procedures, we employ a different technique to insert rows into the employee table.

```
CREATE OR REPLACE PACKAGE BODY Employee_pkg AS

PROCEDURE insert_employee IS
BEGIN
            INSERT INTO EMPLOYEE VALUES ( 1,  \'Suren\', \'Behari\' );
            INSERT INTO EMPLOYEE VALUES ( 2,  \'Tom\', \'Thumb\' );
            INSERT INTO EMPLOYEE VALUES ( 3,  \'Jack\', \'Beanstalk\' );
            INSERT INTO EMPLOYEE VALUES ( 4,  \'Snow\', \'White\' );
END insert_employee;

PROCEDURE RefCurRetAndOutParam(q_cursor OUT empcur, p_cursor OUT
empcur) IS
BEGIN
        OPEN q_cursor FOR
        SELECT em_first_name FROM employee;

        OPEN p_cursor FOR
        SELECT em_first_name, em_last_name FROM employee;
END RefCurRetAndOutParam;

END Employee_Pkg;
```

Step 5: Insert Rows

```
Set strSQL1 = "{ call Employee_Pkg.insert_employee }"
Set bOk = SqlPrepareSP(hSql, strSQL1, STRING_Null )
Set bOk = SqlExecute(hSql)
```

Step 6: Populate the Table Window

```
Set strSQL1 = "{ call Employee_Pkg.RefCurRetAndOutParam( ) }"
Set bOk = SqlPrepareSP(hSql,strSQL1, ' :nCur  ')
Set bOk = SqlExecute(hSql)
Set bEndOfRS = TRUE
Set bOk = SqlGetNextSPResultSet( hSql, ' :tblEmployees.colFirstName,
:tblEmployees.colLastName ', bEndOfRS )
Call SalTblPopulate( tblEmployees, hSql, '', TBL_FillNormal )
If bConnect
        Call SqlDisconnect( hSql )
```

Download Samples

- OraSQLRouterWizard.zip – a sample application demonstrating using the SQLRouter/Oracle to retrieve a result set from an Oracle Stored procedure.
- OraOLEDBWizard.zip – a sample application demonstrating using OLE DB to retrieve a result set from an Oracle Stored procedure.

Reference White Paper

Migrating from SQLRouter for Oracle to OLE DB Using Team Developer 3.1

Requirements:

- Team Developer 3.1
- Oracle server 8i or later
- Oracle client 9i or later
- Oracle OLE DB provider 9.2.0.4 or later

Gupta Technologies, LLC
975 Island Drive
Redwood Shores, CA 94065 USA
Phone +1-650-596-3400
Fax +1-650-596-4690