Using Scrollable Cursor with the Oracle Call Interface Release 9*i*

An Oracle White Paper
May 2001

ORACLE

INTRODUCTION

The Oracle Call Interface (OCI) is an application programming interface (API) that allows an application developer to use a 3rd generation language native procedures or function calls to access the Oracle database server and control all phases of SQL statement execution. The OCI provides a library of standard database access and retrieval functions in the form of a dynamic runtime library, that can be linked in by the application. This eliminates the need to embed SQL or PL/SQL within 3GL language programs. The OCI supports the data types, calling conventions, syntax and semantics of the third generation language. OCI as a generic interface is used throughout the database kernel for distributed database access, for example in Advanced Queuing.

A cursor is a database query and its results set. Execution of a cursor puts the results of the query into a set of rows called the result set This result set can then be fetched sequentially. When all rows have been fetched, and end-of-fetch error is signaled. Rows in a result set are numbered from 1 to n

Scrollable cursors are also called scroll cursors, scrolling cursors or backward scrolling cursors. A scrollable cursor takes the concept of a cursor further by allowing to navigate within the result set. Rows can be retrieved based on absolute cursor position or relative offset position in a given result set, both in forward and backward direction. This means that fetches do not need to be sequential. The OCI scrollable cursor implements statements for fetching the next, previous, last or the n-th row and is read-only.

The paper is organized into the following sections: [i] Motivation for Scrollable Cursor: We describe the reason for implementing a scrollable cursor through the OCI interface; [ii] Programming with Scrollable Cursor: We talk about some of the architectural issues and introduce the new OCI API for the scrollable cursor; [iii] Sample Scrollable Cursor Application: we look at the new OCI statements with the complete program listed in the Appendix, [iv] Recommendations for using Scrollable Cursor are summarized in this section; [v] Updating Rows retrieved with Scrollable Cursor: We look at issuing DML operations on rows retrieved through a scrollable cursor and give implementation guidelines; [vi] Conclusion; [vii] Appendix;

MOTIVATION FOR OCI SCROLLABLE CURSOR

The scrollable cursor feature has been a much requested feature for customers. A number of Oracle development tools will make use of scrollable cursors to help the user navigate through a results set; this includes Oracle Forms, Oracle Discoverer and Oracle Express. The Oracle Migration Workbench which is part of the Oracle 9i database release is another example; mapping the scrollable cursor concept found in other RDBMS will be greatly facilitated.

Previous to Oracle 9i, the database only supported forward sequential cursor operation on a result set. The OCI Scrollable Cursor in release 9i allows to position forward and backward in a given result set, using either absolute or relative position. Many applications, in particular for defining user dialogues and forms are relying on this mechanism to let the user browse through a result set quickly and to select single rows for further processing. The OCI Scrollable Cursor can be applied to offer ease of navigation on result sets to all applications.

The benefits of using OCI Scrollable Cursor are highlighted below:

- **Faster development, less code maintenance**
  With the OCI Scrollable Cursor in Oracle 9i there is no need to implement a separate scrollable cursor with result set buffer, navigational interface and housekeeping. As a consequence, less code development and maintenance is required.
- **Uniform call interface**
  Applications can now use one single uniform interface to access scrollable cursor functionality.
- **Backwards compatibility to OCI 8*i***
  There are no changes to the existing OCI 8*i* interface logic with regards to statement preparation, defining input and output variables, or closing the statement handle; Also, the new scrollable cursors support all operations of current (forward sequential only) cursors.
- **Maximizing performance and scalability**
  With OCI Scrollable Cursor all the other benefits of the OCI API continue to be available. In particular, OCI's pre-fetching and compression capabilities are used to increase throughput and to reduce space requirements for the client-side cache.

PROGRAMMING WITH OCI SCROLLABLE CURSOR

The basic steps for working with a scrollable cursor in an OCI program are:

- Allocate statement handle for scrollable cursor
- Prepare SQL query using SC statement handle
- Set pre-fetching on with value of number of rows
- Bind input and output variables for SQL query

- Execute statement and open scrollable cursor (*new*)
- Fetch rows using forward and backward orientation (*new*)
- Free statement handle

For a detailed description see "*OCI Programmer's Guide Release 9.0.1*".

Feature Overview

A scrollable cursor needs to provide support for forward and backward access into the result set from a given current position, using either absolute or relative row number offsets into the result set.
As there are many ways to now fetch from a (scrollable) result set, there are more fetch orientations possible. Currently, the only fetch orientation was NEXT to get the subsequent row from the current position. The additional fetch orientations introduced are :

- ABSOLUTE - to fetch at arbitrary offsets using absolute position in the result set,
- RELATIVE - to fetch at arbitrary (positive and negative) offsets from the current position,
- FIRST - to fetch the 1st row of the result set
- CURRENT - to fetch the current row (again) from the result set
- LAST - to fetch the last row of the result set
- PREVIOUS - to fetch the previous row from the current position in the result set

After a fetch operation, it is usually necessary to know how many rows actually went into the user's buffers. For non-scrollable cursor, this is currently possible by reading the `OCI_ATTR_ROWS_FETCHED` attribute. Now, the OCI API can directly provide the number of rows fetched successfully into the user's buffers in the last fetch call without another round-trip to the server.

To increase response time, the application can enable client-side caching. The size of the client-side cache is dynamic, and can be set as desired, before the cursor is executed. See section "*Configuring Client-Side Cache*".

The important points of the implementation are summarized below:

- **Forward and backward scrolling**
  The Scrollable Cursor can be set to move in both forward and backward orientation through the result set.
- **Positioning to absolute and relative cursor position**
  When positioning the scrollable cursor, an absolute position within the result set can be indicated; rows in the result set are numbered from 1 to n; positioning the cursor to a row outside the result set will result in an error message.
  The cursor can also be positioned using a relative offset based on the current cursor position in the result set; positioning the cursor to a row outside the result set will result in an error message
  Short forms are provided to position the cursor to the next position, the previous position, the first and to the last position in the result set.

- **Recovery from positioning error**
  In contrast to the default non-scrollable cursor, positioning a scrollable cursor outside the bounds of the result set will not cancel the statement; an error message is issued to the calling application, but the next fetch call will be processed accordingly.
- **Using OCI pre-fetching and compression**
  There is better response time and less memory usage on the client-side due to OCI's pre-fetching and compression capabilities.
- **Minimizing server round-trips**
  With OCI Scrollable Cursor, the number of round-trips to the server is minimal; both the individual rows in a result set and the total size of the result set can be retrieved with one round-trip.
- **Providing client-side caching**
  The caching capability for storing the result set is part of the OCI API. This means that the application does not have to make provisions for memory allocation, cursor navigation and house keeping. This increases the application / mid-tier scalability using existing attributes
- **Scalability through disk-backed storage of result set**
  The result set for a query done with an OCI Scrollable Cursor is transparently managed on the server side. Depending on available memory resources rows will automatically be saved on disk. This means that a given scrollable cursor set-up will scale regardless of the configured memory resources on the server.
- **Read-consistency by using snapshot-based row fetch**
  Whenever a Scrollable Cursor is opened, a statement handle is returned from the OCI interface which points to the result set. This statement handle represents a pointer to a snapshot view of the rows making up the result set. Applications using this statement handle will also use the same snapshot view, thus guaranteeing read-consistency across multiple applications and multiple users.

Statement Execution

When executing the fetch statement with `OCIStmtExecute()` a new mode `OCI_STMT_SCROLLABLE_READONLY` indicates to the database server that the application expects the resulting result set to be scrollable. This mode must be set every time this statement handle is executed, otherwise the cursor will be opened non-scrollable.
The scrollable cursor is opened in read-only mode; DML operations are not allowed through the scrollable statement handle.

New Statement Handle Attributes

The following statement handle attributes are available for the scrollable cursor implementation:

- `OCI_ATTR_CURRENT_POSITION` - Returns the current position of the cursor in the result set. It can be retrieved only by issuing `OCIAttrGet()`
- `OCI_ATTR_ROWS_FETCHED` - Returns the number of rows that were successfully fetched into the client-side cache with the last fetch or execute call.
- `OCI_ATTR_ROW_COUNT` - For scrollable cursors, this returns the highest absolute row number of all the rows that were fetched into the user's buffer since the cursor was executed.
  For non-scrollable cursors, this represents the total number of rows fetched

into the user's buffer using all the fetch calls issued since this statement handle was executed.

Statement Fetch

The new OCI statement `OCIStmtFetch2()` is provided for scrollable cursor support. The existing fetch call `OCIStmtFetch()` is deprecated, though it will still be supported for backward compatibility. It is recommended that you use the new call `OCIStmtFetch2()` in all of your applications. Every fetch on the OCI statement handle will change the current position of the result set, which is initialized to 0 when the query is executed.

The following parameters can be used to set cursor orientation:

- `OCI_FETCH_ABSOLUTE` - fetch row at absolute position in result set using offset;
- `OCI_FETCH_RELATIVE` - fetch row at offset (positive or negative) relative to current cursor position;
- `OCI_FETCH_CURRENT` - fetch row (again) at the current cursor position in the result set;
- `OCI_FETCH_FIRST` - fetch the first row in the result set (position = 0);
- `OCI_FETCH_LAST` - fetch the last row fetched into the result set;
- `OCI_FETCH_PREVIOUS` - fetch previous row from current cursor position in the result set;
- `OCI_FETCH_NEXT` - fetch the next row at current position in the result set; this parameter was already introduced with non-scrollable cursor.

Note: If you continue to use the fetch call `OCIStmtFetch()` an error will be raised if any other orientation besides `OCI_DEFAULT` or `OCI_FETCH_NEXT` is used.

Determining the Size of the Result Set

The fetch parameter `OCI_FETCH_LAST()` positions the cursor to the highest (absolute) row number in the result set which was retrieved with the last `OCIStmtExecute()` call. Retrieving the `OCI_ATTR_CURRENT_POSITION` attribute with `OCIAttrGet()` at this point will return the position of the last row in the result set; this is the total size of the result set. There is no need for executing the query twice to first find the size of the result set, and then to get the column values.
Generally, `OCI_ATTR_CURRENT_POSITION` returns the current position of the cursor within the result set, depending on the last fetch call. This attribute is read-only.
The attribute `OCI_ATTR_ROW_COUNT` will return the (absolute) highest row number fetched into the user's buffer with the last fetch call; used with scrollable cursor this number can vary depending on where the last fetch was positioned. Used in conjunction with the default non-scrollable cursor, `OCI_ATTR_ROW_COUNT` will always represent the total number of rows retrieved with the last execute call.
The attribute `OCI_ATTR_ROWS_FETCHED` represents the number of rows that were successfully retrieved with the last fetch or execute call. It works for both scrollable and non-scrollable cursors.

Error Messages

When fetching row with the new `OCIStmtFetch2()` statement an OCI error may be returned through the error handle. The application must make provisions to process error messages.
Whenever a row is fetched which is outside the result set determined when the statement was executed, OCI_NO_DATA is returned. However, the statement is not cancelled and you can issue another fetch command without re-execution.

SAMPLE OCI SCROLLABLE CURSOR APPLICATION

The OCI Scrollable Cursor is illustrated with a sample application. As there is no difference in the OCI statements necessary to create the OCI environment, we will not include these OCI statements here for brevity. The complete source code is included in "*Appendix A*".

Setting up the Database

The database tables needed for this sample application are created using the following SQL statements:

```
create or replace type empaddr as object (
       state char(2),
       zip   number);

create or replace type evarray is VARRAY(2) of number;

create table empo (empno number,
       ename char(5),
       addr  empaddr,
       ecoll evarray);
```

Preparing SQL Query

We will issue the following SQL query against the table "empo":

```
       SELECT empno, ename, addr, ecoll FROM empo
```

OCI Statements for Scrollable Cursor

A cursor is opened in scrollable mode by issuing `OCIStmtExecute()` with the new `OCI_STMT_SCROLLABLE_READONLY` parameter. The `OCIStmtFetch2()` call is then used to navigate through the result set. We assume that the table contains 30 employee records.

```
/* execute statement and retrieve result set */
/* svchp = service handle, stmthp = statement handle, errhp =
error handle */
OCIStmtExecute(svchp, stmthp, errhp, (ub4) 0, (ub4) 0, (CONST
OCISnapshot *) NULL, (OCISnapshot *) NULL,
OCI_STMT_SCROLLABLE_READONLY)

OCIStmtFetch2(stmthp, errhp, (ub4) 1, OCI_FETCH_LAST, (sb4) 0,
OCI_DEFAULT)

OCIAttrGet  (stmthp, OCI_HTYPE_STMT, (void *) &currentPosition,
(ub4 *) 0, OCI_ATTR_CURRENT_POSITION, errhp)
/* assume currentPosition = 30, i.e. there are at least 30 rows
in the result set */
```

```
OCIStmtFetch2(stmthp, errhp, (ub4) 5, OCI_FETCH_FIRST, (sb4) 0,
OCI_DEFAULT)
/* fetches 5 rows from the 1st row, the current position now is
5 */

OCIStmtFetch2(stmthp, errhp, (ub4) 15, OCI_FETCH_RELATIVE,
(sb4) -2, OCI_DEFAULT)
/* fetches 15 rows from relative offset -2 with respect to the
current position 5, current position is now 12*/

OCIAttrGet  (stmthp, OCI_HTYPE_STMT, (void *) &currentPosition,
(ub4 *) 0, OCI_ATTR_CURRENT_POSITION, errhp)
/* current position = 12 */

OCIAttrGet (stmthp, OCI_HTYPE_STMT, (void *) &rowsFetched, (ub4
*) 0, OCI_ATTR_ROWS_FETCHED, errhp)
/* if the error handle returned end of fetch error, or
rowsFetched < 15 we know that we passed the end of the result
set */

… more fetch calls …

/* close or cancel the statement handle */
OCIHandleFree((dvoid *) envhp, OCI_HTYPE_ENV);
```

RECOMMENDATIONS FOR USING OCI SCROLLABLE CURSOR

Configuring the Client-Side Cache

The existing OCI pre-fetching attributes - `OCI_ATTR_PREFETCH_ROWS` and `OCI_ATTR_PREFETCH_MEMORY` can be used to control the size of the client-cache for both scrollable and non-scrollable result sets. See also page 4-17 in the *OCI Programmer's Guide Release 9.0.1*.

These attributes control the number of rows that are pre-fetched into the user's buffer so that subsequent fetch operations incur no or less round-trips to the database server. The attribute is set after statement preparation with the `OCIAttrGet()` call.

Canceling a Scrollable Cursor Statement

Issuing a `OCIStmtFetch2()` call with the *nrows* parameter set to 0 will cancel the statement, that is it will free all resources on the server. If you want to use the scrollable cursor statement again to retrieve rows from the same result set, you have to re-execute the statement handle again with the `OCIStmtExecute()` call. See also page 5-91 in the *OCI Programmer's Guide Release 9.0.1*.

Releasing Scrollable Cursor Resources

Scrollable cursors are more resource-intensive than non-scrollable cursors for the mid-tier and the database-server. Hence it is advised that statement handles should not be made arbitrarily scrollable, and only when applicable. The new fetch statement `OCIStmtFetch2()` can always be used for both scrollable and non-scrollable cursors.
A scrollable cursor statement handle is released as any other OCI handle by using the `OCIHandleFree()` call.

Recommendations and Limitations

Both the client and server should be of release Oracle9i for scrollable cursors to work.
The Oracle 9i scrollable cursor is currently only exposed via the Oracle Call Interface and is read-only.

UPDATING RECORDS RETRIEVED WITH OCI SCROLLABLE CURSOR

•

The rows in a result set retrieved through an OCI Scrollable Cursor represent a snapshot view; DML operations on any row or rows within this result set are not supported.
You can, however, issue DML operations based on the result set. For example, if the result represent s employee records, you may ver well update any dependant information based on this result set. In the case of the employee record, this could be the address information stored in a separate table.

If you want to issue a DML operations against a row or rows within the result set, follow the following guidelines:

- **Declare ROWID variable(s) with `OCIRowid` descriptor**
  Prepare Scrollable Cursor statement with statement handle `scstmthp`
- **Set pre-fetching on for Scrollable Cursor statement `scstmthp`**
- **Execute statement with `scstmthp` and open Scrollable Cursor**
- **Perform fetches as required by the application**
- **Prepare DML statement with new statement handle `dmlstmthp`**
- **Retrieve ROWID for rows(s) in the result set**
  This can be done without another round-trip to the server by positioning the cursor to the desired row and reading the `OCI_ATTR_ROWID` attribute with the `OCIGetAttr()`
- **Bind ROWID input variable(s) for DML statement `dmlstmthp`**
  Execute DML statement `dmlstmthp`
  Refresh result set to reflect changes by executing `scstmthp` again
  After each DML operation against the result set, you should refresh the result set by re-executing the scrollable cursor statement.

CONCLUSION

The Oracle Call Interface Release 9i supports a scrollable cursor for navigating in a result set both in forward and backward orientation. The cursor can be positioned by using either absolute row number in the result set, or a relative offset from the current position in the result set.

The query must be executed with the new `OCI_STMT_SCROLLABLE_READONLY` attribute. Use the new `OCIStmtFetch2()` to fetch rows through a scrollable cursor. This statement also works for a non-scrollable forward cursor; the previous `OCIStmtFetch()` is supported for backwards compatibility.
OCI Scrollable Cursor provides a uniform and transparent interface and relieves the application from managing buffers, memory and housekeeping. The pre-fetching and compression capabilities of the OCI interface help reduce server

round-trips and increase performance. Disked-backed buffering of the result set on the server allows to scale according to the application needs.

For more information on scrollable cursors or OCI, refer "*Oracle Call Interface - Programmers Guide: Release 9.0.1*"

APPENDIX A

The following source program demonstrates the use of an OCI Scrollable Cursor. For the purpose of this demonstration we assume a total of 14 employee records.

```
/*  Copyright  (c)  2001,  Oracle  Corporation.    All  rights
reserved.

   NAME
     cdemosc.c - OCI demo program for scrollable cursor.

   DESCRIPTION
     An example program which reads employee records from table
empo.

     SQL> describe empo;
     Name                      Null?    Type
     ------------------------- -------- ---------------
     EMPNO                              NUMBER
     ENAME                              CHAR(5)
     ADDR                               EMPADDR
     ECOLL                              EVARRAY

     SQL> describe empaddr;
     Name                      Null?    Type
     ------------------------- -------- ---------------
     STATE                              CHAR(2)
     ZIP                                NUMBER

   EXPORT FUNCTION(S)
     <external functions defined for use outside package - one-
line descriptions>

   INTERNAL FUNCTION(S)
     <other external functions defined - one-line descriptions>

   STATIC FUNCTION(S)
     <static functions defined - one-line descriptions>

   NOTES
     Dependent Files:
       cdemosc.sql  - SQL script to be run before execution.
       Modify the OCIServerAttach call to the use the
appropriate
       service instead of "prodinst1"

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef OCI_ORACLE
#include <oci.h>
#endif

typedef struct address
{
  OCIString * state ;
  OCINumber zip;
} address ;

typedef struct null_address
```

```
{
  sb2 null_state ;
  sb2 null_zip;
} null_address ;


/*---------------------------------------------------------------
---------
PRIVATE TYPES AND CONSTANTS
  ---------------------------------------------------------------
-------*/

#define MAXROWS 3
#define MAX_ENAMELEN 20

static text *username = (text *) "scott";
static text *password = (text *) "tiger";

/* Define SQL statement to be used in program. */
static text *selemp = (text *)"SELECT empno, ename, addr, ecoll
FROM empo";

/* Define static variables and OCI handles     */
static sword  empno[MAXROWS] ;
static text          empname[MAXROWS][MAX_ENAMELEN];
static address      *empaddr [MAXROWS] ;
static null_address *indaddr [MAXROWS] ;
static ub4           rc[MAXROWS] ;
static sword  status;
static OCIColl      *evarray[MAXROWS] ;

static OCIEnv *envhp;
static OCIError     *errhp;
static OCISvcCtx    *svchp;
static OCIStmt      *stmthp;

/*---------------------------------------------------------------
---------
STATIC FUNCTION DECLARATIONS
  ---------------------------------------------------------------
-------*/

static void checkerr(/*_ OCIError *errhp, sword status _*/);
static void checkprint(/*_ OCIError *errhp, sword status, ub4
nrows _*/);
static void cleanup(/*_ void _*/);
static void myprint (/*_ ub4 _*/);
int main(/*_ int argc, char *argv[] _*/);


/* Begin main program */
int main(argc, argv)
int argc;
char *argv[];
{

  OCISession *authp = (OCISession *) 0; /* Authentication
Handle */
  OCIServer  *srvhp;                    /* Server Handle */
  OCISvcCtx  *svchp;                    /* Service Context
Handle */
  OCIType    *addr_tdo1;         /* Bind output variable 1 */
  OCIType    *addr_tdo2;         /* Bind output variable 2 */
  ub4          empaddrsz;              /* Save size of EMPADDR
structure */
```

```
  ub4           prefetch = 5;     /* set number of prefetched
rows  */
  int           num;                     /* save number of rows
*/
  OCIDefine  *defn1p = (OCIDefine *) 0; /* Define input
variable 1 */
  OCIDefine  *defn2p = (OCIDefine *) 0; /* Define input
variable 2 */
  OCIDefine  *defn3p = (OCIDefine *) 0; /* Define input
variable 3 */
  OCIDefine  *defn4p = (OCIDefine *) 0; /* Define input
variable 4 */
  OCIDefine  *defn5p = (OCIDefine *) 0; /* Define input
variable 5 */


  /* Initialize OCI session and set mode */
  (void) OCIInitialize((ub4) OCI_DEFAULT | OCI_OBJECT, (dvoid
*)0,
                       (dvoid * (*)(dvoid *, size_t)) 0,
                       (dvoid * (*)(dvoid *, dvoid *,
size_t))0,
                       (void (*)(dvoid *, dvoid *)) 0 );

  (void) OCIEnvInit( (OCIEnv **) &envhp, OCI_DEFAULT, (size_t)
0,
                     (dvoid **) 0 );

  /* allocate error handle for this environment */
  (void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &errhp,
OCI_HTYPE_ERROR,
                   (size_t) 0, (dvoid **) 0);

  /* create server context */
  (void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &srvhp,
OCI_HTYPE_SERVER,
(size_t) 0, (dvoid **) 0);

  (void) OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &svchp,
OCI_HTYPE_SVCCTX,
(size_t) 0, (dvoid **) 0);

  checkerr(errhp, OCIServerAttach( srvhp, errhp, (text
*)"prodinst1",
(sb4) strlen("prodinst1"), 0));

  /* set attribute server context in the service context */
  (void) OCIAttrSet( (dvoid *) svchp, OCI_HTYPE_SVCCTX, (dvoid
*)srvhp,
(ub4) 0, OCI_ATTR_SERVER, (OCIError *) errhp);

  /* allocate session handle and establish connection */
  (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **)&authp,
(ub4) OCI_HTYPE_SESSION, (size_t) 0, (dvoid **) 0);

  (void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
(dvoid *) username, (ub4) strlen((char *)username),
(ub4) OCI_ATTR_USERNAME, errhp);

  (void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
(dvoid *) password, (ub4) strlen((char *)password),
(ub4) OCI_ATTR_PASSWORD, errhp);

  checkerr(errhp, OCISessionBegin ( svchp,  errhp, authp,
OCI_CRED_RDBMS,
```

```
(ub4) OCI_DEFAULT));

  (void) OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX,
(dvoid *) authp, (ub4) 0,
(ub4) OCI_ATTR_SESSION, errhp);


  /* allocate statement handle for scrollable cursor */
  checkerr(errhp, OCIHandleAlloc((dvoid *) envhp, (dvoid **)
&stmthp,
OCI_HTYPE_STMT, (size_t) 0, (dvoid **) 0));

  checkerr(errhp, OCIStmtPrepare(stmthp, errhp, selemp,
(ub4) strlen((char *) selemp),
(ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

/* set prefetch size for cursor statement handle */
  (void) OCIAttrSet((dvoid *) stmthp, (ub4) OCI_HTYPE_STMT,
(dvoid *) &prefetch, 0,
(ub4) OCI_ATTR_PREFETCH_ROWS, errhp);

  /* bind the input variables */
  checkerr(errhp, OCIDefineByPos(stmthp, &defn1p, errhp, 1,
(dvoid *) empno,
                           (sword) sizeof(sword), SQLT_INT,
(dvoid *) 0,
                           (ub2 *)0, (ub2 *)0, OCI_DEFAULT));

  checkerr(errhp, OCIDefineByPos(stmthp, &defn2p, errhp, 2,
(dvoid **) empname,(sword) MAX_ENAMELEN, SQLT_STR, (dvoid *)
0,(ub2 *)0,(ub2 *)0, OCI_DEFAULT));

  checkerr(errhp, OCITypeByName(envhp, errhp, svchp, (const
text *) 0,
                           (ub4) 0, (const text *) "EMPADDR",
                           (ub4) strlen((const char *)
"EMPADDR"),
                           (CONST text *) 0, (ub4) 0,
OCI_DURATION_SESSION,  OCI_TYPEGET_ALL,
                           &addr_tdo1));

  checkerr(errhp, OCIDefineByPos(stmthp, &defn3p, errhp, 3,
(dvoid *) 0,
                           (sword) 0, SQLT_NTY, (dvoid *) 0,
(ub2 *)0,
(ub2 *)0, OCI_DEFAULT));

  checkerr(errhp, OCIDefineObject(defn3p, errhp, addr_tdo1,
(dvoid **) empaddr, (ub4 *) &empaddrsz,
(dvoid **) indaddr, (ub4 *) rc));

  checkerr(errhp, OCITypeByName(envhp, errhp, svchp, (const
text *) 0,
                           (ub4) 0, (const text *) "EVARRAY",
                           (ub4) strlen((const char *)
"EVARRAY"),
                           (CONST text *) 0, (ub4) 0,
OCI_DURATION_SESSION,  OCI_TYPEGET_ALL,
                           &addr_tdo2));

  checkerr(errhp, OCIDefineByPos(stmthp, &defn4p, errhp, 4,
(dvoid *) 0,
                           (sword) 0, SQLT_NTY, (dvoid *) 0,
(ub2 *)0,
                           (ub2 *)0, OCI_DEFAULT));
```

```
  checkerr(errhp, OCIDefineObject(defn4p, errhp, addr_tdo2,
(dvoid **) evarray, (ub4 *) 0,
(dvoid **) 0, (ub4 *) 0));

  /**** SCROLLABLE RESULT SET *****/

  /* open scrollable cursor and retrieve the employee records
*/
  checkprint(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4)
0, (ub4) 0,
                          (CONST OCISnapshot *) NULL,
(OCISnapshot *) NULL,
OCI_STMT_SCROLLABLE_READONLY ),0);
  /******** Current Position, Row Count = 0, 0 ********/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 3,
                          OCI_FETCH_ABSOLUTE, (sb4) 6,
OCI_DEFAULT),3);
  /******** Current Position, Row Count = 8, 8 ********/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 3,
                          OCI_FETCH_RELATIVE, (sb4) -2,
OCI_DEFAULT),3);
  /******** Current Position, Row Count = 8, 8 ********/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 2,
                          OCI_FETCH_ABSOLUTE, (sb4) 9,
OCI_DEFAULT),2);
  /******** Current Position, Row Count = 10, 10 ********/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 1,
                          OCI_FETCH_LAST, (sb4) 0,
OCI_DEFAULT),1);
  /******** Current Position, Row Count = 14, 14 ********/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 1,
                          OCI_FETCH_FIRST, (sb4) 0,
OCI_DEFAULT),1);
  /******** Current Position, Row Count = 1, 14 ********/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 1,
                          OCI_FETCH_LAST, (sb4) 0,
OCI_DEFAULT),1);
  /******** Current Position, Row Count = 14, 14 ********/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 1,
                          OCI_FETCH_FIRST, (sb4) 0,
OCI_DEFAULT),1);
  /******** Current Position, Row Count = 1, 14 ********/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 1,
OCI_FETCH_ABSOLUTE, (sb4) 15, OCI_DEFAULT),1);
  /******** Error - OCI_NODATA ************************/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 2,
                          OCI_FETCH_NEXT, (sb4) 0,
OCI_DEFAULT),2);
  /******** Current Position, Row Count = 2, 14 ********/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 1,
                          OCI_FETCH_PRIOR, (sb4) 0,
OCI_DEFAULT),1);
  /******* Error - OCI_NODATA ************************/
```

```
  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 3,
                                  OCI_FETCH_RELATIVE, (sb4) -8,
OCI_DEFAULT),3);
  /******** Current Position, Row Count = 4, 14 ********/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 2,
                                  OCI_FETCH_NEXT, (sb4) 0,
OCI_DEFAULT),2);
  /******** Current Position, Row Count = 1, 14 ********/

  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 1,
                                  OCI_FETCH_FIRST, (sb4) 0,
OCI_DEFAULT),1);
  /******** Current Position, Row Count = 0, 14 ********/

  /* cancel the statement handle ... */
  checkprint(errhp, OCIStmtFetch2(stmthp, errhp, (ub4) 0,
                                  OCI_FETCH_NEXT, (sb4) 0,
OCI_DEFAULT),0);
  /******** Current Position, Row Count = 0, 14 ********/

  /* ... and free resources on client and server */
  cleanup() ;

  /* ... return to caller with exit code */
  return 1;
}

/* SUPPORTING ROUTINES FOR DEMO PROGRAM */

/*check fetch status and print rows upon success*/
void checkprint(errhp, status,nrows)
OCIError *errhp;
sword status;
ub4 nrows;
{
  checkerr(errhp,status);
  if (status != OCI_ERROR && status != OCI_NO_DATA)
    myprint(nrows);
}


/*check status and print error information */
void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
  text errbuf[512];
  sb4 errcode = 0;

  switch (status)
  {
  case OCI_SUCCESS:
    break;
  case OCI_SUCCESS_WITH_INFO:
    (void) printf("Error - OCI_SUCCESS_WITH_INFO\n");
    break;
  case OCI_NEED_DATA:
    (void) printf("Error - OCI_NEED_DATA\n");
    break;
  case OCI_NO_DATA:
    (void) printf("Error - OCI_NODATA\n");
    break;
  case OCI_ERROR:
```

```
    (void) OCIErrorGet((dvoid *)errhp, (ub4) 1, (text *) NULL,
&errcode,
                       errbuf, (ub4) sizeof(errbuf),
OCI_HTYPE_ERROR);
    (void) printf("Error - %.*s\n", 512, errbuf);
    break;
  case OCI_INVALID_HANDLE:
    (void) printf("Error - OCI_INVALID_HANDLE\n");
    break;
  case OCI_STILL_EXECUTING:
    (void) printf("Error - OCI_STILL_EXECUTE\n");
    break;
  case OCI_CONTINUE:
    (void) printf("Error - OCI_CONTINUE\n");
    break;
  default:
    break;
  }
}


/*
 *  Exit program with an exit code.
 */
void cleanup()
{
  if (envhp)
    (void) OCIHandleFree((dvoid *) envhp, OCI_HTYPE_ENV);
  return;
}


/*void myfflush()
{
  eb1 buf[50];

  fgets((char *) buf, 50, stdin);
}*/


/*print rows*/
void myprint (ub4 nrows)
{
  int i, j, num, cp, rc, amount ;
  sb4 colsz;
  void * elem ;
  ub4 sz = sizeof(cp) ;
  boolean exist = FALSE;

  checkerr(errhp, OCIAttrGet((CONST void *) stmthp,
OCI_HTYPE_STMT,
                             (void *) & cp, (ub4 *)  & sz,
                             OCI_ATTR_CURRENT_POSITION,
errhp));
  checkerr(errhp, OCIAttrGet((CONST void *) stmthp,
OCI_HTYPE_STMT,
                             (void *) & rc, (ub4 *)  & sz,
                             OCI_ATTR_ROW_COUNT, errhp));
  printf("******** Current position, Row Count = %d, %d
******** \n", cp, rc);

  for (i =0 ; i < nrows ; i++ )
  {
    OCINumberToInt(errhp, & empaddr[i]->zip, sizeof(num),
                   OCI_NUMBER_SIGNED, (void *) & num);
```

```
    printf("\n %d %s %s %d ", empno[i], empname[i],
            OCIStringPtr(envhp, empaddr[i]->state), num);

    colsz = OCICollMax (envhp, (OCIColl *)  evarray[i]) ;
    for (j = 0; j < colsz ; j++)
      {
        OCICollGetElem (envhp, errhp, (OCIColl *) evarray[i],
j, & exist,
                        (void **) & elem, (void **) 0);
        if (!exist)
          printf(" *** error - coll, row %d col-elem %d ", i,
j);
        else {
          checkerr(errhp, OCINumberToInt(errhp, (OCINumber *)
elem,
                              sizeof(int), OCI_NUMBER_SIGNED, &
num));
          printf("%d ", num);
        }
      }
      printf ("\n");
  }
}

/* end of file cdemosc.c */
```

```
/*

  Copyright (c) Oracle Corporation 2001. All Rights Reserved.

    NAME
      cdemosc.sql - Demo program for scrollable cursor.

    DESCRIPTION
      SQL script to prepare table empo and data in the table.

    NOTES
      Neet to run before cdemosc.

*/


connect scott/tiger;
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET ECHO ON

drop table empo
/
drop type evarray
/
drop type empaddr
/

create or replace type empaddr as object (
       state char(2) ,
       zip number )
/
```

```
create or replace type evarray is VARRAY(2) of number
/

create table empo (empno number,
       ename char(5),
       addr empaddr,
       ecoll evarray)
/

insert into empo values (1, 'abc1', empaddr('ca', 94061),
evarray(13,145))
/
insert into empo values (2, 'abc2', empaddr('ca', 94062),
evarray(23,245))
/
insert into empo values (3, 'abc3', empaddr('ca', 94063),
evarray(33,345))
/
insert into empo values (4, 'abc4', empaddr('ca', 94064),
evarray(43,445))
/
insert into empo values (5, 'abc5', empaddr('ca', 94065),
evarray(53,545))
/
insert into empo values (6, 'abc6', empaddr('ca', 94066),
evarray(63,645))
/
insert into empo values (7, 'abc7', empaddr('ca', 94067),
evarray(73,745))
/
insert into empo values (8, 'abc8', empaddr('ca', 94068),
evarray(83,85))
/
insert into empo values (9, 'abc9', empaddr('ca', 94069),
evarray(93,95))
/
insert into empo values (10, 'abc10', empaddr('ca', 94060),
evarray(103,1045))
/
insert into empo values (11, 'abc11', empaddr('ca', 94070),
evarray(113,1045))
/
insert into empo values (12, 'abc12', empaddr('ca', 94071),
evarray(123,1045))
/
insert into empo values (13, 'abc13', empaddr('ca', 94072),
evarray(133,1045))
/
insert into empo values (14, 'abc14', empaddr('ca', 94073),
evarray(143,1045))
/
```

Using Scrollable Cursor with the Oracle Call Interface Release 9i

May 2000
Author: Bernhard Düchting
Contributing Authors: Mehul Bastawala, Luxi Chidambaran, Srinath Krishnaswamy

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Oracle Corporation provides the software that powers the internet.