Dynamic SQL Using DBMS_SQL

Pankaj Jain

@twit_pankajj

pluralsight
hardcore dev and IT training

Why Use DBMS_SQL?

Return Results to Client

Unknown Number of Select Columns

Unknown Number of PlaceHolders

Type of Statements

DML, DDL, Alter Session Statements

Queries

Procedures & Functions

Anonymous Blocks

DBMS_SQL Workflow

Open Cursor

Parse

Bind Variable

Define Column

Execute

Fetch Rows

Variable Value

Column Value

Close Cursor

DBMS_SQL

Owned By SYS

Invoker’s Right

Executing DDL & Session Control Statements

Open Cursor

Parse

Bind Variable

Define Column

Execute

Fetch Rows

Variable Value

Column Value

Close Cursor

Executing DDL & Session Control Statements

Open Cursor

Parse

Execute

Close Cursor

Executing DDL Statements

- Open Cursor

```
FUNCTION OPEN_CURSOR RETURN INTEGER;
```

```
FUNCTION OPEN_CURSOR(security_level IN INTEGER) RETURN INTEGER;
```

  - 0 No security check
  - 1 Userid / Role Parsing Be the Same as Binding / Executing
  - 2 Most Secure

```
CREATE OR REPLACE PROCEDURE drop_table (p_table_name VARCHAR2)  IS
   l_sql  VARCHAR2(100);
   l_cursor_id INTEGER;
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
..
..
END drop_table;
```

Executing DDL Statements

- Parse

```
PROCEDURE PARSE( c                    IN INTEGER,
                 statement            IN  VARCHAR2,
                 language_flag IN  INTEGER);
```

- Language Flag
  - V6 , V7, NATIVE, FOREIGN_SYNTAX

```
CREATE OR REPLACE PROCEDURE drop_table (p_table_name VARCHAR2)  IS
   l_sql  VARCHAR2(100);
   l_cursor_id INTEGER;
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
..
..
END drop_table;
```

Executing DDL Statements

- Execute

```
FUNCTION EXECUTE (c IN INTEGER) RETURN INTEGER;
```

- Optional For DDL

```
CREATE OR REPLACE PROCEDURE drop_table (p_table_name VARCHAR2)  IS
   l_sql  VARCHAR2(100);
   l_cursor_id INTEGER;
   l_return  INTEGER;
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
   l_return := DBMS_SQL.EXECUTE(l_cursor_id);
..
END drop_table;
```

Executing DDL Statements

- Close Cursor

```
PROCEDURE CLOSE_CURSOR( c IN OUT INTEGER);
```

```
CREATE OR REPLACE PROCEDURE drop_table (p_table_name VARCHAR2)  IS
   l_sql  VARCHAR2(100);
   l_cursor_id INTEGER;
   l_return  INTEGER;
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
   l_return := DBMS_SQL.EXECUTE(l_cursor_id);
   DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END drop_table;
```

Executing Session Control Statements

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-RRRR';
```

```
CREATE OR REPLACE PROCEDURE alter_format (p_format VARCHAR2)  IS

            l_sql  VARCHAR2(100);
            l_cursor_id INTEGER;
            l_return  INTEGER;
 BEGIN
            l_sql := 'ALTER SESSION SET NLS_DATE_FORMAT = '||p_format;
            l_cursor_id := DBMS_SQL.OPEN_CURSOR;
            DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
            l_return := DBMS_SQL.EXECUTE(l_cursor_id);
            DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END alter_format;
```

```
EXEC alter_format('''DD-MON-RRRR''');
```

Executing DML Statements, Subprograms & Anonymous Blocks

- Open Cursor
- Parse
- Bind Variable
- Define Column
- Execute
- Fetch Rows
- Variable Value
- Column Value
- Close Cursor

Executing DML Statements, Subprograms & Anonymous Blocks

Open Cursor

Parse

Bind Variable

Execute

Variable Value

Close Cursor

Bind Variable

```
DBMS_SQL.BIND_VARIABLE (
  c           IN INTEGER,
  name        IN VARCHAR2,
  value       IN VARCHAR2 CHARACTER SET ANY_CS [,out_value_size IN
                                                        INTEGER]);
```

```
dbms_sql.bind_variable(
  c       IN INTEGER,
  name    IN VARCHAR2,
  value   IN NUMBER);
```

```
dbms_sql.bind_variable(
c       IN INTEGER,
name  IN VARCHAR2,
value  IN DATE);
```

Executing DML Statements

- Insert Statement

```
CREATE OR REPLACE PROCEDURE insert_record (p_table_name    VARCHAR2,
                                          p_col1_name     VARCHAR2,
                                          p_col1_value    NUMBER,
                                          p_col2_name     VARCHAR2,
                                          p_col2_value    NUMBER ) IS
          l_sql  VARCHAR2(100);
          l_cursor_id INTEGER;
          l_return  INTEGER;
 BEGIN
          l_sql := 'INSERT INTO  '||p_table_name || '('||
                    p_col1_name||' , '||
                    p_col2_name||
                    ' )   '||
                    'VALUES( :col1_value,:col2_value)';
          l_cursor_id := DBMS_SQL.OPEN_CURSOR;
          DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
          DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':col1_value', p_col1_value);
          DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':col2_value', p_col2_value);
          l_return := DBMS_SQL.EXECUTE(l_cursor_id);
          DBMS_OUTPUT.PUT_LINE('Rows Processed '||l_return);
          DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
          COMMIT;
END insert_record;
```

Variable Value

```
dbms_sql.variable_value(
c    IN  INTEGER,
name  IN  VARCHAR2,
value OUT VARCHAR2 CHARACTER SET ANY_CS);
```

```
dbms_sql.variable_value(
c    IN  INTEGER,
name  IN  VARCHAR2,
value OUT NUMBER);
```

```
dbms_sql.variable_value(
c    IN  INTEGER,
name  IN  VARCHAR2,
value OUT DATE);
```

Executing DML Statements

- Returning Into Clause

```
DECLARE
      l_item_value   items.item_value%TYPE   := 100;
      l_item_id        items.item_id%TYPE        := 1;
      l_item_name   items.item_name%TYPE;

      l_sql  VARCHAR2(200);
      l_cursor_id INTEGER;
      l_return  INTEGER;
  BEGIN
      l_sql :=  'UPDATE   items SET item_value =  :p_item_val '||
          ' WHERE item_id = :p_item_id RETURNING item_name INTO  :l_name '  ;
      l_cursor_id :=  DBMS_SQL.OPEN_CURSOR;
      DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_val', l_item_value);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_id', l_item_id);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':l_name', l_item_name,60);
      l_return := DBMS_SQL.EXECUTE(l_cursor_id);
      DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':l_name', l_item_name);
      DBMS_OUTPUT.PUT_LINE('Rows Processed '||l_return||'Item Name '||l_item_name);
      DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
      COMMIT;
   END;
```

Executing DML Statements

- Returning Into Clause

```
DECLARE
     l_item_value   items.item_value%TYPE    := 100;
     l_item_id      items.item_id%TYPE       := 1;
     l_item_name    items.item_name%TYPE;

     l_sql  VARCHAR2(200);
     l_cursor_id INTEGER;
     l_return  INTEGER;
  BEGIN
      l_sql :=  'UPDATE   items SET item_value =  :p_item_val '||
          ' WHERE item_id = :p_item_id RETURNING item_name INTO  :l_name '  ;
      l_cursor_id :=  DBMS_SQL.OPEN_CURSOR;
      DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_val', l_item_value);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_id', l_item_id);
      DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':l_name', l_item_name);
      l_return := DBMS_SQL.EXECUTE(l_cursor_id);
      DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':l_name', l_item_name);
      DBMS_OUTPUT.PUT_LINE('Rows Processed '||l_return||'Item Name '||l_item_name);
      DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
      COMMIT;
   END;
```

ORA-6502: PL/SQL: numeric or value error

Executing DML Statements

- Returning Into Clause

```
DECLARE
      l_item_value    items.item_value%TYPE    := 100;
      l_item_id         items.item_id%TYPE         := 1;
      l_item_name    items.item_name%TYPE := 'Maximum Item Length Name ';

      l_sql  VARCHAR2(200);
      l_cursor_id INTEGER;
      l_return  INTEGER;
  BEGIN
       l_sql :=  'UPDATE   items SET item_value =  :p_item_val '||
           ' WHERE item_id = :p_item_id RETURNING item_name INTO  :l_name '  ;
       l_cursor_id :=  DBMS_SQL.OPEN_CURSOR;
       DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
       DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_val', l_item_value);
       DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_id', l_item_id);
       DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':l_name', l_item_name);
       l_return := DBMS_SQL.EXECUTE(l_cursor_id);
       DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':l_name', l_item_name);
       DBMS_OUTPUT.PUT_LINE('Rows Processed '||l_return||'Item Name '||l_item_name);
       DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
       COMMIT;
   END;
```

Executing Procedures

```
CREATE OR REPLACE PROCEDURE calculate_tier
(p_act_id    IN        accounts.act_id%TYPE,
 p_act_bal  IN OUT accounts.act_bal%TYPE,
 p_tier              OUT NUMBER) IS
     ….
     ….
 END calculate_tier;
```

```
DECLARE
  l_act_id accounts.act_id%TYPE := 1;
  l_act_bal accounts.act_bal%TYPE;
  l_tier NUMBER;
  l_sql  VARCHAR2(200);
  l_cursor_id INTEGER;
  l_return  INTEGER;

BEGIN
   l_sql :=  ' CALL calculate_tier(:act_id,:act_bal,:tier) ';
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
   DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':act_id', l_act_id);
   DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':act_bal', l_act_bal);
   DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':tier', l_tier);
   l_return := DBMS_SQL.EXECUTE(l_cursor_id);
   DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':act_bal', l_act_bal);
   DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':tier', l_tier);
   DBMS_OUTPUT.PUT_LINE('Act Bal'||l_act_bal||'Tier: '||l_tier);
   DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

Executing Function With Anonymous Block

```
CREATE OR REPLACE FUNCTION get_tier
(p_act_id    IN        accounts.act_id%TYPE,
 p_act_bal  IN OUT accounts.act_bal%TYPE,
 p_tier              OUT NUMBER)
   RETURN NUMBER IS
     ….

 END get_tier;
```

```
DECLARE
 l_act_id    accounts.act_id%TYPE := 1;
 l_act_bal  accounts.act_bal%TYPE;
 l_tier NUMBER;
 l_out NUMBER;

 l_sql  VARCHAR2(200);
 l_cursor_id INTEGER;
 l_return  INTEGER;

BEGIN
 l_sql:=  ' BEGIN :l_out := get_tier(:act_id,:act_bal,:tier);  END; ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':act_id', l_act_id);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':act_bal', l_act_bal);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':tier', l_tier);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':l_out', l_out);
  l_return := DBMS_SQL.EXECUTE(l_cursor_id);
  DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':act_bal', l_act_bal);
  DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':tier', l_tier);
  DBMS_SQL.VARIABLE_VALUE(l_cursor_id, ':l_out', l_out);
  DBMS_OUTPUT.PUT_LINE('Act Bal'||l_act_bal||'Tier:  '||l_tier||
                                                 ' l_out '||l_out);
  DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

Executing Select Statements

Open Cursor

Parse

Bind Variable

Define Column

Execute

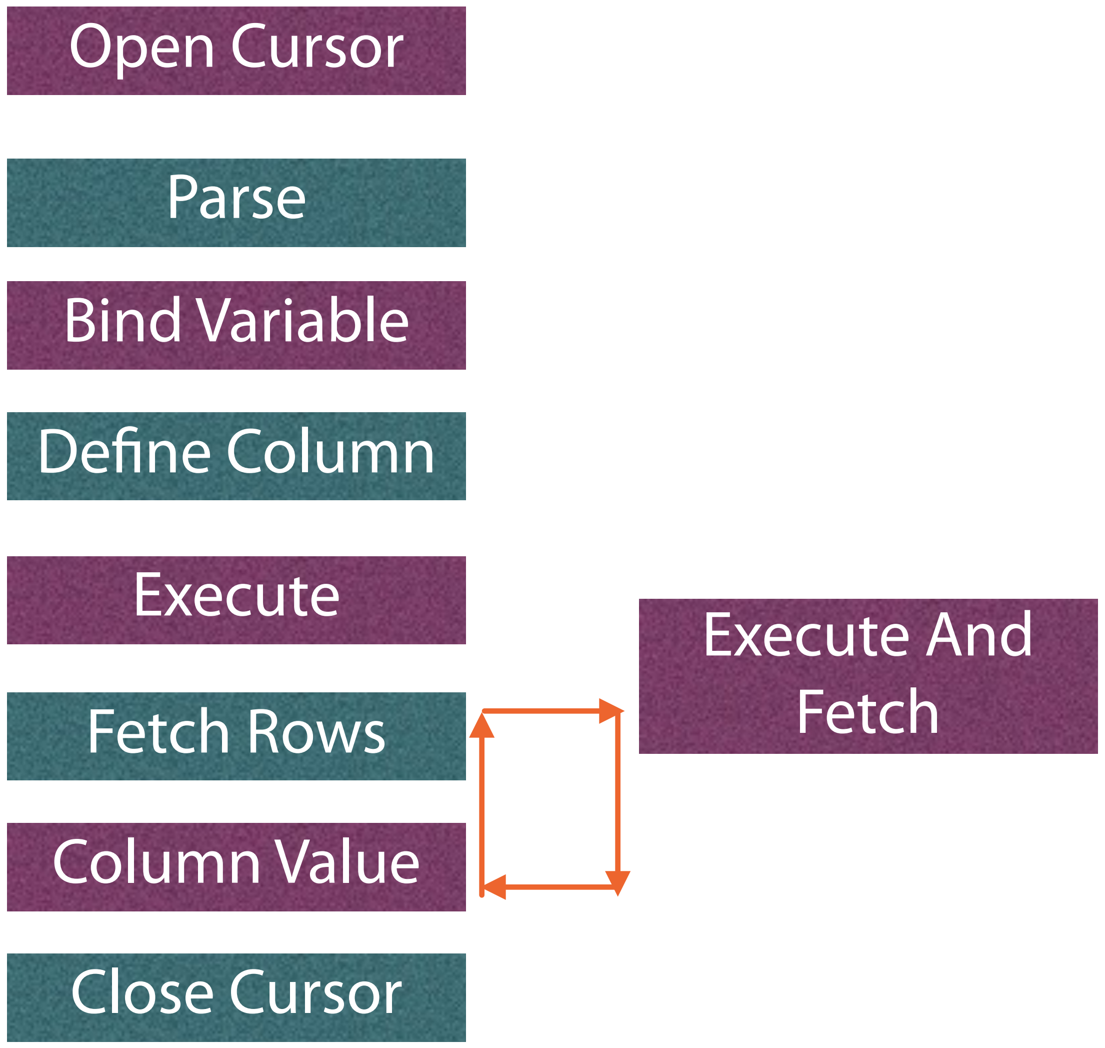
Fetch Rows

Variable Value

Column Value

Close Cursor

Executing Select Statements



Define Column

```
PROCEDURE  define_column(
                    c                IN  INTEGER,
                    position         IN  INTEGER,
                    column           IN VARCHAR2,
                    column_size      IN INTEGER);
```

```
PROCEDURE  define_column(
                    c                IN  INTEGER,
                    position         IN  INTEGER,
                    column           IN NUMBER);
```

```
PROCEDURE  define_column(
                    c                IN  INTEGER,
                    position         IN  INTEGER,
                    column           IN DATE);
```

Column Value

```
PROCEDURE  column_value(
               c              IN  INTEGER,
               position       IN  INTEGER,
               value          OUT VARCHAR2);
```

```
PROCEDURE  column_value(
               c              IN  INTEGER,
               position       IN  INTEGER,
               value          OUT VARCHAR2,
               column_error   OUT NUMBER,
               actual_length  OUT NUMBER);
```

Fetch Rows

```
FUNCTION fetch_rows( c  IN  INTEGER) RETURN INTEGER;
```

```
FUNCTION execute_and_fetch( c         IN  INTEGER,
                            exact IN  BOOLEAN DEFAULT FALSE)
          RETURN INTEGER;
```

Multi Row Select

```
DECLARE
 l_item_id              items.item_id%TYPE;
 l_item_name            items.item_name%TYPE;
 l_value                items.item_value%TYPE:= 50;
 l_sql                  VARCHAR2(200);
 l_cursor_id            INTEGER;
 l_return               INTEGER;

BEGIN
  l_sql:=  ' SELECT item_id, item_name FROM items WHERE item_value > :p_value ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_value', l_value);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 1, l_item_id);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 2, l_item_name, 100);
  l_return := DBMS_SQL.EXECUTE(l_cursor_id);
  LOOP
    IF DBMS_SQL.FETCH_ROWS(l_cursor_id) = 0 THEN
        exit;
     END IF;
     DBMS_SQL.COLUMN_VALUE(l_cursor_id, 1, l_item_id);
     DBMS_SQL.COLUMN_VALUE(l_cursor_id, 2, l_item_name);
     DBMS_OUTPUT.PUT_LINE('Item Id: ' ||l_item_id|| ' Item Name:  ' ||l_item_name);
   END LOOP;
  DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

Multi Row Select

```
DECLARE
 l_item_id          items.item_id%TYPE;
 l_item_name        items.item_name%TYPE;
 l_value            items.item_value%TYPE:= 50;
 l_sql              VARCHAR2(200);
 l_cursor_id        INTEGER;
 l_return           INTEGER;
BEGIN
  l_sql:=  ' SELECT item_id, item_name FROM items WHERE item_value > :p_value ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_value', l_value);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 1, l_item_id);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 2, l_item_name, 100);
  l_return := DBMS_SQL.EXECUTE(l_cursor_id);
  LOOP
    IF DBMS_SQL.FETCH_ROWS(l_cursor_id) = 0 THEN
       exit;
     END IF;
     DBMS_SQL.COLUMN_VALUE(l_cursor_id, 1, l_item_name);
     DBMS_SQL.COLUMN_VALUE(l_cursor_id, 2, l_item_name);
     DBMS_OUTPUT.PUT_LINE('Item Id: ' ||l_item_id|| ' Item Name:  ' ||l_item_name);
   END LOOP;
  DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

ORA-6562 type of out argument must match type of column or bind variable

LAST_ERROR_POSITION

```
FUNCTION  LAST_ERROR_POSITION RETURN INTEGER;
```

```
DECLARE
  l_errpos INTEGER;
 …
BEGIN
  l_sql:=  ' SELECT item_id, item_name ,  FROM items WHERE item_value > :p_value ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  …
 DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
EXCEPTION
 WHEN OTHERS THEN
    l_errpos := DBMS_SQL.LAST_ERROR_POSITION;
    DBMS_OUTPUT.PUT_LINE (SQLERRM || ' at pos ' || l_errpos);
    DBMS_SQL.CLOSE_CURSOR (l_cursor_id);
END;
```

```
ora-00936 missing expression at pos 28
```

Array Processing

```
PROCEDURE   bind_array(
              c               IN  INTEGER,
              name            IN  VARCHAR2,
              table_variable  IN table_datatype);
```

```
PROCEDURE   define_array(
              c               IN  INTEGER,
              position        IN  INTEGER,
              table_variable  IN table_datatype,
              cnt             IN INTEGER,
              index           IN  INTEGER);
```

DESCRIBE_COLUMNS

```
DBMS_SQL.DESCRIBE_COLUMNS (
                          c         IN    INTEGER,
                          col_cnt   OUT   INTEGER,
                          desc_t    OUT   DESC_TAB);
```

```
DBMS_SQL.DESCRIBE_COLUMNS2 (
                          c         IN    INTEGER,
                          col_cnt   OUT   INTEGER,
                          desc_t    OUT   DESC_TAB2);
```

```
DBMS_SQL.DESCRIBE_COLUMNS3 (
                          c         IN    INTEGER,
                          col_cnt   OUT   INTEGER,
                          desc_t    OUT   DESC_TAB3);
```

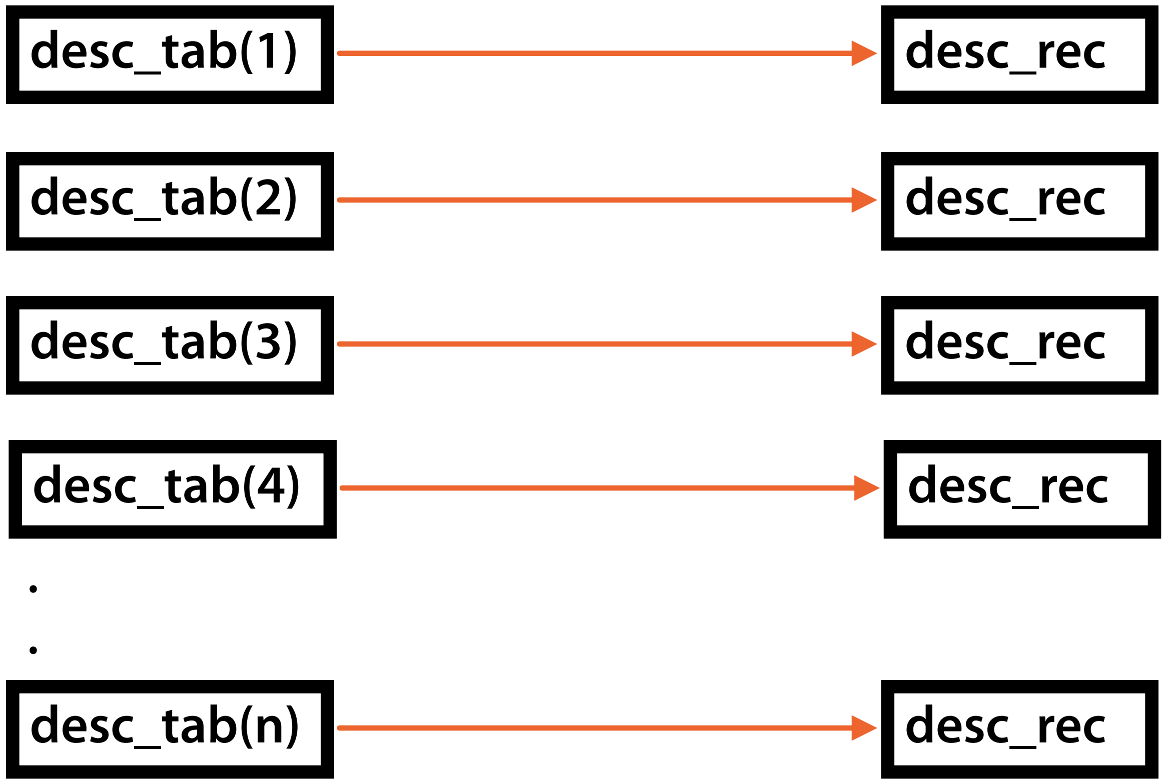
DESC_TAB

```
TYPE desc_tab IS TABLE OF desc_rec INDEX BY BINARY_INTEGER;
```

```
TYPE desc_tab2 IS TABLE OF desc_rec2 INDEX BY BINARY_INTEGER;
```

```
TYPE desc_tab3 IS TABLE OF desc_rec3 INDEX BY BINARY_INTEGER;
```

DESC_TAB


desc_tab(1)
desc_rec
desc_tab(2)
desc_rec
desc_tab(3)
desc_rec
desc_tab(4)
desc_rec
desc_tab(n)
desc_rec

DESC_REC

```
TYPE desc_rec IS RECORD (
  col_type              binary_integer := 0,
  col_max_len           binary_integer := 0,
  col_name              varchar2(32) := '',
  col_name_len          binary_integer := 0,
  col_schema_name       varchar2(32)   := '',
  col_schema_name_len binary_integer := 0,
  col_precision         binary_integer := 0,
  col_scale             binary_integer := 0,
  col_charsetid         binary_integer := 0,
  col_charsetform       binary_integer := 0,
  col_null_ok           boolean       := TRUE);
```

DESC_REC2

```
col_name  varchar2(32767) := ''
```

```
  col_type_name         varchar2(32767)   := '',
  col_type_name_len   binary_integer := 0);
```

DESC_REC3

Column Types

Column Type	Value
VARCHAR2	1
NVARCHAR	1
NUMBER	2
INTEGER	2
LONG	8
ROWID	11
DATE	12
RAW	23
LONG RAW	24
CHAR	96
NCHAR	96
MLSLABEL	106
CLOB (Oracle8)	112
NCLOB (Oracle8)	112
BLOB (Oracle8)	113
BFILE (Oracle8)	114
Object Type (Oracle8)	121
Nested Table Type (Oracle8)	122
Variable Array (Oracle8)	123

Unknown No of Select Columns

```
CREATE OR REPLACE PROCEDURE desc_columns (p_query VARCHAR2) AUTHID DEFINER IS
 l_cursor_id          INTEGER;
 l_no_of_columns  INTEGER;
 l_desc_tab2          DBMS_SQL.DESC_TAB2;
 l_desc_rec2          DBMS_SQL.DESC_REC2;
BEGIN
 l_cursor_id := DBMS_SQL.OPEN_CURSOR;
 dbms_sql.parse(l_cursor_id, p_query, DBMS_SQL.NATIVE);
 DBMS_SQL.DESCRIBE_COLUMNS2(l_cursor_id, l_no_of_columns, l_desc_tab2);
  FOR i IN 1 .. l_no_of_columns LOOP
    l_desc_rec2 := l_desc_tab2(i);
    DBMS_OUTPUT.PUT_LINE('Column Name '||l_desc_rec2.col_name);
    DBMS_OUTPUT.PUT_LINE('Column Type '||l_desc_rec2.col_type);
 END LOOP;
 DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END desc_columns;
```

```
EXEC desc_columns('SELECT order_act_id, item_name FROM orders, items WHERE order_item_id = item_id');
```

```
Column Name ORDER_ACT_ID
Column Type 2
Column Name ITEM_NAME
Column Type 1
```

Unknown No of Select Columns

```
CREATE OR REPLACE PROCEDURE desc_columns (p_query VARCHAR2, p_key VARCHAR2, p_value VARCHAR2)
AUTHID DEFINER IS
  l_cursor_id         INTEGER;
  l_return            INTEGER;
  l_no_of_columns     INTEGER;
  l_desc_tab2         DBMS_SQL.DESC_TAB2;
  l_desc_rec2         DBMS_SQL.DESC_REC2;
  l_number            NUMBER;
  l_date              DATE;
  l_varchar2          VARCHAR2(100);
BEGIN
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.parse(l_cursor_id, p_query, DBMS_SQL.NATIVE);
  DBMS_SQL.DESCRIBE_COLUMNS2(l_cursor_id, l_no_of_columns, l_desc_tab2);
  -- Define columns
  FOR i IN 1 .. l_no_of_columns LOOP
      l_desc_rec2 := l_desc_tab2(i);
      IF l_desc_rec2.col_type = 2 THEN
        DBMS_SQL.DEFINE_COLUMN(l_cursor_id, i, l_number);
      ELSIF l_desc_rec2.col_type = 12 THEN
        DBMS_SQL.DEFINE_COLUMN(l_cursor_id, i, l_date);
     ELSE
       DBMS_SQL.DEFINE_COLUMN(l_cursor_id, i, l_varchar2, 100);
    END IF;
  END LOOP;
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, p_key, p_value);
   l_return := DBMS_SQL.EXECUTE(l_cursor_id);
……
END desc_columns;
```

DBMS_SQL Security Aspects

- **Invalid Cursor Check**

Demo User Session

```
DECLARE
 …
BEGIN
  l_sql:= ' SELECT item_id, item_name FROM items WHERE ' ||
            ' item_value > :p_value ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  dbms_output.put_line('Cursor id is '||l_cursor_id);
  …
  LOOP
   IF DBMS_SQL.FETCH_ROWS(l_cursor_id) = 0 THEN
    exit;
   END IF;
   DBMS_SQL.COLUMN_VALUE(l_cursor_id, 1, l_item_name);
   …
  END LOOP;
  CLOSE_CURSOR;
 EXCEPTION
  WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Inside Exception Section '||SQLERRM);
  RAISE;
END;
```

1655307019

Hacker

```
DECLARE
  l_sql VARCHAR2(200);
  l_return INTEGER;
BEGIN
   l_sql:= ' DELETE FROM items ';
   DBMS_SQL.PARSE(1655307019,
                          l_sql, DBMS_SQL.NATIVE);
   l_return :=
DBMS_SQL.EXECUTE(1655307019);
END;
```

```
ORA-29471: DBMS_SQL access denied
```

DBMS_SQL Security Aspects

- **Random Cursor Number Generation**

```
FUNCTION OPEN_CURSOR(security_level IN INTEGER) RETURN INTEGER;
```

- **Open Cursor**
  - 0 No security check
  - 1 Userid / Role Parsing Be the Same as Binding / Executing
  - 2 Most Secure

- **Checks**
  - Current Calling User Same As the Recent Parse User
  - Enabled Roles on Current Call Same As Enabled Roles on Recent Parse
  - Container on Current Call Same As Container on Recent Parse

```
ORA-29470: Effective userid or roles are not the same as when cursor was parsed
```

DBMS_SQL Security Aspects

Demo User Session

```
CREATE OR REPLACE FUNCTION get_count_cursor RETURN
NUMBER AUTHID DEFINER IS
  l_sql VARCHAR2(200);
  l_cursor_id INTEGER;
 BEGIN
  l_sql:= ' SELECT count(*) FROM orders ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR(2);
  DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
  RETURN l_cursor_id;
 END get_count_cursor;
```

Test

```
DECLARE
  l_cursor_id INTEGER;
  l_count     INTEGER;
  l_return INTEGER;
 BEGIN
   l_cursor_id := demo.get_count_cursor;
   DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 1, l_count);
   …
  WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Inside Exception Section '||SQLERRM);
  RAISE;
END;
```

ORA-29470: Effective userid or roles are not the same as when cursor was parsed

DBMS_SQL vs Native Dynamic SQL

DBMS_SQL

```
CREATE OR REPLACE PROCEDURE drop_table (p_table_name VARCHAR2)  IS
   l_sql  VARCHAR2(100);
   l_cursor_id INTEGER;
   l_return  INTEGER;
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   l_cursor_id := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
   l_return := DBMS_SQL.EXECUTE(l_cursor_id);
   DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END drop_table;
```

Native Dynamic SQL

```
CREATE OR REPLACE PROCEDURE drop_table  (p_table_name  VARCHAR2) IS
   l_sql  VARCHAR2(100);
 BEGIN
   l_sql := 'DROP TABLE '||p_table_name;
   EXECUTE IMMEDIATE  l_sql ;
END drop_table;
```

DBMS_SQL vs Native Dynamic SQL

DBMS_SQL

```
DECLARE
 l_item_id            items.item_id%TYPE;
 l_item_name          items.item_name%TYPE;
 l_value              items.item_value%TYPE:= 50;
 l_sql                VARCHAR2(200);
 l_cursor_id          INTEGER;
 l_return             INTEGER;

BEGIN
  l_sql:=  ' SELECT item_id, item_name FROM items WHERE item_value > :p_value ';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(l_cursor_id, l_sql,DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_value', l_value);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 1, l_item_id);
  DBMS_SQL.DEFINE_COLUMN(l_cursor_id, 2, l_item_name, 100);
  l_return := DBMS_SQL.EXECUTE(l_cursor_id);
  LOOP
    IF DBMS_SQL.FETCH_ROWS(l_cursor_id) = 0 THEN
        exit;
     END IF;
     DBMS_SQL.COLUMN_VALUE(l_cursor_id, 1, l_item_id);
     DBMS_SQL.COLUMN_VALUE(l_cursor_id, 2, l_item_name);
     DBMS_OUTPUT.PUT_LINE('Item Id: ' ||l_item_id|| ' Item Name:  ' ||l_item_name);
   END LOOP;
  DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

DBMS_SQL vs Native Dynamic SQL

Native Dynamic SQL

```
DECLARE
 l_item_id            items.item_id%TYPE;
 l_item_name          items.item_name%TYPE;
 l_value              items.item_value%TYPE:= 50;
 l_sql                VARCHAR2(200);
 l_ref_cursor         SYS_REFCURSOR;

BEGIN
  l_sql:=  ' SELECT item_id, item_name FROM items WHERE item_value > :p_value ';
  OPEN l_ref_cursor FOR l_sql USING l_value;
   LOOP
      FETCH l_ref_cursor INTO l_item_id, l_item_name;
      DBMS_OUTPUT.PUT_LINE('Item Id: ' ||l_item_id|| ' Item Name:  ' ||l_item_name);
      EXIT WHEN l_ref_cursor%NOTFOUND;
END;
```

When Use DBMS_SQL?

Unknown Number of Select Columns

Unknown Number of PlaceHolders

Interoperability

```
DBMS_SQL.TO_REFCURSOR(cursor_number IN OUT INTEGER)
  RETURN SYS_REFCURSOR;
```

```
DECLARE
  l_cursor_id  INTEGER;
  l_sql             VARCHAR2(200);
  l_ref_cursor  SYS_REFCURSOR;
  l_items_rec     items%ROWTYPE;
BEGIN
  l_sql:=  ' SELECT * FROM items WHERE item_value = :p_item_value';
  l_cursor_id := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(l_cursor_id, l_sql, DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE(l_cursor_id, ':p_item_value' ,100);
  DBMS_SQL.EXECUTE(l_cursor_id);
  l_ref_cursor := DBMS_SQL.TO_REFCURSOR(l_cursor_id);
   LOOP
     FETCH l_ref_cursor INTO l_items_rec;
     EXIT WHEN l_ref_cursor%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE('Item Name is  '||l_items_rec.item_name);
   END LOOP;
   CLOSE l_ref_cursor;
END;
```

Interoperability

```
DBMS_SQL.TO_CURSOR_NUMBER(rc IN OUT SYS_REFCURSOR)
 RETURN INTEGER;
```

```
CREATE OR REPLACE PROCEDURE getinfo(p_query VARCHAR2) IS
  l_cursor_id     INTEGER;
  l_ref_cursor    SYS_REFCURSOR;
  l_col_count     INTEGER;
  l_desc_tab2     DBMS_SQL.DESC_TAB2;
  l_desc_rec2     DBMS_SQL.DESC_REC2;
  l_return        INTEGER;
BEGIN
  OPEN l_ref_cursor FOR p_query;
  l_cursor_id := dbms_sql.to_cursor_number(l_ref_cursor);
  DBMS_SQL.DESCRIBE_COLUMNS2(l_cursor_id, l_col_count, l_desc_tab2);
          FOR i IN 1 .. l_col_count LOOP
          l_desc_rec2 := l_desc_tab2(i);
          DBMS_OUTPUT.PUT_LINE('Column Name '||l_desc_rec2.col_name);
          DBMS_OUTPUT.PUT_LINE('Column Type '||l_desc_rec2.col_type);
  END LOOP;
  DBMS_SQL.CLOSE_CURSOR(l_cursor_id);
END;
```

```
EXEC getinfo('SELECT order_item_id, order_act_id FROM orders');
```

Summary

What Is DBMS_SQL?

Usage

Security Implications