R S I

ORACLE

HOST LANGUAGE INTERFACE

Oracle Programmer's Guide - Version 2.3

O R A C L E

HOST LANGUAGE INTERFACE

TABLE OF CONTENTS

O R A C L E

HOST LANGUAGE INTERFACE

INTRODUCTION

SQL is designed to be used as a stand-alone terminal language for interactive users, and as a data sublanguage embedded in a host programming language. All SQL query, data manipulation, data definition, and data control facilities are available from both the interactive and data sublanguage interface.

ORACLE interfaces to FORTRAN, COBOL, PL/1, "C" and other high level host programming languages by means of program calls. ORACLE interfaces to assembly language via macro instructions.

A program establishes communication with ORACLE by issuing the LOGON call. Communication takes place via the Logon Data Area (LDA) defined within the user program. A user program issues one and only one LOGON to ORACLE.

A program opens a database and creates a "cursor" by issuing an OPEN call. A cursor is the name of a data area which is used to identify and control an active SQL statement. The Cursor data area is defined within the user program. ORACLE permits user programs to have multiple active SQL statements. This is accomplished by a single program issuing multiple OPEN's to establish multiple cursors.

The SQL call is used to associate a SQL statement with a cursor. In the case of a query, the SQL call defines a set of rows to be retrieved and logically positions the cursor just before the first row.

Subsequent calls related to the same SQL statement reference the same cursor.

The following sections assume that the reader is familiar with SQL, section 3.12 of the "ORACLE Introduction", and at least one programming language.

O R A C L E

HOST LANGUAGE INTERFACE

PROGRAM CALL SUMMARY

There are eleven different ORACLE calls or macros that may be used to access the database.

The LOGON call establishes communication between a program and ORACLE.

The OPEN call connects a program to a database and creates a cursor.

The SQL call is used to pass a SQL statement to ORACLE.

The DESCRIBE call is used to dynamically determine the number and data types of fields to be retrieved during a query operation.

The NAME call is used to retrieve table and column names.

The DEFINE call identifies to ORACLE the location of data field buffers in the user program.

The BIND call allows programs to dynamically substitute variables into SQL statements.

The EXECUTE call causes ORACLE to process the SQL statement.

The FETCH call is used to retrieve rows, one at a time, during a program query operation.

The CLOSE call closes the database and disconnects the cursor from ORACLE.

The LOGOFF call disconnects the user program from ORACLE.

O R A C L E

COMPILER LANGUAGE INTERFACE

CODING RULES

The following general rules apply to user programs calling ORACLE:

literals	may be used within the CALL parameter list if they are permitted by the compiler. Note that the use of a literal must generate a pointer to the literal as a result of the CALL; the PDP-11 compilers always generate pointers to literals, but the VAX compilers may generate a pointer to a character string descriptor which cannot be used by the ORACLE Host Language Interface. An example is the VAX FORTRAN compiler which generates pointers to descriptors for all references to character data types. In order to circumvent the FORTRAN calling defaults, use the "%REF" function to force a call by reference.
variable-length fields	variable length parameters are passed to ORACLE with an accompanying length field in the form: parameter,length.
length field formats	Length fields are binary numbers (FORTRAN INTEGER; COBOL PICTURE S9999 COMP; etc.) of the standard word size for the computer on which ORACLE is running (32 bits for the VAX, 16 bits for the PDP-11). The length field may be omitted if the variable parameter is terminated by a binary zero.

missing parameters	If a length field or other optional parameter is omitted from a call parameter list, the user may code comma comma (,,) to indicate the absence of the parameter. For example: "param1,len1,param2,len2 vs. param1,,param2,,". In some languages ("C" for example) the comma comma notation is not allowed to indicate a missing parameter in a call parameter list. If comma comma notation is not permitted, the user may code a minus one (-1) to indicated the missing parameter. For example: (param1,-1,param2,-1).

The LOGON Call

```
CALL OLOGON (lda[,areacount])
```

The LOGON call establishes communication between ORACLE and a user program.

Communication takes place via the Logon Data Area (LDA) defined within the user program. The LOGON call connects the LDA to ORACLE. A program logs on to ORACLE one and only one time. A program has one and only one LDA.

The LDA is a 64 byte data area defined within the user program. The first two bytes of the LDA contains a return code indicating the result of the LOGON. A zero return code indicates a successful LOGON. Error return codes are listed in the "Messages and Return Codes" section of this manual.

lda	specifies the name of the 64 byte Logon Data Area defined within the user program.
areacount	specifies an integer number indicating the number of ORACLE SQL statement work areas (SWA's) to be concurrently maintained in main storage. This optional parameter is used only if the user program is opening multiple cursors and and the user does not want ORACLE to swap SWA's. (See the section on "Program Interface Data Areas." at the end of this manual.) Areacount should be equal to or less than the maximum number of cursors the program will open. The default value for areacount is one.

LOGON Examples:

FORTRAN

Log on to ORACLE using a Logon Data Area named LDAREA.

```
CALL OLOGON(LDAREA)
```

COBOL

Log on to ORACLE using a Logon Data Area named LDAREA and a integer named AREACOUNT.

```
CALL "OLOGON" USING LDAREA,AREACOUNT.
```

MACRO-11 ASSEMBLY LANGUAGE

Log on to ORACLE specifying the address of the LDA in register 2.

```
CCALL OLOGON,R2
```

The OPEN Call

```
CALL OOPEN (cursor,lda,dbn,dbnlen[,areasize][,uid,uidlen])
```

The OPEN call establishes a cursor to operate on a specific database.

A cursor is a data area defined within the user program. The OPEN connects the cursor to ORACLE. The cursor name is used to identify an active SQL statement within the user program.

Each cursor may control only one SQL statement at a time. The same cursor may be reused to control another SQL statement after the first statement's operation has been completed.

A single user program can have multiple SQL statements active at the same time. This is accomplished by issuing multiple OPEN's to establish multiple cursors within the program. These OPEN's can be to the same or different databases.

The cursor data area contains status information on an active SQL operation. All subsequent ORACLE calls referencing a SQL statement reference it by cursor name. The first two bytes of the cursor contain a return code indicating the result of the OPEN. A zero return code indicates a successful OPEN. Error return codes are listed in the "Messages and Return Codes" section of this manual.

cursor	specifies the name of a 64 byte data area within the user program. The cursor data area is connected to ORACLE by the OPEN call. Each cursor defines an active SQL statement within the program.
lda	specifies the name of the Logon Data Area specified in the LOGON call.
dbn	specifies the name of the ORACLE database as defined in the Database File (DBF) utility.
dbnlen	specifies a binary integer indicating the length of the database name. If the database name was specified as a literal, this parameter may be omitted.

areasize	specifies a binary integer indicating the size of the ORACLE SQL Work Area (SWA) in increments of 1K bytes. This optional parameter is used only if the user wants a work area other than the default size of 3K bytes. The SWA must be large enough to contain the compiled SQL statement plus one row of data of the table or view being processed. ORACLE SQL work areas can vary in size from 1 to 16. See the section on "Program Interface Data Areas" at the end of this chapter. Note that if multiple cursors of different sizes are to be opened, the one with the largest SWA size must be opened first.
uid	specifies the user identification and password as defined by either the Database File (DBF) utility or the SQL "DEFINE USER" function.
uidlen	specifies a binary integer indicating the length of the user identification and password. If they were specified as a literal, this parameter may be omitted.

OPEN Examples:

FORTRAN

Open the PERSONNEL database and establish the cursor CURS1. The name of the Logon Data Area (LDA) is LDAREA. Specify the database name and user id as literals and take the default 3K SQL work area size.

```
CALL OOPEN(CURS1,LDAREA,'PERSONNEL',,,'SCOTT/TIGER')
```

Open the personnel database and establish the cursor CURS2. The name of the program's LDA is LDAREA. Specify the database name and the length of the database name as program variables DBN and DBL. Specify a 5K SQL work area. The user identification and password are not specified, i.e., the database is not secure.

```
CALL OOPEN(CURS2,LDAREA,DBN,DBL,5)
```

COBOL

Open the personnel database and establish a cursor CURS1. The name of the Logon Data Area is LDAREA, the database name is contained in a variable named DBN, its length is in DBNLEN, the area size is in a variable named AREASIZE, the user identification and password are contained in a variable named UID, and its length is in UIDL.

```
CALL "OOPEN" USING CURS1,LDAREA,DBN,DBNLEN,AREASIZE,UID,UIDL.
```

MACRO-11 ASSEMBLY LANGUAGE

Open the personnel database and pass the address of the cursor to ORACLE in register Zero. The address of the program's LDA is in register 2. Specify the database name in program variable DBN and pass the length of the database name as a literal. Specify a 1K SQL work area. The user identification and password are not specified, i.e., the database is not secure.

```
OOPEN R0,R2,BN,#9,#1
```

The SQL Call

```
CALL OSQL (cursor,sqlstatement,sqllen)
```

The SQL call passes a SQL statement to ORACLE, and associates that SQL statement with an open cursor. Subsequent calls reference the SQL statement by cursor name.

The SQL call may contain any valid SQL query, data manipulation, data definition or data control statement. ORACLE parses the statement and selects an optimal access path to perform the requested function, however, the operation is not executed at this time.

SQL syntax error codes will be returned in the cursor RETURN-CODE area along with a pointer to the text in error. See the parse errors section in Messages and Return Codes for a complete list of syntax errors.

cursor	specifies the name of a 64 byte data area within the user program. The cursor data area contains status information on an active SQL operation. Each cursor defines an open SQL statement within the user program. The SQL call attaches a SQL statement to the cursor. A cursor may be serially reused by subsequent SQL calls within a user program, or the program may define multiple concurrent cursors.
sqlstatement	specifies any valid SQL query, data manipulation, data definition, or data control statement. The statement may contain substitution variables anywhere a constant is permitted. Substitution variables are identified by preceding the variable name with an ampersand, i.e. &EMPNO. These substitution variables then may be referenced symbolically in a BIND call.
sqllen	specifies a binary integer containing the length of the SQL statement. If the SQL statement was specified as a literal this parameter may be omitted.

SQL Examples:

FORTRAN

Pass a SQL query statement to ORACLE using the cursor CURS1. Specify the SQL statement as a literal.

```
CALL OSQL(CURS1,'SELECT ENAME,SAL FROM EMP WHERE DEPTNO =
&DNO;')
```

Pass a SQL statement to ORACLE using the cursor CURS2. Specify the SQL statement as a program variable named QUERY1 with the length of the SQL statement specified as as literal.

```
CALL OSQL(CURS2,QUERY1,28)
```

COBOL

Pass the SQL statement contained in a variable named SQLSTM with length contained in SQLSTML to ORACLE. Use the cursor named CURS1.

```
CALL "OSQL" using CURS1,SQLSTM,SQLSTML).
```

MACRO-11 ASSEMBLY LANGUAGE

Pass a SQL statement to ORACLE using the cursor CURS1. Register 1 points to the length of the SQL statement. Register 1 plus 2 points to the the SQL statement itself.

```
CCALL OSQL,#CURS1,2(R1),(R1)
```

The DESCRIBE Call

```
CALL ODSRBN (cursor,position[,dbsize][,dbtype][,fsize])
```

The DESCRIBE call returns internal data type and size information for a field or expression listed in the SELECT clause of a query statement.

DESCRIBE operates positionally, one field at a time, referencing each field in the SELECT clause as if each were numbered consecutively, left to right, beginning with 1.

DESCRIBE can be used after a SQL, EXECUTE, or FETCH call to determine the maximum size and internal data type (dbsize & dbtype) of fields to be returned as the result of a query. If DESCRIBE is used after a FETCH Call, the actual size of the field just fetched (fsize) may be returned in addition to dbsize and dbtype.

If the user specifies a position number greater than the number of fields in the SELECT list, DESCRIBE returns an end-of-file indicator in the RETURN-CODE of the cursor data area. This allows programs to dynamically determine the number of fields to be returned as the result of a query. This is necessary if the program does not know in advance how many fields there are in the SELECT list, as in the case of SELECT *.

cursor	specifies the name of a 64 byte data area within the user program. The cursor data area contains status information on an active SQL operation. Each cursor defines an active SQL statement within the user program. DESCRIBE uses the cursor name to reference a specific SQL query statement that had been passed to ORACLE in a prior SQL call. The RETURN CODE area of the cursor indicates success (code of 0) or failure (non-zero) of the DESCRIBE call. All error return codes are listed in the "Messages and Return Codes" section of this manual.

position	specifies the position of a field or expression listed in the SELECT clause of a SQL query statement. Fields and expressions in a SELECT list are separated by commas. Each field or expression is then referenced positionally as if the fields were numbered left to right consecutively beginning with 1. The position number is specified as a binary integer. If the user specifies a position number greater than the number of fields in the SELECT list, DESCRIBE returns an end-of-file indicator (+4) in the RETURN-CODE of the cursor data area.
dbsize	returns a binary integer specifing the maximum size of the field. If the field is defined as CHAR in the CREATE or EXPAND TABLE, the length returned is the maximum length specified for the field in that particular CREATE or EXPAND TABLE. Fields defined as NUMBER in the CREATE TABLE, and fields that contain the results of expressions always return a size of 8.
dbtype	returns a binary integer that indicates the internal data type of the field as it is stored in the database. Fields defined as CHAR in the CREATE or EXPAND TABLE are stored as variable length ASCII strings and return a value of 1. Fields defined as NUMBER are stored in ORACLE extended precision floating-point and return a value of 2. Fields that contain expression results also return a value of 2.

fsize	returns a binary integer that indicates the actual size of the data field returned by the last FETCH operation. The value returned is the actual length of the field as stored in the database before it is moved to the user buffer where padding or truncation may take place. ORACLE suppresses leading zeros on numeric data and trailing blanks on character data before storing the fields in the database. This field is valid only if the DESCRIBE is issued after a FETCH call.

DESCRIBE Examples:

FORTRAN

Request a description of the first data field in the SELECT list in the SQL query statement referenced by the cursor CURS1. Specify the position as a literal. Return the maximum size of the field and the internal data type into the program variables SIZE and TYPE.

```
CALL ODSRBN(CURS1,1,SIZE,TYPE)
```

COBOL

Request a description of the data field in the SELECT list in the SQL statement referenced by cursor CURS1 whose number is given in variable POS.

```
CALL "ODSRBN" USING CURS1,POS,SIZE,TYPE,ACTSIZE.
```

MACRO-11 ASSEMBLY LANGUAGE

Describe the second data field in a SELECT list controlled by the cursor pointed to by register 0. Return the data fields maximum size and internal data type into the program variables SIZE and TYPE.

```
ODSRBN R0,#2,#SIZE,#TYPE
```

The NAME Call

```
CALL ONAME (cursor,position,tbuf,tbufl,cbuf,cbufl)
```

The NAME call is used to retrieve the names of the tables and columns used in a SELECT clause of a SQL program query. NAME operates positionally one field at a time, referencing each field or expression in a SELECT clause as if each were numbered sequentially from left to right beginning with 1.

NAME may be used after a SQL call to determine the table name, column name, or expression string of fields to be returned. If the requested field is an expression (e.g., SAL+COMM), then no table name can be returned, and the column name is the literal expression text.

If the user specifies a position number greater than the number of fields in the SELECT list, NAME returns an end-of-file indicator in the RETURN-CODE of the cursor data area. Programs can use the end-of-file status to dynamically determine the names of fields to be returned as the result of a query for which the number of fields is unknown, as in the case of SELECT *.

cursor	specifies the name of a 64 byte data area within the user program. The cursor data area contains status information on an active SQL operation. Each cursor defines an active SQL statement within the user program. The NAME call references the cursor name to associate a data field buffer with a specific SQL statement.
position	specifies the position of a field or expression in the SELECT clause of a SQL query statement. Fields are separated by commas and numbered left to right consecutively beginning with 1. The position number is specified as a binary integer. NAME uses the position number to relate buffers to fields in the SELECT list.

tbuf	specifies the location of the data field buffer within the user program where the name of the table of which the field is a column is to be stored. If tbuf is zero, then the table name will not be stored.
tbufl	specifies the location of a binary integer which indicates the length of tbuf. If the table name to be stored is longer than tbufl, then the table name will be truncated; if it is shorter, then tbufl will point to a binary integer which is the length of the table name stored in tbuf. If tbufl is zero, then the table name will not be stored.
cbuf	specifies the location of the data field buffer within the user program where the name of the column or expression is to be stored. If cbuf is zero, then the column name will not be stored.
cbufl	specifies the location of a binary integer which indicates the length of cbuf. If the column name to be stored is longer than cbufl, then the column name will be truncated; if it is shorter, then cbufl will point to a binary integer which is the length of the column name stored in cbuf. If cbufl is zero, then the column name will not be stored.

NAME Examples:

FORTRAN

Retrieve the table and column names for the second field in the SELECT list associated with the cursor CURS1.

```
CALL ONAME(CURS1,2,TABLE,TABLEL,COL,COLL)
```

COBOL

Retrieve the table and column names in the SELECT list defined by CURS4 which has its position number specified by a variable named SELPOS.

```
CALL "ONAME" USING CURS4,SELPOS,TABLE,TABLEL,COL,COLL.
```

MACRO-11 ASSEMBLY LANGUAGE

Retrieve the table and column names for the second field in the SELECT list associated with cursor CURS2.

```
CCALL ONAME,#CURS2,#2,#TABLE,#TABLEL,#COL,#COLL
```

The DEFINE Call

```
CALL ODFINN (cursor,pos,buffer,bufl[,ftype][,rcode][,fdig])
```

The DEFINE call is used to define one output buffer for each field in a SELECT list within a SQL program query.

DEFINE specifies the location and size of a data field buffer in the user program. Define also passes the external data type of the field as defined by the user program, and optionally specifies a field return code. DEFINE defines one data field buffer at a time. Buffers are defined after the SQL call and prior to the FETCH call.

SELECT buffers are always defined positionally. Fields within the SELECT list are referenced as if they were numbered consecutively, left to right, beginning with 1. During a FETCH, ORACLE will convert each field from internal to the specified external data type and then store the fields in the defined buffers.

ORACLE provides return code information at the row level, "cursor" return code, and optionally at the field level, "field" return code. During each FETCH, ORACLE establishes a return code for each field processed. This code indicates either successful completion (return code = 0) or an exceptional condition such as: null field fetched, field truncated, etc. The field return code is stored in the rcode variable for each defined field. At the completion of each FETCH, the last non-zero "field" return code encountered is placed in the "cursor" return code.

cursor	specifies the name of a 64 byte data area within the user program. The cursor data area contains status information on an active SQL operation. Each cursor defines an active SQL statement within the user program. The DEFINE call references the cursor name to associate a data field buffer with a specific SQL statement.

pos	specifies the position of a field or expression in the SELECT clause of a SQL query statement. Fields are separated by commas and numbered left to right consecutively beginning with 1. The position number is specified as a binary integer. DEFINE uses the position number to relate buffers to fields in the SELECT list.
buffer	specifies the location of the data field buffer within the user program.
bufl	specifies the length of the buffer being defined. The buffer length is specified as a binary integer.
ftype	is a binary integer that specifies the data type that the field is to be converted to before it is moved to the user buffer. If the ftype parameter is omitted no conversion takes place. A list of external data types and type codes is contained in the section on data types in this manual.
rcode	specifies a two byte binary field defined in the user's program into which ORACLE will place a field return code. Field return codes are filled in after a FETCH operation.
fdig	The number of fractional digits (to the right of the decimal point) to be returned for datatype 7 (COBOL implied decimal). fdig is required for datatype 7, and is ignored for all others.

DEFINE Examples:

FORTRAN

Define a data field buffer for the second field in the SELECT list associated with the cursor CURS1. The data field is to be fetched into a program variable named DEPT which is defined as INTEGER*2. ORACLE is to convert the field to integer external data type (3). At the completion of each fetch ORACLE will place a return code into the program variable RC2.

```
CALL ODFINN(CURS1,2,DEPT,2,3,RC2)
```

COBOL

Define a buffer for the field in the SELECT list defined by CURS4 which has its position number specified by a variable named SELPOS. The buffer is EMPNO with its length in variable EMPNOL. The data type is specified in the variable CBL and has FRAC fractional digits. The picture for EMPNO is S99999V99 USAGE IS DISPLAY.

```
MOVE 8 TO EMPNOL.  MOVE 2 TO FRAC.  CALL "ODFINN" USING
CURS4,SELPOS,EMPNO,EMPNOL,CBL,FRAC.
```

MACRO-11 ASSEMBLY LANGUAGE

Define a data field buffer for the second field in the SELECT list associated with cursor CURS2. The data field buffer in the user program is named DNAME and has a length of 20, specified as a literal. The field is to be returned to the program in ASCII format (1). After each fetch, ORACLE will place a return code for the data field into the program variable ERR1.

```
CCALL ODFINN,#CURS2,#2,NAME,#20.,#1,#ERR1
```

The BIND Call

```
CALL OBIND (cursor,sqlvar,sqlvl,progvar,progvl[,ftype])

CALL OBINDN (cursor,sqlvarnum,progvar,progvl[,ftype])
```

The BIND call is used to dynamically modify a SQL statement after it has been passed to ORACLE in a SQL call. The statement may then be executed, modified again, re-executed, etc.

The BIND call specifies that a program value is to be assigned to a SQL substitution variable within a SQL statement.

BIND and BINDN are exactly the same except in the way they reference SQL substitution variables. BINDN references SQL substitution variables numerically. BIND references SQL substitution variables symbolically by name. The name of the variable to be bound must be specified in upper case.

At the time of the BIND, ORACLE converts the program variable from external to internal format, and then moves the data value into the SQL statement. BIND is used after the SQL call and prior to the EXECUTE call. The completion status of the BIND is indicated in the RETURN CODE area of the cursor. All return codes are specified in the "Messages and Return Codes" section of this manual.

cursor	specifies the name of a 64 byte data area within the user program. The cursor data area contains status information on an active SQL operation. Each cursor defines an active SQL statement within the user program. BIND uses the cursor name to reference a specific SQL statement.
sqlvar	specifies the character string name of a substitution variable within the SQL statement, i.e. WHERE EMPNO = &EMPLOYEE. BIND moves the program variable value into the SQL substitution variable &EMPLOYEE.

sqlvl	specifies a binary integer indicating the length of the character string specified for the sqlvar parameter. For example, &EMPLOYEE has a length of 9. If the substitution variable was specified as a literal this parameter may be omitted.
sqlvarnum	specifies a binary integer that references a SQL substitution variable within the SQL referenced by the cursor. For example, if sqlvarnum contains the value 2, it references a SQL substitution variable defined as &2.
progvar	specifies the name of a variable defined within the user program. The value within the program variable is substituted into the SQL statement at the time of the BIND.
progvl	specifies a binary integer containing the length of the program variable. A length of zero indicates a null value is to be bound to the progvar parameter.
ftype	specifies a binary integer that indicates the data type of the program variable as it is defined within the user program. ORACLE converts the program variable from external to internal format before it is bound to the SQL statement. A list of external data types and type codes is contained in a separate section of this manual.

BIND Examples:

FORTRAN

Bind the value contained in the program variable DEPT to the SQL substitution variable &DNO. DEPT is defined in the user program as INTEGER*4.

```
CALL OBIND(CURS1,'&DNO',,DEPT,4,3)
```

Bind the value contained in the program variables EMPNO and DEPT to the SQL substitution variables &1 and &2

```
 CALL OSQL(CURS1,'SELECT EMPNO FROM EMP
1 WHERE EMPNO=&1 AND DEPTNO=&2')
 CALL OBINDN(CURS1,1,EMPNO,4,3)
 CALL OBINDN(CURS1,2,DEPT,4,3)
```

COBOL

Bind the value contained in program variable EMPNO to the SQL substitution variable which is specified in variable EMPNO-NAME which has a length specified in EMPNO-N-L.

```
CALL "OBIND" USING CURS1,EMPNO-NAME
  ,EMPNO-N-L,EMPNO,EMPNO-L,INT4.
```

MACRO-11 ASSEMBLY LANGUAGE

Bind the value contained in program variable DEPT to the SQL substitution variable pointed to by ADRDNO. ADRDNO is a program variable containing the 4 byte ASCII string &DNO. The data type of the value contained in DEPT is integer with a length of 4.

```
CCALL OBIND,R0,#ADRDNO,#4,EPT,#4,#3
```

EXECUTE Call

```
CALL OEXEC (cursor)
```

The EXECUTE call causes the SQL statement currently associated with the cursor to be processed.

If the SQL statement is a data manipulation, data definition, or data control statement, the entire SQL function is performed at this time; the RETURN CODE is set and a count of the rows processed by the statement is placed in ROW COUNT field of the cursor data area. If the SQL statement is a query, the user program must explicitly request each row of the result using the FETCH call.

cursor	specifies the name of a 64 byte data area within the user program. The cursor data area contains status information on an active SQL operation. Each cursor defines an active SQL statement within the user program. The EXECUTE call executes the SQL statement attached to the cursor.

EXECUTE Examples:

FORTRAN

Execute the SQL statement which was passed to ORACLE using CURS1.

```
CALL OEXEC(CURS1)
```

COBOL

Execute the SQL statement which was passed to ORACLE using CURS1.

```
CALL "OEXEC" USING CURS1.
```

MACRO-11 ASSEMBLY LANGUAGE

Execute the SQL statement which was passed to ORACLE using the cursor pointed to by register zero.

```
CCALL OEXEC,R0
```

The FETCH Call

```
CALL OFETCH (cursor)
```

The FETCH call returns rows of a query result to the user program, one row at a time. Each field of the query is placed into a buffer identified by a previously executed DEFINE call. Fields that are requested by the user program in character string format are left justified and padded with trailing blanks. Character strings that are too large for the field buffer are truncated and the ORACLE field return code is set to +3.

If null values are encountered in any field of the fetch, the ORACLE field return code for that field is set to +2 and the user buffer remains unchanged. To determine which specific fields are null or have been truncated, the user program must have specified field return codes in the DEFINE buffer calls, or a DESCRIBE call may be issued. If multiple non-zero field return codes are encountered in a single FETCH, the cursor return code will contain the last non-zero field return code.

After the last row of the query result has been returned to the user program, the next fetch will return an end-of-file return code of +4. After each FETCH the cursor row count is updated. When end-of-file has been reached, the row count will contain the total number of rows found by the query.

cursor	specifies the name of a 64 byte data area within the user program. The cursor data area contains status information on an active SQL operation. Each cursor defines an active SQL statement within the user program. The cursor maintains position on a set of rows that satisfy a query as those rows are retrieved, one at a time by the FETCH call.

FETCH Examples:

FORTRAN

Fetch a row for the SELECT statement passed to ORACLE using CURS1. CALL OFETCH(CURS2)

COBOL

Fetch a row for the SELECT statement passed to ORACLE using CURS1.

```
CALL "OFETCH" USING CURS1.
```

MACRO-11 ASSEMBLY LANGUAGE

Fetch a row for the SELECT statement passed to ORACLE using CURS2.

```
CCALL OFETCH,#CURS2
```

The CLOSE Call

```
CALL OCLOSE (cursor)
```

The CLOSE call disconnects a cursor from ORACLE and frees all resources obtained by the OPEN, SQL, and EXECUTE functions using this cursor. If the CLOSE fails, the RETURN CODE area of the cursor contains the status indicator. All of the return codes are listed in the "Messages and Return Codes" section of this manual.

cursor	specifies the name of a 64 byte data area within the user program. The cursor data area contains status information on an active SQL operation. Each cursor defines an active SQL statement within the user program. CLOSE disconnects the cursor from ORACLE.

CLOSE Examples:

FORTRAN

Close the cursor CURS1.

```
CALL OCLOSE(CURS1)
```

COBOL

Close the cursor CURS1.

```
CALL "OCLOSE" USING CURS1.
```

MACRO-11 ASSEMBLY LANGUAGE

Close the cursor pointed to by register zero.

```
OCLOSE R0
```

The LOGOFF Call

```
CALL OLOGOF (lda)
```

The LOGOFF call disconnects a program from ORACLE and frees all ORACLE resources owned by this program. If the LOGOFF fails, the reason is indicated in the first two bytes of the Logon Data Area (LDA). A complete list of return codes is given in the "Messages and Return Codes" section of this manual.

lda	specifies the name of the Logon Data Area specified in the LOGON call.

LOGOFF Examples:

FORTRAN

Log off from ORACLE.

```
CALL OLOGOF(LDAREA)
```

COBOL

Log off from ORACLE

```
CALL "OLOGOF" USING LDAREA.
```

MACRO-11 ASSEMBLY LANGUAGE

Log off from ORACLE.

```
CCALL OLOGOF,R2
```

SQL SUBSTITUTION VARIABLES

When SQL is used within a host programming language, substitution variables may be used within the SQL statement. SQL substitution variables allow user programs to dynamically modify SQL statements, and then execute those modified statements.

SQL substitution variables are identified by an ampersand. Substitution variables may be used anywhere in a SQL statement that a constant may be used. For example, in FORTRAN:

```
 CALL OSQL (CUR1,'SELECT ENAME,SAL
1 FROM EMP
2 WHERE DEPTNO = &DEPT;')
```

The BIND call is used to substitute values into a SQL substitution variable.

```
 CALL OBIND (CUR1,'&DEPT',5,DEPT,2,3)
```

DEPT is a variable defined in the user program as a 2 byte fixed point number.

SQL substitution variables may also be used to BIND values in INSERT and SET statements.

```
 CALL OSQL (CUR2,'INSERT INTO DEPT:
1<&A,&B,&C,NULL>;')
```

Null values may be inserted into the database by specifying NULL in the INSERT list or by binding a value with a zero length to a SQL substitution variable.

```
CALL OBIND (CUR2,&C,2,LOC,0,1)
```

O R A C L E

HOST LANGUAGE INTERFACE

DATA TYPES

ORACLE performs data conversions for most data types provided by the supported languages. On retrieval (SELECT) operations, ORACLE will convert from the internal format of the data as stored in the database, to an external format as defined by the user program. On storage (INSERT and UPDATE) operations, ORACLE will convert from external to internal data types.

The user specifies the external data type for SELECT operations with the DEFINE call. The user specifies external data types for INSERT and UPDATE operations with the BIND call.

Internally, ORACLE stores characters in ASCII and numbers in a variable length extended precision (maximum 22 bytes) floating point format.

If the user does not want ORACLE to do any conversion on numeric data, the data may be defined as a character string for both the internal and external data type.

The following is a list of the external data types supported by ORACLE:

DATA TYPE	CODE	FORTRAN	COBOL
Varying-length character string	01	LOGICAL*1	PIC X...X
ORACLE internal numeric	02	N/A	N/A
8 bit fixed point	03	LOGICAL*1	N/A
16 Bit fixed point	03	INTEGER*2	PIC S9(4) COMP
32 Bit fixed point	03	INTEGER*4	PIC S9(9) COMP
32 Bit floating point	04	REAL*4	N/A
64 Bit floating point	04	REAL*8	N/A
Null terminated string	05	LOGICAL*1	PIC X...X
Raw data	06	LOGICAL*1	PIC X...X
COBOL implied decimal	07	N/A	PIC S9V9

O R A C L E

HOST LANGUAGE INTERFACE

DATA TYPE DESCRIPTIONS

The use of each of the ORACLE external data types is described below.

DATA TYPE	DESCRIPTION
01	The varying length character string format is a string of ASCII characters whose length is determined by a length field. Trailing blank characters are discarded on input. ORACLE pads the string with trailing blanks on output. If the length specification is missing on input, the string length is determined by scanning the string until a null character (a zero byte) is encountered. The length is required on output. An all blank field, or one whose length is specified as zero on input is treated as a NULL by ORACLE if the internal datatype is CHAR; if the internal type is NUMBER, then an all blank field is converted to zero. A zero length output length specification is invalid. When the ORACLE internal data type is NUMBER, input character strings are converted to internal numeric format until an invalid numeric character is encountered, e.g., '1234.45bcd' is converted to 1234.45, and the 'bcd' is ignored without any error indication. Output to an ASCII buffer from an internal numeric datatype to a character string which contains the ascii character representation of the internal number. The field width determines the precision. The number will be converted to scientific notation if required by the field size., e.g., if the number is 123456789 and the field width is 6, the output string will be '1.2E08'.
02	See the section on internal numeric format.

03	The integer number format is used to process numbers which have no fractional parts. The integer number is a signed binary number of one, two, or four bytes. The significance order is determined by the host language being used. The length specification is required for input and output. If the number being output from ORACLE is not integral, the fractional part is discarded. Integer numbers may be used only with internal numeric columns.
04	The floating number format is used to process numbers which have fractional parts, or which exceed the capacity of integer number format. The number is represented using the computer's floating point format with a length of either 4 or 8 bytes (REAL*4 or REAL*8). The length specification is required for input and output. Since the internal numeric format is decimal based, some precision may be lost during the conversion from the computer's binary floating point format to and from ORACLE's decimal format.
05	The null terminated string format is exactly like the varying length character string format, except that the string is always terminated by one byte of zero (the NULL character). On input, the string length is ignored and the string scanned for the null. On output, the null is placed after the last character returned. If the string exceeds the field length specified, the string is truncated and the last character position of the buffer contains the null. Trailing blanks are discarded on input. A zero length string (null in the first position or an all blank field) is treated as NULL by ORACLE.
06	The raw data format is used for binary data. The contents of the buffer are not interpreted in any way by ORACLE for either input or output. The length is required for input and output. On output, only the number of characters stored in the database are returned; the remainder of the output buffer is not modified. The number of characters actually returned may be determined using the DESCRIBE call.

07 The COBOL implied decimal data type is used to return non-integral numbers from ORACLE to a COBOL data type which is suitable for calculation. The COBOL data area must be a signed numeric display field with an implied decimal point. The number of digits to the right of the decimal point is specified with the DEFINE call. The value returned may be used as is for COBOL calculations, or may be moved to a computational field prior to calculations. The number will never be converted to scientific notation. If the number to be returned loses significant digits during the conversion, ORACLE fills the buffer with "*" characters.

DATA CONVERSIONS

The following table specifies the data conversions supported by ORACLE.

SYSTEM FORMAT / USER FORMAT	TO ORACLE		FROM ORACLE	
	CHAR	NUMBR	CHAR	NUMBR
ASCII	YES	YES	YES	YES
INTERNAL NUMERIC	YES	YES	YES	YES
8 BIT FIXED PT.	-	YES	-	YES
16 BIT FIXED PT.	-	YES	-	YES
32 BIT FIXED PT.	-	YES	-	YES
32 BIT FLOATING	-	YES	-	YES
64 BIT FLOATING	-	YES	-	YES
NULL TERM. STR.	YES	YES	YES	YES
RAW DATA	YES	YES	YES	YES
COBOL IMPLIED	NO	NO	NO	YES*

* NOTE: The COBOL 'number' datatype must have a picture of the following form:
PICTURE S9(N)V9(N) USAGE DISPLAY SIGN
LEADING SEPARATE.

O R A C L E

HOST LANGUAGE INTERFACE

INTERNAL NUMERIC FORMAT

Database fields defined as NUMBER in the CREATE TABLE are stored in the database in ORACLE's variable length extended precision floating point format.

ORACLE floating point numbers vary in length and occupy from 1 byte to 22 bytes of real storage. The ORACLE internal numeric format is depicted below:

```
+-----------------------------------------+
|S|  EXP  |          DIGITS               |
|1| 2 - 8 | 9                         176 |
+-----------------------------------------+
```

The number consists of a sign bit (0 is negative, 1 is positive), a 7 bit exponent and up to 21 bytes of significant digits.

The exponent represents the number of digit positions to shift the radix point (which is assumed to be to the left of the leftmost digit). The exponent is expressed in excess 64 format, so that an exponent of 64 indicates a 0 shift, 63 a shift to the left of one digit, 65 a shift to the right of one digit, and so on.

Each digit is stored in one byte and is a base 100 digit represented as a binary number from 0 to 99. Each shift of the radix point changes the magnitude of the number by a power of 100. Trailing zero digits are discarded. Negative numbers are indicated by a sign bit of zero and the digits are stored as the 100's complement of the number.

The ORACLE numeric format can represent numbers ranging from 10 ** -128 to (10 ** 128) - 1 with up to 42 digits of precision.

O R A C L E

HOST LANGUAGE INTERFACE

CURSOR DATA AREA

The cursor is a 64 byte data area defined within a user program. A cursor is identified to ORACLE in an OPEN call. The cursor data area contains status information on an active SQL operation. Each cursor defines an active SQL statement within a user program. All ORACLE calls that reference a SQL statement reference it by cursor name. The cursor format is depicted below:

```
+-----------------------------------------------------+
| 0                          | 2                       |
|       RETURN CODE          |      FUNCTION TYPE      |
|-----------------------------------------------------|
| 4                                                   |
|              ROWS PROCESSED COUNT                   |
|-----------------------------------------------------|
| 8                          |11          |10         |
|   PARSE ERROR OFFSET       |  FILLER    | FUNC CODE |
|-----------------------------------------------------|
|12                                                   |
|                  ORACLE  SYSYEM                     |
|                                                     |
|                  PARAMETER AREA                     |
|                                              64     |
+-----------------------------------------------------+
```

RETURN CODE	contains a two byte binary number that indicates the completion code for the specified operation. Zero indicates a successful result. A positive return code indicates a successful result with an exceptional condition. A negative return code indicates an error was encountered in attempting to perform the specified operation. See ORACLE Messages and Codes for a complete list of return codes.

FUNCTION CODE	contains an operation code indicating the type of ORACLE function requested. The function codes are: 02 - SQL 04 - EXECUTE 06 - BIND 08 - DEFINE 10 - DESCRIBE 12 - FETCH 14 - OPEN 16 - CLOSE 18 - BINBN
ROWS PROCESSED COUNT	contains a four byte binary number indicating the count of the number of rows processed by a SQL operation. The count will contain the number of rows inserted, updated, or deleted by a data manipulation statement, or the number of rows fetched in a query statement. This field is valid only after an EXECUTE or FETCH operation.
PARSE ERROR OFFSET	contains a two byte binary number indicating the offset in characters into the SQL text where the parse error occured.

O R A C L E

HOST PROGRAM INTERFACE

PROGRAM INTERFACE DATA AREAS

ORACLE allows a single program to have multiple cursors open at the same time.

To optimize program performance, it is useful to have an understanding of the communication between ORACLE and a user program.

The following is a diagram of a user program named UPDT3 with two open cursors.

```
         USER PROGRAM                          ORACLE
 +--------------------------+      +--------------------------+
 |          UPDT3           |      | COMMUNICATION REGION     |
 |                          |      | +---------+ +---------+  |
 |      +-----------+       |      | | UPDT3   | | unused  |  |
 |      | LOGON LDA |       |      | |  LCA    | |  LCA    |  |
 |      +-----------+       |      | |+--+ +--+| |         |  |
 |                          |      | ||C1| |C2|| |         |  |
 | +-------+    +-------+   |      | |+--+ +--+| |         |  |
 | |OPEN C1|    |OPEN C2|   |      | +---------+ +---------+  |
 | +-------+    +-------+   |      |==========================|
 |                          |      |                          |
 |  SELECT       UPDATE     |      |                          |
 |  FROM         SET        |      |       O R A C L E        |
 |  WHERE        WHERE      |      |                          |
 |                          |      |                          |
 |                          |      |                          |
 +--------------------------+      +--------------------------+
```

When program UPDT3 issues the LOGON call, ORACLE allocates an Logon Control Area (LCA) for UPDT3 in the ORACLE Communication Region. ORACLE connects this LCA to the Logon Data Area (LDA) defined within UPDT3. ORACLE will allocate one and only one LCA for each terminal and program currently logged on to ORACLE.

When program UPDT3 issues an OPEN call, ORACLE allocates a SQL Work Area (SWA) for UPDT3 in the ORACLE Communication Region. ORACLE connects the SWA to the Cursor (C1) defined within the UPDT3.

When program UPDT3 issues a second OPEN call, ORACLE allocates a second SWA and connects it to UPDT3's second cursor (C2). ORACLE will allocate one SWA for every open cursor.

If a program opens multiple cursors causing ORACLE to allocate multiple SWA's to that program, some of that program's SWA's may be swapped to disk. However, ORACLE will maintain at least one SWA in main memory for each program or terminal logged on to ORACLE.

The default number of SWA's to be maintained in memory for a given program is one, unless the user specifies a different number in the optional "areacount" parameter in the LOGON call.

The default size of each SWA is 3K bytes. The user can override the default by specifying the "areasize" parameter of the OPEN call. The SWA must be large enough to contain the compiled SQL statement plus one row of data of the table or view being processed.

```
*****************************************
*                                       *
*            F O R T R A N              *
*                                       *
*   E X A M P L E    P R O G R A M S    *
*                                       *
*****************************************
```

Table of Contents

```
      PROGRAM SAMPL1
C
C     SAMPL1 IS WRITTEN FOR THE VAX FORTRAN COMPILERS. NOTE
C     THE %REF IS USED WHENEVER A LITERAL STRING IS PASSED
C     TO ORACLE.
C
C     SAMPL1 IS A SIMPLE EXAMPLE PROGRAM WHICH ADDS NEW EMPLOYEES
C     ROWS TO THE PERSONNEL DATA BASE.  VERY LITTLE CHECKING IS
C     DONE TO INSURE THE INTEGRITY OF THE DATA BASE.  THE PROGRAM
C     QUERIES THE USER (VIA DEVICE 5) FOR DATA AS FOLLOWS:
C
C        Enter employee number:
C        Enter employee name  :
C        Enter employee job   :
C        Enter employee salary:
C        Enter employee dept  :
C
C     THE NEW EMPLOYEE ROW IS INSERTED AND THE DEPARTMENT
C     TABLE IS UPDATED TO INCREASE THE EMPLOYEE COUNT.  IF
C     THE EMPLOYEE NUMBER IS ENTERED AS '0', THEN THE
C     PROGRAM TERMINATES.
C
      IMPLICIT INTEGER*2 (A-Z)
      LOGICAL*1 ENAME(10),JOB(9)
      INTEGER*2 LDA(32),CUR1(32),CUR2(32)
C
C     LOGON TO ORACLE
C
      CALL OLOGON(LDA)
      IF (LDA(1).NE.0) GO TO 10000
C
C     OPEN TWO CURSORS FOR THE PERSONNEL DATA BASE
C

      CALL OOPEN(CUR1,LDA,
     X   %REF('PERSONNEL'),,,%REF('QA/TEST'))
      IF (CUR1(1).NE.0) GO TO 10000

      CALL OOPEN(CUR2,LDA,
     X   %REF('PERSONNEL'),,,%REF('QA/TEST'))
      IF (CUR2(1).NE.0) GO TO 10000
C
C     PASS THE SQL STATEMENTS TO ORACLE
C
      CALL OSQL(CUR1,
     1   %REF('INSERT INTO EMP(EMPNO,ENAME,JOB,SAL,DEPTNO)
     2   <&EMPNO,&ENAME,&JOB,&SAL,&DEPTNO>;'))
      IF (CUR1(1).NE.0) GO TO 10000
```

```
C
C     NOTE THAT THE 'NULLF' FUNCTION FORCES THE EMPCNT TO ZERO
C     IF IT IS NULL
C
      CALL OSQL(CUR2,
     1   %REF('UPDATE DEPT SET EMPCNT=NULLF(EMPCNT,0)+1
     2   WHERE DEPTNO=&DEPTNO;'))
      IF (CUR2(1).NE.0) GO TO 10000

C
C READ THE USER'S INPUT FROM DEVICE 5 (NORMALLY, THE TERMINAL)
C
10    WRITE(5,100)
100   FORMAT('$Enter employee number: ')
      READ (5,110,END=5000,ERR=10)EMPNO
110   FORMAT(I5)
      IF (EMPNO.EQ.0) GO TO 5000
      WRITE (5,120)
120   FORMAT('$Enter employee name  : ')
      READ (5,130)ENAME
130   FORMAT(10A1)
      WRITE (5,140)
140   FORMAT('$Enter employee job   : ')
      READ (5,150)JOB
150   FORMAT(9A1)
153   WRITE (5,155)
155   FORMAT('$Enter employee salary: ')
      READ (5,110,ERR=153)SAL
158   WRITE (5,160)
160   FORMAT('$Enter employee dept  : ')
      READ (5,110,ERR=158)DEPT
C
C     BIND ALL SQL SUBSTITUTION VARIABLE VALUES.
C     IF ANY ERRORS OCCUR, PRINT AN ERROR MESSAGE,
C     BUT CONTINUE.
C
      CALL OBIND(CUR1,%REF('&EMPNO'),,EMPNO,2,3)
      IF (CUR1(1).NE.0) GO TO 1000
      CALL OBIND(CUR1,%REF('&ENAME'),,ENAME,10,1)
      IF (CUR1(1).NE.0) GO TO 1000
      CALL OBIND(CUR1,%REF('&JOB'),,JOB,9,1)
      IF (CUR1(1).NE.0) GO TO 1000
      CALL OBIND(CUR1,%REF('&SAL'),,SAL,2,3)
      IF (CUR1(1).NE.0) GO TO 1000
      CALL OBIND(CUR1,%REF('&DEPTNO'),,DEPT,2,3)
      IF (CUR1(1).NE.0) GO TO 1000
      CALL OBIND(CUR2,%REF('&DEPTNO'),,DEPT,2,3)
      IF (CUR2(1).NE.0) GO TO 1000
```

```
C
C     EXECUTE THE SQL STATEMENTS: CUR1 INSERTS A ROW INTO THE
C     'EMP' TABLE.
C
      CALL OEXEC(CUR1)
      IF (CUR1(1).NE.0) GO TO 1000
C
C     ...CUR2 UPDATES THE 'EMPCNT' COLUMN OF THE 'DEPT' TABLE.
C
      CALL OEXEC(CUR2)
      IF (CUR2(1).NE.0) GO TO 1000
      GO TO 10
1000  CALL ERRRPT(LDA,CUR1,CUR2)
      GO TO 10
C
C     CLOSE THE TWO CURSORS
C
5000  CALL OCLOSE(CUR1)
      CALL OCLOSE(CUR2)
C
C     LOGOFF FROM ORACLE
C
      CALL OLOGOF(LDA)
      STOP 'END OF SAMPL1'
10000 CALL ERRRPT(LDA,CUR1,CUR2)
      GO TO 5000
      END

      SUBROUTINE ERRRPT(LDA,C1,C2)
C
C     ERRRPT PRINTS THE CURSOR NUMBER, THE ERROR CODE, AND THE
C     ORACLE FUNCTION CODE.  IF THE LDA CONTAINS AN ERROR CODE,
C     A LOGON ERROR IS ASSUMED
C
C     LDA IS THE LOGON DATA AREA ARRAY
C     C1  IS THE FIRST CURSOR ARRAY
C     C2  IS THE SECOND CURSOR ARRAY
```

```
C
      INTEGER*2 LDA(32),C1(32),C2(32)

      IF (LDA(1).EQ.0) GO TO 100
      WRITE (5,10)LDA(1)
10    FORMAT('0Logon error: ',I5)
      GO TO 500
100   IF (C1(1).EQ.0) GO TO 200
      WRITE (5,110) 1,C1(1),C1(6)
110   FORMAT('0ORACLE error on cursor ',I1,
     1 ': CODE IS ',I5,', OP IS ',I5)
      GO TO 500
200   IF (C2(1).EQ.0) GO TO 300
      WRITE (5,110),2,C2(1),C2(6)
      GO TO 500
300   WRITE (5,310)
310   FORMAT('0Unknown ORACLE error')
500   RETURN
      END
      PROGRAM SAMPL2
C
C     SAMPL2 IS A SIMPLE EXAMPLE PROGRAM WHICH ADDS NEW EMPLOYEE
C     ROWS TO THE PERSONNEL DATA BASE.  SOME CHECKING
C     IS DONE TO INSURE THE INTEGRITY OF THE DATA BASE.
C     THE PROGRAM QUERIES THE USER (VIA DEVICE 5) FOR DATA AS FOL
C
C        Enter employee number:
C        Enter employee name  :
C        Enter employee job   :
C        Enter employee salary:
C        Enter employee dept  :
C
C     THE NEW EMPLOYEE ROW IS INSERTED AND THE DEPARTMENT
C     TABLE IS UPDATED TO INCREASE THE EMPLOYEE COUNT.  IF
C     THE EMPLOYEE NUMBER IS ENTERED AS '0', THEN THE
C     PROGRAM TERMINATES.
C
      IMPLICIT INTEGER*2 (A-Z)
      LOGICAL*1 ENAME(11),JOB(10),DEPT(20)
      INTEGER*2 CURS(32,4)
C
C     LOGON TO ORACLE
C
      CALL OLOGON(CURS(1,1))
      IF (CURS(1,1).NE.0) GO TO 10000
C
C     OPEN THREE CURSORS FOR THE PERSONNEL DATA BASE
C
      CALL OOPEN(CURS(1,2),CURS(1,1),'PERSONNEL',,,'QA/TEST')
      IF (CURS(1,2).NE.0) GO TO 10000

      CALL OOPEN(CURS(1,3),CURS(1,1),'PERSONNEL',,,'QA/TEST')
      IF (CURS(1,3).NE.0) GO TO 10000
      CALL OOPEN(CURS(1,4),CURS(1,1),'PERSONNEL',,,'QA/TEST')
      IF (CURS(1,4).NE.0) GO TO 10000
C
C     PASS THE SQL STATEMENTS TO ORACLE
C
      CALL OSQL(CURS(1,2),'INSERT INTO EMP(EMPNO,ENAME,JOB,SAL,DE
     1   <&EMPNO,&ENAME,&JOB,&SAL,&DEPTNO>;')
      IF (CURS(1,2).NE.0) GO TO 10000
C
C     NOTE THAT THE 'NULLF' FUNCTION FORCES THE EMPCNT TO ZERO
C     IF IT IS NULL
```

```
C
      CALL OSQL(CURS(1,3),'UPDATE DEPT SET EMPCNT=NULLF(EMPCNT,0)
     1   WHERE DEPTNO=&DEPTNO;')
      IF (CURS(1,3).NE.0) GO TO 10000

      CALL OSQL(CURS(1,4),'SELECT DNAME FROM DEPT WHERE
     1   DEPTNO=&DEPTNO;')
      IF (CURS(1,4).NE.0) GO TO 10000
C
C     DEFINE A BUFFER TO RECEIVE THE DEPARTMENT NAME FOR ORACLE
C
      CALL ODFINN(CURS(1,4),1,DEPT,20,5)
      IF (CURS(1,4).NE.0) GO TO 10000

C
C READ THE USER'S INPUT FROM DEVICE 5 (NORMALLY, THE TERMINAL)
C
10    WRITE(5,100)
100   FORMAT('$Enter employee number: ')
      READ (5,110,END=5000,ERR=10)EMPNO
110   FORMAT(I5)
      IF (EMPNO.EQ.0) GO TO 5000
      WRITE (5,120)
120   FORMAT('$Enter employee name  : ')
      DO 125 I=1,11
125   ENAME(I)=0
      READ (5,130)ENAME
130   FORMAT(11A1)
      WRITE (5,140)
140   FORMAT('$Enter employee job   : ')
      DO 145 I=1,10
145   JOB(I)=0
      READ (5,150)JOB
150   FORMAT(10A1)
153   WRITE (5,155)
155   FORMAT('$Enter employee salary: ')
      READ (5,110,ERR=153)SAL
158   WRITE (5,160)
160   FORMAT('$Enter employee dept  : ')
      READ (5,110,ERR=158)DEPTNO
C
C     BIND ALL SQL SUBSTITUTION VARIABLE VALUES
C     IF ANY ERRORS OCCUR, PRINT AN ERROR MESSAGE,
C     BUT CONTINUE.
```

```
C
      CALL OBIND(CURS(1,2),'&EMPNO',,EMPNO,2,3)
      IF (CURS(1,2).NE.0) GO TO 1000
      CALL OBIND(CURS(1,2),'&ENAME',,ENAME,10,5)
      IF (CURS(1,2).NE.0) GO TO 1000
      CALL OBIND(CURS(1,2),'&JOB',,JOB,9,5)
      IF (CURS(1,2).NE.0) GO TO 1000
      CALL OBIND(CURS(1,2),'&SAL',,SAL,2,3)
      IF (CURS(1,2).NE.0) GO TO 1000
      CALL OBIND(CURS(1,2),'&DEPTNO',,DEPTNO,2,3)
      IF (CURS(1,2).NE.0) GO TO 1000
      CALL OBIND(CURS(1,3),'&DEPTNO',,DEPTNO,2,3)
      IF (CURS(1,3).NE.0) GO TO 1000
      CALL OBIND(CURS(1,4),'&DEPTNO',,DEPTNO,2,3)
      IF (CURS(1,4).NE.0) GO TO 1000
C
C     EXECUTE THE SQL STATEMENTS.  CURSOR NUMBER 3 SELECTS
C     'DNAME' FROM THE 'DEPT' TABLE.  IF THERE IS NO SUCH DEPARTM
C     (AS DETECTED BY A RETURN CODE OF 4 TO THE FETCH CALL), THEN
C     AN ERROR MESSAGE IS DISPLAYED.
      CALL OEXEC(CURS(1,4))
      IF (CURS(1,4).NE.0) GO TO 1000
      DO 450 I=1,20
450   DEPT(I) = 0
      CALL OFETCH(CURS(1,4))
      IF (CURS(1,4).EQ.0) GO TO 500
      IF (CURS(1,4).NE.4) GO TO 1000
      WRITE (5,400)
400   FORMAT('0No such department number')
      GO TO 10
C
C     CURSOR NUMBER 1 INSERTS A NEW ROW INTO THE 'EMP' TABLE.
C
500   CALL OEXEC(CURS(1,2))
      IF (CURS(1,2).NE.0) GO TO 1000
C
C     CURSOR NUMBER 2 UPDATE THE 'EMPCNT' COLUMNS IN THE 'DEPT' T
C
      CALL OEXEC(CURS(1,3))
      IF (CURS(1,3).NE.0) GO TO 1000
      WRITE (5,600)ENAME,DEPT
600   FORMAT(' ',11A1,' added to the ',20A1,' department')
      GO TO 10
1000  CALL ERRRPT(CURS(1,1),4)
      GO TO 10
```

```
C
C CLOSE THE THREE CURSORS
C
5000  CALL OCLOSE(CURS(1,2))
      CALL OCLOSE(CURS(1,3))
      CALL OCLOSE(CURS(1,4))
C
C     LOGOFF FROM ORACLE
C
      CALL OLOGOF(CURS(1,1))
      STOP 'END OF SAMPL2'
10000 CALL ERRRPT(CURS(1,1),4)
      GO TO 5000
      END

      SUBROUTINE ERRRPT(CURS,N)
C
C     ERRRPT PRINTS THE CURSOR NUMBER, THE ERROR CODE, AND THE
C     ORACLE FUNCTION CODE.  IF THE LDA CONTAINS AN ERROR CODE,
C     A LOGON ERROR IS ASSUMED
C
C     CURS IS THE CURSOR ARRAY
C     N IS THE NUMBER OF CURSORS (INCLUDING THE LDA) IN THE ARRAY
C
      INTEGER*2 CURS(32,N)

      IF (CURS(1,1).EQ.0) GO TO 100
      WRITE (5,10)CURS(1,1)
10    FORMAT('0Logon error: ',I5)
      GO TO 500
100   DO 110 I=1,N
      IF (CURS(1,I).NE.0) GO TO 150
110   CONTINUE
150   IF (I.LE.N) GO TO 300
      WRITE (5,200)
200   FORMAT('0Unknown ORACLE error')
      GO TO 500
300   WRITE (5,400)I,CURS(1,I),CURS(6,I)
400   FORMAT('0ORACLE error on cursor ',I1,
     1   ': code is ',I5,', op is ',I5)
500   RETURN
      END
      PROGRAM SAMPL3
C
C     SAMPL3 IS A SIMPLE EXAMPLE PROGRAM WHICH ADDS NEW EMPLOYEE
C     ROWS TO THE PERSONNEL DATA BASE.  CHECKING
C     IS DONE TO INSURE THE INTEGRITY OF THE DATA BASE.
C     THE EMPLOYEE NUMBERS ARE AUTOMATICALLY SELECTED USING
C     THE CURRENT MAXIMUM EMPLOYEE NUMBER AS THE START.
C     IF ANY EMPLOYEE NUMBER IS A DUPLICATE, IT IS SKIPPED.
C     THE PROGRAM QUERIES THE USER (VIA DEVICE 5) FOR DATA AS FOL
C
C        Enter employee name  :
C        Enter employee job   :
C        Enter employee salary:
C        Enter employee dept  :
C
C     THE NEW EMPLOYEE ROW IS INSERTED AND THE DEPARTMENT
C     TABLE IS UPDATED TO INCREASE THE EMPLOYEE COUNT.  IF
C     THE EMPLOYEE NAME IS NOT ENTERED, THEN THE PROGRAM
C     TERMINATES.
C
C     IF THE ROW IS SUCCESSFULLY INSERTED, THE FOLLOWING
C     IS PRINTED:
C
C     ENAME added to DNAME department as employee # NNNNN
C
      IMPLICIT INTEGER*4 (A-Z)
C
C     THE MAXIMUM LENGTHS OF THE 'ENAME', 'JOB', AND 'DNAME'
C     COLUMNS WILL BE DETERMINED BY AN ORACLE CALL.  THE
C     PROGRAM ASSUMES THAT THE SUM OF THE LENGTHS WILL BE
C     LESS THAN 100 BYTES.  THE COLUMNS WILL ALL BE STORED
C     IN ONE ARRAY -- STRNGS.
C

      LOGICAL*1 STRNGS(100),ENMFMT(6),JOBFMT(6),DEPFMT(70)
      INTEGER*2 CURS(32,6)
C
C     LOGON TO ORACLE
C
      CALL OLOGON(CURS(1,1))
      IF (CURS(1,1).NE.0) GO TO 10000
C
C     OPEN FIVE CURSORS FOR THE PERSONNEL DATA BASE
```

```
C
      CALL OOPEN(CURS(1,2),CURS(1,1),'PERSONNEL',,,'QA/TEST')
      IF (CURS(1,2).NE.0) GO TO 10000

      CALL OOPEN(CURS(1,3),CURS(1,1),'PERSONNEL',,,'QA/TEST')
      IF (CURS(1,3).NE.0) GO TO 10000

      CALL OOPEN(CURS(1,4),CURS(1,1),'PERSONNEL',,,'QA/TEST')
      IF (CURS(1,4).NE.0) GO TO 10000

      CALL OOPEN(CURS(1,5),CURS(1,1),'PERSONNEL',,,'QA/TEST')
      IF (CURS(1,5).NE.0) GO TO 10000

      CALL OOPEN(CURS(1,6),CURS(1,1),'PERSONNEL',,,'QA/TEST')
      IF (CURS(1,6).NE.0) GO TO 10000
C
C     RETRIEVE THE CURRENT MAXIMUM EMPLOYEE NUMBER
C
C     PASS THE SQL STATEMENT TO ORACLE
C
      CALL OSQL(CURS(1,2),'SELECT MAX(EMPNO) + 10 FROM EMP;')
      IF (CURS(1,2).NE.0) GO TO 10000
C
C     DEFINE A BUFFER TO RECEIVE THE MAX(EMPNO)+10 FROM ORACLE
C
      CALL ODFINN(CURS(1,2),1,EMPNO,4,3)
      IF (CURS(1,2).NE.0) GO TO 10000
C
C     EXECUTE THE SQL STATEMENT
C
      CALL OEXEC(CURS(1,2))
      IF (CURS(1,2).NE.0) GO TO 10000
C
C     FETCH THE DATA FROM ORACLE INTO THE DEFINED BUFFER
C
      CALL OFETCH(CURS(1,2))
      IF (CURS(1,2).EQ.0) GO TO 50
      IF (CURS(1,2).NE.4) GO TO 10000
C
C     CURSOR RETURN CODE 4 MEANS THAT NO ROW SATISFIED THE QUERY,
C     SO GENERATE THE FIRST EMPNO
C
      EMPNO=10
50    CONTINUE
C
C     DETERMINE THE MAX LENGTH OF THE EMPLOYEE NAME AND JOB TITLE
C
C     PASS THE SQL STATEMENT TO ORACLE.  IT WILL NOT BE EXECUTED.
```

```
C
      CALL OSQL(CURS(1,2),'SELECT ENAME,JOB FROM EMP;')
      IF (CURS(1,2).NE.0) GO TO 10000
C
C     CALL ORACLE TO DESCRIBE THE TWO FIELDS SPECIFIED IN THE ABO
C     SQL STATEMENT.  WE ARE ONLY CONCERNED ABOUT THE LENGTH.
C
      CALL ODSRBN(CURS(1,2),1,ENAMEL)
      IF (CURS(1,2).NE.0) GO TO 10000

      CALL ODSRBN(CURS(1,2),2,JOBL)
      IF (CURS(1,2).NE.0) GO TO 10000
C
C     PUT THE LENGTHS INTO THE FORMATS SO THAT THE ENAME AND JOB
C     COLUMNS WILL BE PRINTED CORRECTLY.
C
      ENCODE(6,60,ENMFMT)ENAMEL
60    FORMAT('(',I2,'A1)')
      ENCODE(6,60,JOBFMT)JOBL
C
C     PASS THE SQL STATEMENTS TO ORACLE
C
      CALL OSQL(CURS(1,2),'INSERT INTO EMP(EMPNO,ENAME,JOB,SAL,DE
     1   <&EMPNO,&ENAME,&JOB,&SAL,&DEPTNO>;')
      IF (CURS(1,2).NE.0) GO TO 10000

      CALL OSQL(CURS(1,3),'UPDATE DEPT SET EMPCNT=NULLF(EMPCNT,0)
     1   WHERE DEPTNO=&DEPTNO;')
      IF (CURS(1,3).NE.0) GO TO 10000
      CALL OSQL(CURS(1,4),'SELECT DNAME FROM DEPT WHERE
     1   DEPTNO=&DEPTNO;')
      IF (CURS(1,4).NE.0) GO TO 10000

      CALL OSQL(CURS(1,5),'BEGIN TRANSACTION 1 ON TABLE EMP,DEPT
     1   UPDATE;')
      IF (CURS(1,5).NE.0) GO TO 10000

      CALL OSQL(CURS(1,6),'END TRANSACTION 1;')
      IF (CURS(1,6).NE.0) GO TO 10000
C
C     CALL ORACLE TO DESCRIBE THE 'DNAME' COLUMNS - ONLY THE LENG
C     IS OF CONCERN
C
      CALL ODSRBN(CURS(1,4),1,DEPTL)
      IF (CURS(1,4).NE.0) GO TO 10000
C
C     PUT THE MAXIMUM 'DNAME' LENGTH INTO A FORMAT SO THAT IT WIL
C     BE PRINTED CORRECTLY.
C
      ENCODE(70,70,DEPFMT)ENAMEL,DEPTL
70    FORMAT('('' '',',I2,'A1,'' added to the '',',I2,'A1,''
     1 department as employee # '',I5)')
```

```
C
C     DEFINE THE BUFFER TO RECEIVE 'DNAME' FOR ORACLE
C
      CALL ODFINN(CURS(1,4),1,STRNGS(ENAMEL+JOBL+4),DEPTL,5)
      IF (CURS(1,4).NE.0) GO TO 10000

C
C READ THE USER'S INPUT FROM DEVICE 5
C
100   WRITE (5,120)
120   FORMAT('$Enter employee name   : ')
      READ (5,ENMFMT,END=1000)(STRNGS(J),J=1,ENAMEL)
      IF (STRNGS(1).EQ.' ') GO TO 1000
      WRITE (5,140)
140   FORMAT('$Enter employee job    : ')
      READ (5,JOBFMT)(STRNGS(J),J=ENAMEL+2,ENAMEL+2+JOBL-1)
153   WRITE (5,155)
155   FORMAT('$Enter employee salary: ')
      READ (5,158,ERR=153)SAL
158   FORMAT(I5)
300   WRITE (5,160)
160   FORMAT('$Enter employee dept   : ')
      READ (5,158,ERR=300)DEPTNO
C
C     BIND THE DEPTNO VARIABLE
C
      CALL OBIND(CURS(1,4),'&DEPTNO',,DEPTNO,4,3)
      IF (CURS(1,4).NE.0) GO TO 700
C
C     EXECUTE THE SQL STATEMENT
C
      CALL OEXEC(CURS(1,4))
      IF (CURS(1,4).NE.0) GO TO 700
C
C     FETCH THE ROWS:  DEPTNO IS A UNIQUE COLUMN, SO A MAXIMUM OF
C     ONE ROW WILL BE FETCHED.  IF CURSOR RETURN CODE 4 IS
C     RETURNED, THEN THERE IS NO SUCH DEPARTMENT.
C
C     NOTE THAT THE DNAME AREA OF STRNGS WILL BE SET TO ALL NULLS
C     PRIOR TO THE CALL TO ORACLE
C
      DO 390 I=ENAMEL+JOBL+4,ENAMEL+JOBL+4+DEPTL+2
390   STRNGS(I)=0
      CALL OFETCH(CURS(1,4))
      IF (CURS(1,4).EQ.0) GO TO 410
      IF (CURS(1,4).NE.4) GO TO 700
      WRITE (5,400)
400   FORMAT('0No such department number')
      GO TO 300
```

```
C
C     BIND ALL SQL SUBSTITUTION VARIABLE VALUES
C     IF ANY ERRORS OCCUR, PRINT AN ERROR MESSAGE,
C     BUT CONTINUE.
C
410   CALL OBIND(CURS(1,2),'&ENAME',,STRNGS(1),ENAMEL,1)
      IF (CURS(1,2).NE.0) GO TO 700
      CALL OBIND(CURS(1,2),'&JOB',,STRNGS(ENAMEL+2),JOBL,1)
      IF (CURS(1,2).NE.0) GO TO 700
      CALL OBIND(CURS(1,2),'&SAL',,SAL,4,3)
      IF (CURS(1,2).NE.0) GO TO 700
      CALL OBIND(CURS(1,2),'&DEPTNO',,DEPTNO,4,3)
      IF (CURS(1,2).NE.0) GO TO 700
      CALL OBIND(CURS(1,3),'&DEPTNO',,DEPTNO,4,3)
      IF (CURS(1,3).NE.0) GO TO 700
C
C     EXECUTE THE SQL STATEMENTS.  CURSOR 5 ASKS ORACLE TO
C     BEGIN AN UPDATE TRANSACTION ON TABLES 'EMP' AND 'DEPT'.
C     ALL OTHER TRANSACTIONS ON THOSE TWO TABLES WILL BE
C     BLOCKED UNTIL AN 'END TRANSACTION': CURSOR 6.
C
      CALL OEXEC(CURS(1,5))
      IF (CURS(1,5).NE.0) GO TO 700
C
C     BIND THE EMPNO
C
450   CALL OBIND(CURS(1,2),'&EMPNO',,EMPNO,4,3)
      IF (CURS(1,2).NE.0) GO TO 700
C
C     EXECUTE THE INSERT (CURSOR 2)
C
500   CALL OEXEC(CURS(1,2))
      IF (CURS(1,2).EQ.0) GO TO 600
C
C     IF THE CALL RETURNS CODE -9 (DUPLICATE VALUE IN INDEX), THE
C     GENERATE THE NEXT POSSIBLE EMPLOYEE NUMBER
C
      IF (CURS(1,2).NE.-9) GO TO 700
      EMPNO=EMPNO+10
      GO TO 450
C
C     EXECUTE THE UPDATE (CURSOR 3)
C
600   CALL OEXEC(CURS(1,3))
      IF (CURS(1,3).NE.0) GO TO 700
530   WRITE (5,DEPFMT)(STRNGS(J),J=1,ENAMEL),(STRNGS(K),
     1   K=ENAMEL+2+JOBL+2,ENAMEL+2+JOBL+2+DEPTL-1),EMPNO
      GO TO 800
700   CALL ERRRPT(CURS(1,1),6)
800   CALL OEXEC(CURS(1,6))
      EMPNO=EMPNO+10
      GO TO 100
```

```
C
C     CLOSE THE FIVE CURSORS
C
1000  CALL OCLOSE(CURS(1,2))
      CALL OCLOSE(CURS(1,3))
      CALL OCLOSE(CURS(1,4))
      CALL OCLOSE(CURS(1,5))
      CALL OCLOSE(CURS(1,6))
C
C     LOGOFF FROM ORACLE
C
      CALL OLOGOF(CURS(1,1))
      STOP 'END OF SAMPL3'
10000 CALL ERRRPT(CURS(1,1),6)
      GO TO 1000
      END

      SUBROUTINE ERRRPT(CURS,N)
C
C     ERRRPT PRINTS THE CURSOR NUMBER, THE ERROR CODE, AND THE
C     ORACLE FUNCTION CODE.  IF THE LDA CONTAINS AN ERROR CODE,
C     A LOGON ERROR IS ASSUMED
C
C     CURS IS THE CURSOR ARRAY
C     N IS THE NUMBER OF CURSORS (INCLUDING THE LDA) IN THE ARRAY
C
      INTEGER*2 CURS(32,N)

      IF (CURS(1,1).EQ.0) GO TO 100
      WRITE (5,10)CURS(1,1)
10    FORMAT('0Logon error: ',I5)
      GO TO 500
100   DO 158 I=1,N
      IF (CURS(1,I).NE.0) GO TO 150
158   CONTINUE
150   IF (I.LE.N) GO TO 300
      WRITE (5,200)
200   FORMAT('0Unknown ORACLE error')
      GO TO 500
300   WRITE (5,400)I,CURS(1,I),CURS(6,I)
400   FORMAT('0ORACLE error on cursor ',I1,
     1   ': code is ',I5,', op is ',I5)
500   RETURN
      END
```

```
*****************************************
*                                       *
*                   C                   *
*                                       *
*   E X A M P L E    P R O G R A M S    *
*                                       *
*****************************************
```

Table of Contents

```
/* VOID sample1

   sample1 is a simple example program which adds new employee
           records to the personnel database.  Very little
           checking is done to insure the integrity of the
           database.  The program queries the user for data
           as follows:

           Enter employee number:
           Enter employee name:
           Enter employee job:
           Enter employee salary:
           Enter employee dept:

           The new employee record is inserted and the department
           table is updated to increase the employee count.  If
           the employee number is entered as '0', then the progra
           terminates.
*/
#include <std.h>
char dbn[]{"personnel"};                   /* data base name
char uid[]{"qa/test"};                     /* user id/password
char insert[]{"INSERT INTO EMP(EMPNO,ENAME,JOB,SAL,DEPTNO):\
<&EMPNO,&ENAME,&JOB,&SAL,&DEPTNO>;"};
char update[]{"UPDATE DEPT SET EMPCNT=\
NULLF(EMPCNT,0)+1 WHERE DEPTNO=&DEPTNO;"};
main()
{
   int empno,sal,deptno;                   /* employee number, salary
                                              department number
   int lda[32],curs1[32],curs2[32];        /* lda and two cursors
   char ename[11],job[11];                  /* employee name and job
/*
   log on to ORACLE, open the database (two cursors), and pass
   the SQL statements to ORACLE.  The program exits if any error
   occur.
*/
   if (ologon(lda,-1) ||
       oopen(curs1,lda,dbn,-1,-1,uid,-1) ||
       oopen(curs2,lda,dbn,-1,-1,uid,-1) ||
       osql (curs1,insert,-1) ||
       osql (curs2,update,-1))
           {
           errrpt(lda,curs1,curs2);
           goto errexit;
           }
```

```
/*
   read the user's input from STDIN.  If the employee number is
   entered as zero (or just <cr> or if -eof- (control Z) is
   encountered, exit.
*/
   for(;0 < askn("Enter employee number: ",&empno) && empno != 0;
       {
       asks("Enter employee name  : ",ename);
       asks("Enter employee job   : ",job);
       askn("Enter employee salary: ",&sal);
       askn("Enter employee dept:   ",&deptno);
/*
   bind all SQL substitution variable values and execute the SQL
   insert and update.  If any errors occur, print an error messag
   but continue.
*/
       if (obind(curs1,"&EMPNO",-1,&empno,2,3)   ||
           obind(curs1,"&ENAME",-1,ename,-1,1)   ||
           obind(curs1,"&JOB",-1,job,-1,1)       ||
           obind(curs1,"&SAL",-1,&sal,2,3)       ||
           obind(curs1,"&DEPTNO",-1,&deptno,2,3)||
           obind(curs2,"&DEPTNO",-1,&deptno,2,3)||
           oexec(curs1)                          ||
           oexec(curs2))
               errrpt(lda,curs1,curs2);
       }
errexit:
/*
   close the cursors and log off from ORACLE
*/
       oclose(curs1);
       oclose(curs2);
       ologof(lda);
       return(0);
}
/*
COUNT askn(text,variable)

   print the 'text' on STDOUT and read an integer variable from
   SDTIN.

   text points to the null terminated string to be printed
   variable points to an integer variable

   askn returns a 1 if the variable was read successfully or a
       -1 if -eof- was encountered
*/
int askn(text,variable)
   char text[];
   int *variable;
   {
   return(ask("%i",text,variable));
   }
```

```
/*
COUNT asks(text,variable)

   print the 'text' on STDOUT and read up to 10 characters into
   the buffer pointed to by variable from STDIN.

   text points to the null terminated string to be printed
   variable points to a buffer of at least 11 characters
   (to insure room for the trailing NULL)

   asks returns a 1 if the variable was read successfully or a
       -1 if -eof- was encountered
*/
asks(text,variable)
   char text[],variable[];
   {
   return(ask("%10.10p",text,variable));
   }
```

```
/*
COUNT ask(fmt,text,variable)

   print the 'text' on STDOUT and read from STDIN according to th
   format text pointed to by 'fmt'.  The format string is passed
   directly to the c library routine 'getfmt'.

   fmt points to a format string for getfmt
   text points to tne null terminated string to be printed
   variable points to a buffer of sufficient length to hold the
       input specified by the format.  No length checking is
       performed.

   ask returns a 1 if the variable was read successfully or a
       -1 if -eof- was encountered
*/
ask(fmt,text,variable)
   char fmt[],text[],variable[];
   {
   putfmt("\n%p",text);
   putch(-1);
   return(getfmt(fmt,variable));
   }
```

```
/*
VOID errrpt(lda,curs1,curs2)

   errrpt prints the cursor number, the error code, and the
   ORACLE function code.  If the lda contains an error code,
   a log on error is assumed.

   lda points to an ORACLE log on data area.
   curs1 points to an ORACLE cursor area.
   curs2 points to another ORACLE cursor area.
*/
errrpt(lda,curs1,curs2)
   int lda[32],curs1[32],curs2[32];
   {
   int curserr,cursfcn,cursnum;
   if (lda[0])
       putfmt("Logon error: %i\n",lda[0]);
       else
       {
       if (curs1[0])
           {
           cursnum=1;
           curserr = curs1[0];
           cursfcn = curs1[5];
           }
           else
           {
           cursnum=2;
           curserr = curs2[0];
           cursfcn = curs2[5];
           }
       putfmt("ORACLE error on cursor %i: \
code is %i, op is %i\n",cursnum,curserr,cursfcn);
       }
   return(0);
   }
```

```
/* VOID sample2

   sample2 is a simple example program which adds new employee
           records to the personnel database.  Some checking
           is done to insure the integrity of the data base.
           The program queries the user for data as follows:

           Enter employee number:
           Enter employee name:
           Enter employee job:
           Enter employee salary:
           Enter employee dept:

           The new employee record is inserted and the department
           table is updated to increase the employee count.  If
           the employee number is not entered, then the program
           terminates.

*/
#include <std.h>
char dbn[]{"personnel"};                  /* data base name
char uid[]{"qa/test"};                    /* user id/password
char insert[]{"INSERT INTO EMP(EMPNO,ENAME,JOB,SAL,DEPTNO):\
<&EMPNO,&ENAME,&JOB,&SAL,&DEPTNO>;"};
char update[]{"UPDATE DEPT SET EMPCNT=\
NULLF(EMPCNT,0)+1 WHERE DEPTNO=&DEPTNO;"};
char select[]{"SELECT DNAME FROM DEPT WHERE \
DEPTNO=&DEPTNO;"};
main()
{
   int empno,sal,deptno;                  /* employee number, salary
                                             department number
   int curs[4][32];                       /* lda and three cursors
   char ename[11],job[11],dept[21];       /* employee name,job,dept
/*
   log on to ORACLE, open the data base (three cursors), and pars
   the SQL statements.  The program exits if any errors occur.
*/
   if (ologon(curs[0],-1) ||
       oopen(curs[1],curs[0],dbn,-1,-1,uid,-1) ||
       oopen(curs[2],curs[0],dbn,-1,-1,uid,-1) ||
       oopen(curs[3],curs[0],dbn,-1,-1,uid,-1) ||
       osql (curs[1],insert,-1) ||
       osql (curs[2],update,-1) ||
       osql (curs[3],select,-1) ||
       odfinn (curs[3],1,dept,21,5,-1))
           {
           errrpt(curs[0],4);
           goto errexit;
           }
```

```
/*
   read the user's input from STDIN.  If the employee number is
   entered as zero (or just <cr> or if -eof- (control Z) is
   encountered,exit.
   Verify that the entered department number is valid and echo th
   departments name
*/
   for(;0 < askn("Enter employee number: ",&empno) && empno != 0;
       {
       asks("Enter employee name   : ",ename);
       asks("Enter employee job    : ",job);
       askn("Enter employee salary: ",&sal);
       askn("Enter employee dept:    ",&deptno);
/*
   bind all SQL substitution variable values and execute the SQL
   statements.  If any errors occur, print an error message,
   then continue.
*/
       if (obind(curs[1],"&EMPNO",-1,&empno,2,3)   ||
           obind(curs[1],"&ENAME",-1,ename,-1,1)   ||
           obind(curs[1],"&JOB",-1,job,-1,1)       ||
           obind(curs[1],"&SAL",-1,&sal,2,3)       ||
           obind(curs[1],"&DEPTNO",-1,&deptno,2,3)||
           obind(curs[2],"&DEPTNO",-1,&deptno,2,3)||
           obind(curs[3],"&DEPTNO",-1,&deptno,2,3)||
           oexec(curs[3]) ||
           ofetch(curs[3])||
           oexec(curs[1])                          ||
           oexec(curs[2]))
               {
               if (curs[3][0]==4)
                   putfmt("\nNo such department number\n");
                   else
                   errrpt(curs[0],4);
               }
           else
           putfmt ("\n%p added to the %p department\n",ename,dept
       }
errexit:
/*
   close the cursors and log off from ORACLE
*/
       oclose(curs[1]);
       oclose(curs[2]);
       ologof(curs[0]);
       return(0);
}
```

```
/*
COUNT askn(text,variable)

   print the 'text' on STDOUT and read an integer variable from
   SDTIN.

   text points to the null terminated string to be printed
   variable points to an integer variable

   askn returns a 1 if the variable was read successfully or a
       -1 if -eof- was encountered
*/
int askn(text,variable)
   char text[];
   int *variable;
   {
   return(ask("%i",text,variable));
   }
```

```
/*
COUNT asks(text,variable)

   print the 'text' on STDOUT and read up to 10 characters into
   the buffer pointed to by variable from STDIN.

   text points to the null terminated string to be printed
   variable points to a buffer of at least 11 characters
   (to insure room for the NULL)

   asks returns a 1 if the variable was read successfully or a
       -1 if -eof- was encountered
*/
asks(text,variable)
   char text[],variable[];
   {
   return(ask("%10.10p",text,variable));
   }
/*
COUNT ask(fmt,text,variable)

   print the 'text' on STDOUT and read from STDIN according to th
   format text pointed to by 'fmt'.  The format string is passed
   directly to the c library routine 'getfmt'.

   fmt points to a format string for getfmt
   text points to tne null terminated string to be printed
   variable points to a buffer of sufficient length to hold the
       input specified by the format.  No length checking is
       performed.

   ask returns a 1 if the variable was read successfully or a
       -1 if -eof- was encountered
*/
ask(fmt,text,variable)
   char fmt[],text[],variable[];
   {
   putfmt("\n%p",text);
   putch(-1);
   return(getfmt(fmt,variable));
   }
/*
VOID errrpt(cur,n)

   errrpt prints the cursor number, the error code, and the
   ORACLE function code.  If the lda contains an error code,
   a log on error is assumed.

   cur points to an ORACLE cursor array. curs[0] is assummed to b
       the lda.
   n the the number of cursors in the array (including the lda)
*/
errrpt(cur,n)
```

```
    int n;
    int cur[][32];
    {
    int i;
    if (cur[0][0])
        putfmt("Logon error: %i\n",cur[0][0]);
        else
        {
        for (i=1;i>=n||cur[i][0]!=0;i+=1){}
        if (i==n)
        putfmt("Unknown ORACLE error\n");
        else
        putfmt("ORACLE error on cursor %i: \
code is %i, op is %i\n",i,cur[i][0],cur[i][5]);
        }
    return(0);
    }
```

```
/* VOID sample3

   sample3 is a simple example program which adds new employee
           records to the personnel data base.  Checking
           is done to insure the integrity of the data base.
           The employee numbers are automatically selected using
           the current maximum employee number as the start.
           If any employee number is a duplicate, it is skipped.
           The program queries the user for data as follows:

           Enter employee name:
           Enter employee job:
           Enter employee salary:
           Enter employee dept:

           The new employee record is inserted and the department
           table is updated to increase the employee count.  If
           the employee name is not entered, then the program
           terminates.

           If the record is successfully inserted, the following
           is printed:

           ename added to department dname as employee # nnnnnn

*/
#include <std.h>
char dbn[]{"personnel"};                    /* data base name
char uid[]{"qa/test"};                      /* user id/password
char insert[]{"INSERT INTO EMP(EMPNO,ENAME,JOB,SAL,DEPTNO):\
<&EMPNO,&ENAME,&JOB,&SAL,&DEPTNO>;"};
char update[]{"UPDATE DEPT SET EMPCNT=\
NULLF(EMPCNT,0)+1 WHERE DEPTNO=&DEPTNO;"};
char select[]{"SELECT DNAME FROM DEPT WHERE \
DEPTNO=&DEPTNO;"};
char maxemp[]{"SELECT MAX(EMPNO) + 10 FROM EMP;"};
char selemp[]{"SELECT ENAME,JOB FROM EMP;"};   /* used to determi
                                                  ename, job size
char begtrn[]{"BEGIN TRANSACTION 1 ON TABLE EMP,DEPT UPDATE;"};
char endtrn[]{"END TRANSACTION 1;"};
main()
{
   int empno,sal,deptno;                    /* employee number, salary
                                               department number
   int curs[6][32];                         /* lda and three cursors
   char strings[100];                       /* employee name,job,dept
   int enamel,jobl,deptl;                   /* the max length of cols
```

```
/*
   log on to ORACLE, open the data base (three cursors), and pars
   the SQL statements.  The program exits if any errors occur.
   Determine the lengths of the variable length strings via ODSRB
*/
   if (ologon(curs[0],-1) ||
       oopen(curs[1],curs[0],dbn,-1,-1,uid,-1) ||
       oopen(curs[2],curs[0],dbn,-1,-1,uid,-1) ||
       oopen(curs[3],curs[0],dbn,-1,-1,uid,-1) ||
       oopen(curs[4],curs[0],dbn,-1,-1,uid,-1) ||
       oopen(curs[5],curs[0],dbn,-1,-1,uid,-1))
           {
           errrpt(curs[0],6);
           goto errexit;
           }
       /*
           retrieve the current maximum employee number
       */
       if (osql(curs[1],maxemp,-1) ||
           odfinn(curs[1],1,&empno,2,3,-1) ||
           oexec(curs[1]) ||
           ofetch(curs[1]))
           {
           if(curs[1][0]==4) empno=10;
           else
               {
               errrpt(curs[0],6);
               goto errexit;
               }
           }
/*
   determine the max length of the employee name and job title
*/
       if (osql(curs[1],selemp,-1)||
           odsrbn(curs[1],1,&enamel,-1,-1) ||
           odsrbn(curs[1],2,&jobl,-1,-1)   ||
           odfinn(curs[1],1,strings,enamel,1,-1) ||
           odfinn(curs[1],2,&strings[enamel+1],jobl,1,-1))
           {
           errrpt(curs[0],6);
           goto errexit;
           }
```

```
      /
/*
   parse the insert, select, and update statements
*/
       if (osql (curs[2],update,-1) ||
       osql (curs[3],select,-1) ||
       osql (curs[1],insert,-1) ||
       osql (curs[4],begtrn,-1) ||
       osql (curs[5],endtrn,-1) ||
       odsrbn(curs[3],1,&deptl,-1,-1) ||
       odfinn (curs[3],1,&strings[enamel+jobl+2],deptl,5,-1))
           {
           errrpt(curs[0],6);
           goto errexit;
           }
/*
   read the user's input from STDIN.  If the employee name is
   not entered, exit.
   Verify that the entered department number is valid and echo th
   department's name
*/
   for(;0 < asks("Enter employee name  : ",\
       strings,enamel);empno+=10)
       {
       asks("Enter employee job   : ",&strings[enamel+1],jobl);
       askn("Enter employee salary: ",&sal);
       for (;0>=askn("Enter employee dept  :   ",&deptno)||
       obind(curs[3],"&DEPTNO",-1,&deptno,2,3)||
           oexec(curs[3]) ||
           ofetch(curs[3]);
           putfmt("\nNo such department\n")){}
/*
   bind all SQL substitution variable values and execute the SQL
   statements.  If any errors occur, print an error message,
   then continue.
*/
       if (obind(curs[1],"&ENAME",-1,strings,-1,1)  ||
           obind(curs[1],"&JOB",-1,&strings[enamel+1],-1,1)
           obind(curs[1],"&SAL",-1,&sal,2,3)      ||
           obind(curs[1],"&DEPTNO",-1,&deptno,2,3)||
           obind(curs[2],"&DEPTNO",-1,&deptno,2,3)||
           oexec(curs[4]))
               errrpt(curs[0],6);
               else
               {
               for (; 0==obind(curs[1],"&EMPNO",-1,&empno,2,3) &&
                     -9==oexec(curs[1]);empno+=10)
                     /*  test code */
                     {
                     errrpt(curs[0],6);
                     }
                     /* end of test code */
```

```
                 if (curs[1][0])
                     errrpt(curs[0],6);
                     else
                     {
                     if (oexec(curs[2]))
                         errrpt(curs[0],6);
                     else
                         putfmt("\n%p added to the %p department \
as employee number %i\n",\
                         strings,&strings[enamel+jobl+2],empno);
                     }
                 if (oexec(curs[5]))
                     errrpt(curs[0],6);
                 }
         }
errexit:
/*
   close the cursors and log off from ORACLE
*/
       oclose(curs[1]);
       oclose(curs[2]);
       oclose(curs[3]);
       oclose(curs[4]);
       oclose(curs[5]);
       ologof(curs[0]);
       return(0);
}
/*
COUNT askn(text,variable)

   print the 'text' on STDOUT and read an integer variable from
   SDTIN.

   text points to the null terminated string to be printed
   variable points to an integer variable

   askn returns a 1 if the variable was read successfully or a
       -1 if -eof- was encountered
*/
int askn(text,variable)
   char text[];
   int *variable;
   {
   return(ask("%i",text,variable));
   }
```

```
/*
COUNT asks(text,variable,len)

   print the 'text' on STDOUT and read up to 'len' characters int
   the buffer pointed to by variable from STDIN.

   text points to the null terminated string to be printed
   variable points to a buffer of at least 'len'+1 characters

   asks returns the number of character read into the string, or
       -1 if -eof- was encountered
*/
asks(text,variable,len)
   char text[],variable[];
   {
   char fmt[14],lens[6];
   int x;
   x=itob(lens,len,10);
   lens[x]='\0';
   cpystr(fmt,"%",lens,".",lens,"p",NULL);
   return(EOF==ask(fmt,text,variable)?EOF:lenstr(variable));
   }
/*
COUNT ask(fmt,text,variable)

   print the 'text' on STDOUT and read from STDIN according to th
   format text pointed to by 'fmt'.  The format string is passed
   directly to the c library routine 'getfmt'.

   fmt points to a format string for getfmt
   text points to tne null terminated string to be printed
   variable points to a buffer of sufficient length to hold the
       input specified by the format.  No length checking is
       performed.

   ask returns a 1 if the variable was read successfully or a
       -1 if -eof- was encountered
*/
ask(fmt,text,variable)
   char fmt[],text[],variable[];
   {
   putfmt("\n%p",text);
   putch(-1);
   return(getfmt(fmt,variable));
   }
```

```
/*
VOID errrpt(cur,n)

   errrpt prints the cursor number, the error code, and the
   ORACLE function code.  If the lda contains an error code,
   a log on error is assumed.

   cur points to an ORACLE cursor array. curs[0] is assummed to b
       the lda.
   n the the number of cursors in the array (including the lda)
*/
errrpt(cur,n)
   int n;
   int cur[][32];
   {
   int i;
   if (cur[0][0])
       putfmt("Logon error: %i\n",cur[0][0]);
       else
       {
       for (i=1;i<n&&cur[i][0]==0;i+=1){}
       if (i==n)
       putfmt ("Unknown ORACLE error\n");
       else
       putfmt("ORACLE error on cursor %i: \
code is %i, op is %i\n",i,cur[i][0],cur[i][5]);
       }
   return(0);
   }
```

```
*****************************************
*                                       *
*               C O B O L               *
*                                       *
*    E X A M P L E    P R O G R A M     *
*                                       *
*****************************************
```

Table of Contents

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ORACBL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  PDP-11.
OBJECT-COMPUTER.  PDP-11.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  LDA.
    02  LDA-RC   PIC S9999 COMP.
    02  FILLER   PIC S9999 COMP OCCURS 31 TIMES.
01  CURSOR.
    02  C-RC     PIC S9999 COMP.
    02  C-TYPE   PIC S9999 COMP.
    02  C-ROWS   PIC S9(9) COMP.
    02  C-OFFS   PIC S9999 COMP.
    02  C-FNC    PIC S9999 COMP.
    02  FILLER   PIC S9999 COMP OCCURS 26 TIMES.
77  AREA-COUNT   PIC S9999 VALUE 1 COMP.
77  AREA-SIZE    PIC S9999 VALUE 3 COMP.
77  DATA-BASE    PIC X(16) VALUE "personnel".
77  DATA-BASE-L  PIC S9999 VALUE 9 COMP.
77  USER-ID      PIC X(8)  VALUE "qa/test".
77  USER-ID-L    PIC S9999 VALUE 7 COMP.
77  SQL-SEL      PIC X(40) VALUE "SELECT EMPNO,ENAME FROM EMP".
77  SQL-SEL-L    PIC S9999 VALUE 40 COMP.
77  SQL-INS      PIC X(40) VALUE "INSERT INTO EMP:<&EMPNOX,&ENAMEX
77  SQL-INS-L    PIC S9999 VALUE 40 COMP.
77  EMPNO-RC     PIC S9999 COMP.
77  EMPNO-N      PIC S9999 VALUE 1 COMP.
77  ENAME        PIC X(19).
77  ENAME-L      PIC S9999 VALUE 10 COMP.
77  ENAME-RC     PIC S9999 COMP.
77  ENAME-N      PIC S9999 VALUE 2 COMP.
77  FTYPE        PIC S9999 COMP.
77  ERRTYPE      PIC +9999.
77  EMPNOX       PIC S9(9) COMP VALUE 0.
77  EMPNOX-L     PIC S9999 VALUE 4 COMP.
77  EMPNOX-N     PIC X(7) VALUE "&EMPNOX".
77  EMPNOX-N-L   PIC S9999 COMP VALUE 7.
77  ENAMEX-N     PIC X(7) VALUE "&ENAMEX".
77  ENAMEX-N-L   PIC S9999 COMP VALUE 7.
77  INT4         PIC S9999 COMP VALUE 3.
77  ASC          PIC S9999 COMP VALUE 1.
77  EMPNOX-A     PIC S9(9) SIGN LEADING SEPARATE DISPLAY
                 VALUE -1.
77  EMPNOX-A-X   REDEFINES EMPNOX-A PIC X(10).
77  EMPNOX-A-L   PIC S9999 COMP VALUE 10.
77  MSG-TO-OP    PIC X(45) VALUE
                    "+^^^^^^^^^ Enter new empno (+|- and 9 digits)
```

```
PROCEDURE DIVISION.
BEGIN.
*
* LOGON TO ORACLE
*
    CALL "OLOGON" USING LDA-RC,AREA-COUNT.
    IF LDA-RC NOT = 0 PERFORM ORA-ERR GO TO EXIT-STOP.
*
* OPEN THE PERSONNEL DATA BASE
*
    CALL "OOPEN" USING C-RC,LDA-RC,DATA-BASE,DATA-BASE-L,
    AREA-SIZE,USER-ID,USER-ID-L.
    IF C-RC NOT = 0 PERFORM ORA-ERR GO TO EXIT-LOGOF.
*
* INSERT A RECORD
*
    CALL "OSQL" USING C-RC,SQL-INS,SQL-INS-L.
    IF C-RC NOT = 0 PERFORM ORA-ERR GO TO EXIT-CLOSE.
INSERT-ONE.
    DISPLAY MSG-TO-OP.
    ACCEPT EMPNOX-A.
    IF EMPNOX-A = 0 GO TO SELECT-IT.
    MOVE EMPNOX-A TO EMPNOX.
    CALL "OBIND" USING C-RC,EMPNOX-N,EMPNOX-N-L,EMPNOX,
       EMPNOX-L,INT4.
    IF C-RC NOT = 0 PERFORM ORA-ERR GO TO INSERT-ONE.
    CALL "OBIND" USING C-RC,ENAMEX-N,ENAMEX-N-L,EMPNOX-A-X,
       EMPNOX-A-L,ASC.
    IF C-RC NOT = 0 PERFORM ORA-ERR GO TO INSERT-ONE.
    CALL "OEXEC" USING C-RC.
    IF C-RC NOT = 0 PERFORM ORA-ERR.
    GO TO INSERT-ONE.
*
* PARSE THE SELECT
*
SELECT-IT.
    CALL "OSQL" USING C-RC,SQL-SEL,SQL-SEL-L.
    IF C-RC NOT = 0 PERFORM ORA-ERR GO TO EXIT-CLOSE.
*
* DEFINE THE RECEIVING DATA AREAS
*
    CALL "ODFINN" USING C-RC,EMPNO-N,EMPNOX,EMPNOX-L,INT4,
    EMPNO-RC.
    IF C-RC NOT = 0 PERFORM ORA-ERR GO TO EXIT-CLOSE.
    CALL "ODFINN" USING C-RC,ENAME-N,ENAME,ENAME-L,ASC,
    ENAME-RC.
    IF C-RC NOT = 0 PERFORM ORA-ERR GO TO EXIT-CLOSE.
```

```
*
* EXECUTE THE QUERY BLOCK
*
    CALL "OEXEC" USING C-RC.
    IF C-RC NOT = 0 PERFORM ORA-ERR GO TO EXIT-CLOSE.
FETCH-ONE.
*
* BLANK ALPHA AREAS AND FETCH THE ROWS
*
    MOVE SPACES TO ENAME.
    CALL "OFETCH" USING C-RC.
    IF ENAME-RC NOT = 0
       MOVE ENAME-RC TO ERRTYPE
       DISPLAY "NON-ZERO RETURN ON FETCH OF ENAME; CODE IS :",
           ERRTYPE.
    IF C-RC NOT = 0 PERFORM ORA-ERR  GO TO EXIT-S.
    MOVE EMPNOX TO EMPNOX-A.
    DISPLAY "EMPNO = ",EMPNOX-A," ;ENAME = ",ENAME.
    GO TO FETCH-ONE.
EXIT-S.
EXIT-CLOSE.
*
* CLOSE THE DATA BASE
*
    CALL "OCLOSE" USING C-RC.
    IF C-RC NOT = 0 PERFORM ORA-ERR.
EXIT-LOGOF.
*
* LOG OFF FROM ORACLE
*
    CALL "OLOGOF" USING LDA-RC.
    IF LDA-RC NOT = 0 PERFORM ORA-ERR.
EXIT-STOP.
    STOP RUN.
ORA-ERR.
*
* PRINT ORACLE ERROR NOTICE
*
    DISPLAY "ORACLE ERROR"
    MOVE C-FNC TO ERRTYPE.
    DISPLAY "ORACLE FUNCTION = ",ERRTYPE.
    MOVE LDA-RC TO ERRTYPE.
    DISPLAY "LDA ERROR = ",ERRTYPE.
    MOVE C-RC TO ERRTYPE.
    DISPLAY "CUR ERROR = ",ERRTYPE.
```

O R A C L E

ASSEMBLY LANGUAGE INTERFACE

To use the ORACLE user interface from a MACRO-11 language program, the programmer must invoke and use the CCALL macro as described below. The CCALL macro works as follows:

1. The CCALL macro is the SDLLIB.MLB. The user must include this library as part of his assembly.

2. The CCALL macro is invoked with the .MCALL directive.

3. The format of the CCALL macro is:

   ```
           CCALL   FUNC,P1,...,Pn
   ```

   where:

FUNC	is the function to be called (e.g. OOPEN, OSQL,...)
P1,...,P2	is the parameter list as defined in the programming interface. All of the parameters must be able to be objects of a MOV instruction. A parameter can be a list of values, in which case the values are added together before they are put in the parameter list. An example of this is <R1,#OFFSET>. This would cause R1 and #OFFSET to be added together to produce a single parameter for the parameter list. This mechanism allows users to easily pass pointers to a data area in a structure pointed to by a register. Also, if the first item in this list is 'B' then the next item in the list is a byte and must be able to be the object of a MOVB instruction.

4. An example of the CCALL macro:

   ```
   CURSOR: .BLKB   64.
   LDA:    .BLKB   64.
   DBNAME: .ASCIZ  /ACCOUNTING/
           CCALL   OOPEN,#CURSOR,#LDA,#DBNAME,#-1,#-1
   ```

5. When calling an ORACLE function, registers R0 and R1 are volatile and may be destroyed.

6. Upon return from an ORACLE function, R0 contains the return code (the first word of the cursor).

7. To include the MACRO-11 interface to ORACLE the user should follow the instructions that C programmers do to include the C interface.

ORACLE

HOST LANGUAGE INTERFACE

LINKING INSTRUCTIONS FOR RSX/IAS

When a user wishes to create his or her own database task, the distributed user interface modules must be included in the task image. The user code interface is provided in four object libraries:

* OFOLIB.OLB

* OCELIB.OLB

* ORALIB.OLB

* CLIB.OLB

The first two libraries, OFOLIB and OCELIB, provide the Host Language Interface between a host language, such as FORTRAN, and ORACLE. The libraries are used in the following way in the task build:

```
OCELIB/LB:ORACEE for C and assembly language programs
OFOLIB/LB:ORAFOR for FORTRAN and COBOL programs
```

The third library, ORALIB, contains object code which has several responsibilities. The first is to provide the primary intertask communication and handshake between the user task and ORACLE via the SDAT$ and RCVD$ system service routines. The second function provides the user interface to the ORACLE context region. This region is used by the user interface to pass commands and data to ORACLE and used in turn by ORACLE to pass data back to the user. This module will identify the user task name to ORACLE and then wait to receive a mapping context message from ORACLE. Upon the completion of this initialization, the user task may enter into transactions with ORACLE.

The fourth library contains common routines used by the 'C' language programs in OFOLIB, OCELIB, and ORALIB.

LINKING AN RSX-11M TASK

Two special steps must be taken to include the ORACLE Interface program in a user task.

1. Include OFOLIB or OCELIB, ORALIB, and CLIB in the list of library files.

2. Add an extra address window at task build time.

Examples:

The TKB command input which builds the SAMPL1 FORTRAN program:

```
SAMPL1/-FP,SAMPL1/-SP
SAMPL1,[1,1]F4PEIS,[1,1]F4POTS/LB
[1,1]OFOLIB/LB:ORAFOR
[1,1]ORALIB/LB
[1,1]CLIB/LB
/
WNDWS=1
STACK=3000
UNITS=13
//
```

The TKB command input which builds the sample3 "C" program:

```
sample3/cp,sample3/-sp
sample3
[1,1]ocelib/lb:oracee
[1,1]oralib/lb
[1,1]clib/lb:chdr
[1,1]clib/lb
/
stack=3000
units = 13
wndws = 1
extsct=$99998:3000
//
```

LINKING AN IAS TASK

Two special steps are required to include the ORACLE Interface in a user task.

1. Include OFOLIB, OCELIB or ORALIB, and CLIB in the list of object files;

2. Add an extra region descriptor block;

Examples:
The command input to TKB to build the SAMPL1 FORTRAN program:

```
SAMPL1/CP/-FP,SAMPL1/-SP
SAMPL1
F4POST/LB
OFOLIB/LB:ORAFOR
ORALIB/LB
CLIB/LB
/
STACK=3000
UNITS = 13
ATRG = 1
//
```

The command input to TKB to build the sample3 "C" program:

```
sample3/cp,sample3/-sp
sample3
ocelib/lb:oracee
oralib/lb
clib/lb:chdr
clib/lb
/
stack=3000
units = 13
atrg = 1
extsct=$99998:3000
//
```

Note: if a user task is multi-user, the task must have a name of the form: $$$abc

SYSTEM RESOURCES FOR PDP-11 PROGRAMS

ORACLE makes use of certain system resources which are discussed here to prevent conflict with user software.

SEND/RECEIVE DATA directives may not be used.

Neither SQL statements nor data to be used by ORACLE may reside in locations mapped to addresses 140000(8) through 177777(8).

None of the modules included from the ORACLE libraries (OFOLIB, OCELIB, ORALIB, and CLIB) may be mapped by an address window which also maps virtual locations 140000(8) through 177777(8).

Programs which are larger than 140000(8) bytes should use an overlay structure in which the ORACLE library modules are in the root and the remainder of the program is in a memory resident overlay. An example task builder imput file is contained in DBFTKB.CMD and DBF.ODL which follow.

DBFTKB.CMD (referenced by TKB @DBFTKB on RSX11M systems):

```
DBF/CP,DBF/-SP=DBF/MP
STACK=3000
UNITS=13
WNDWS=1
TASK=...DBF
ASG=TI:13
EXTSCT=$99998:4000
/
```

DBF.ODL (referenced above in the first TKB command line):

```
.ROOT    DBF010-*!(DBF040-DBF050)
DBF010: .FCTR   DBF020-DBF025-DBF030
DBF020: .FCTR   CLIB/LB:CHDR:.END:SBREAK
DBF025: .FCTR   DBFLIB/LB:DBFKNL:DBFDTA
DBF030: .FCTR   CLIB/LB-OCELIB/LB:ORACEE-ORALIB/LB-CLIB/LB
DBF040: .FCTR   DBFLIB/LB:MAIN-DBFLIB/LB
DBF050: .FCTR   ORALIB/LB-CLIB/LB
        .END
```

LINKING INSTRUCTIONS FOR VAX/VMS

Two object libraries provide the native mode user code interface on the VAX/VMS distribution:

* ORALIB.OLB

* CLIB.OLB

The first library, ORALIB, provides the Host Language Interface between a host language, such as FORTRAN, and ORACLE. The library also contains an object code which has several responsibilities. The first is to provide for the creation of and subsequent communication with a detached ORACLE process. The second function provides the user interface to the ORACLE context region. This region is used by the user interface to pass commands and data to ORACLE and used in turn, by ORACLE to pass data back to the user.

It should be noted a user image must reserve the address range 0C000-10000 (hex). This is may be accomplished by including the distributed pad option file in your link command. (Note the inclusion of ORAPAD.OPT in the example, below).

A sample command procedure for linking a "C" program is listed below:

```
$!
$!       UFILNK.COM
$!
$!      VAX/VMS UFI LINK COMMAND FILE
$!
$DEL UFI.EXE;*,UFI.MAP;*
$LINK /EXE=UFI/MAP=UFI -
     CLIB/INCLU=CHDR/LIBR,-
     UFILIB/INCLU=UFIPAD/LIBR,-
     ORAPAD/OPTIONS,-
     ORALIB/LIBR,-
     CLIB/LIBR
```

A sample command procedure for linking a FORTRAN program is listed below:

```
$!
$!      SAMPLE FORTRAN/ORACLE LINK PROCEDURE
$!
$LINK /EXE=SAMPL1/MAP=SAMPL1 -
     SAMPL1,-
     ORALIB/LIBR,-
     ORAPAD/OPTIONS,-              ! PADS 0C000-10000
     CLIB/LIBR
```