Package dbms_sql

- Execute dynamically created SQL statements in PL/SQL.
- These statements are created as a character string and are sent to Oracle for syntax checking.
- After they are verified to be syntactically correct, they are sent for execution.

Executing non-query statements

The steps involved to execute an `insert`, `delete` or `update` statement are as follows:

1. **Open the cursor**.

```
handle := DBMS_SQL.OPEN_CURSOR;
```

2. **Parse the statement**.

```
procedure parse(handle IN INTEGER,
                stmt IN VARCHAR2,
                language IN INTEGER);
```

`language` is one of the following constants

- `DBMS_SQL.V6` for Oracle Version 6.
- `DBMS_SQL.V7` for Oracle Version 7.
- `DBMS_SQL.NATIVE` for database to which the program is connected.

3. Bind any input variables (place holders).

```
  procedure bind_variable(handle IN INTEGER,
                          name IN VARCHAR2,
                          value IN NUMBER);
  procedure bind_variable(handle IN INTEGER,
                          name IN VARCHAR2,
                          value IN VARCHAR2);
  procedure bind_variable(handle IN INTEGER,
                          name IN VARCHAR2,
                          value IN DATE);
  procedure bind_variable_char(handle IN INTEGER,
                               name IN VARCHAR2,
                               value IN CHAR);
```

Examples:

```
  DBMS_SQL.BIND_VARIABLE(handle,':n',10);
  DBMS_SQL.BIND_VARIABLE(handle,':n',enum);
  DBMS_SQL.BIND_VARIABLE(handle,':c','Jones');
  DBMS_SQL.BIND_VARIABLE(handle,':c',ename);
  DBMS_SQL.BIND_VARIABLE(handle,':d',hdate);
```

4. Execute the statement.

```
  nrows := DBMS_SQL.EXECUTE(handle);
```

5. Close the cursor.

```
  DBMS_SQL.CLOSE_CURSOR(handle);
```

Executing a dynamic `update`

```
DECLARE
  handle INTEGER;
  stmt varchar2(256);
  discount parts.price%type := 0.8;
  part_number parts.pno%type := 10506;
  nrows INTEGER;
BEGIN

  stmt := 'update parts set price = price * :fract' ||
          ' where pno = :pnum';

  handle := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(handle,stmt,DBMS_SQL.V7);
  DBMS_SQL.BIND_VARIABLE(handle,':fract',discount);
  DBMS_SQL.BIND_VARIABLE(handle,':pnum',part_number);
  nrows := DBMS_SQL.EXECUTE(handle);
  dbms_output.put_line('Number of rows updated = ' ||
                       nrows);
  DBMS_SQL.CLOSE_CURSOR(handle);

  EXCEPTION
    WHEN OTHERS THEN
      DBMS_SQL.CLOSE_CURSOR(handle);
END;
/
show errors
```

Dynamic `create, drop` etc.

- Place holders are not allowed in such statements.
- So, BIND_VARIABLE and EXECUTE are not necessary since these statements are executed immediately after PARSE automatically.

```
DECLARE
  handle INTEGER;
  stmt1 varchar2(256);
  stmt2 varchar2(256);
BEGIN
  stmt1 := 'drop table zzz';
  stmt2 := 'create table zzz (col1 integer,' ||
           ' col2 number(4), col3 varchar2(30))';
  handle := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(handle,stmt1,DBMS_SQL.V7);
  dbms_output.put_line('Table ZZZ dropped');
  DBMS_SQL.PARSE(handle,stmt2,DBMS_SQL.V7);
  dbms_output.put_line('Table ZZZ created');
  DBMS_SQL.CLOSE_CURSOR(handle);
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_SQL.CLOSE_CURSOR(handle);
END;
/
show errors
```

Executing queries

The sequence of steps required to process a dynamic query is as follows:

1. **Open the cursor**. Same as before.
2. **Parse the statement**. Same as before.
3. **Bind any input variables**. Same as before.

4. **Define the output variables**.

```
  procedure define_column(handle IN INTEGER,
                          position IN INTEGER,
                          column IN NUMBER);
  procedure define_column(handle IN INTEGER,
                          position IN INTEGER,
                          column IN VARCHAR2,
                          col_size IN INTEGER);
  procedure define_column(handle IN INTEGER,
                          position IN INTEGER,
                          column IN DATE,
                          col_size IN INTEGER);
  procedure define_column_char(handle IN INTEGER,
                          position IN INTEGER,
                          column IN CHAR,
                          col_size IN INTEGER);
```

Examples:

```
  DBMS_SQL.DEFINE_COLUMN(handle,1,ccno);
  DBMS_SQL.DEFINE_COLUMN(handle,2,ccname,30);
  DBMS_SQL.DEFINE_COLUMN(handle,3,ccstreet,30);
  DBMS_SQL.DEFINE_COLUMN(handle,4,ccdate,9);
```

5. **Execute the query**. Same as before.

6. **Fetch the rows**.

```
if DBMS_SQL.FETCH_ROWS(handle) = 0 then
      exit;
else
  ...
```

The function returns the number of rows in the result of the query.

7. **Return the results to PL/SQL variables**.

```
procedure column_value(handle IN INTEGER,
                       position IN INTEGER,
                       value OUT NUMBER);
procedure column_value(handle IN INTEGER,
                       position IN INTEGER,
                       value OUT VARCHAR2);
procedure column_value(handle IN INTEGER,
                       position IN INTEGER,
                       value OUT DATE);
procedure column_value(handle IN INTEGER,
                       position IN INTEGER,
                       value OUT CHAR);
```

Examples:

```
DBMS_SQL.COLUMN_VALUE(handle,1,ccno);
DBMS_SQL.COLUMN_VALUE(handle,2,ccname);
DBMS_SQL.COLUMN_VALUE(handle,3,ccstreet);
DBMS_SQL.COLUMN_VALUE(handle,4,ccdate);
```

8. **Close the cursor**. Same as before.

Example

A package, called `dsql`, is defined which contains two procedures:

- `get_columns`. This procedure takes as input `startch`, a character and returns information about all the columns of any database table in the database whose name starts with `startch`.
- `get_query_results`. This procedure takes as input
  - `query_string`, which has as its value an SQL select statement,
  - `ncols`, the number of select list items in the SQL select statement, and
  - `column_types`, a PL/SQL table that contains the data type of each of the select list items.

  It then executes the query and returns the results in the output parameter `result` along with the number of rows in the result in variable `nrows`.

  The `result` output parameter is a PL/SQL table of strings. The individual columns in a row of the result of the query are concatenated into a long string with the bar character | between two values.

```
-- p20.sql
CREATE OR REPLACE PACKAGE dsql AS
  TYPE dd_rec_type IS RECORD (
    table_name varchar2(30),
    column_name varchar2(30),
    data_type  varchar2(9));
  TYPE dd_table_type IS TABLE OF dd_rec_type
       INDEX BY BINARY_INTEGER;
  TYPE string1024_table IS TABLE OF varchar2(1024)
       INDEX BY BINARY_INTEGER;
  TYPE string9_table IS TABLE OF varchar2(9)
       INDEX BY BINARY_INTEGER;
  TYPE number_table IS TABLE OF number
       INDEX BY BINARY_INTEGER;

PROCEDURE get_columns(startch IN char,
                      dd_table OUT dd_table_type,
                      n OUT number);

procedure get_query_results(query_string IN varchar2,
                            ncols IN number,
                            column_types IN string9_table,
                            result OUT string1024_table,
                            nrows OUT number);

END dsql;
/
show errors
```

```
CREATE OR REPLACE PACKAGE BODY dsql AS

PROCEDURE get_columns(startch in char,
                      dd_table OUT dd_table_type,
                      n OUT number) AS
handle Integer;
dbms_return Integer;
tablename varchar(30);
colname varchar(30);
datatype varchar(9);
counter integer := 0;

begin

  handle := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(handle,
    'select distinct table_name,column_name,data_type ' ||
    'from   user_tab_columns '                          ||
    'where  table_name like '''                         ||
    startch                                          ||
    '%'''                                            ||
    'order by table_name,column_name', DBMS_SQL.V7);

  DBMS_SQL.DEFINE_COLUMN(handle, 1, tablename, 30);
  DBMS_SQL.DEFINE_COLUMN(handle, 2, colname, 30);
  DBMS_SQL.DEFINE_COLUMN(handle, 3, datatype, 9);
```

```
  dbms_return := DBMS_SQL.EXECUTE(handle);

  counter := 0;
  loop
    if DBMS_SQL.FETCH_ROWS(handle) = 0 then
      exit;
    else
      DBMS_SQL.COLUMN_VALUE(handle, 1, tablename);
      DBMS_SQL.COLUMN_VALUE(handle, 2, colname);
      DBMS_SQL.COLUMN_VALUE(handle, 3, datatype);

      counter := counter + 1;
      dd_table(counter).table_name := tablename;
      dd_table(counter).column_name := colname;
      dd_table(counter).data_type := datatype;
    end if;
  end loop;
  n := counter;
  DBMS_SQL.CLOSE_CURSOR(handle);

EXCEPTION
  WHEN OTHERS THEN
    DBMS_SQL.CLOSE_CURSOR(handle);
    n := -1;
END get_columns;
```

```
procedure get_query_results(query_string IN varchar2,
                            ncols IN number,
                            column_types IN string9_table,
                            result OUT string1024_table,
                            nrows OUT number) AS

y1 string1024_table;
y2 number_table;
counter Integer;
i Integer;
handle Integer;
dbms_return Integer;
temp string1024_table;

BEGIN

  for i in 1 .. ncols loop
    y1(i) := '';
    y2(i) := 0;
  end loop;

  counter := 0;
  handle := DBMS_SQL.OPEN_CURSOR;
  DBMS_SQL.PARSE(handle,query_string,DBMS_SQL.V7);

  for i in 1 .. ncols loop
    if ((column_types(i) = 'VARCHAR2') or
        (column_types(i) = 'CHAR')) then
      DBMS_SQL.DEFINE_COLUMN(handle, i, y1(i) , 300);
    else
      DBMS_SQL.DEFINE_COLUMN(handle, i, y2(i));
    end if;
  end loop;
```

```
dbms_return := DBMS_SQL.EXECUTE(handle);

loop
   if DBMS_SQL.FETCH_ROWS(handle) = 0 then
     exit;
   else
     for i in 1 .. ncols loop
       if ((column_types(i) = 'VARCHAR2') or
           (column_types(i) = 'CHAR')) then
         DBMS_SQL.COLUMN_VALUE(handle, i, y1(i));
       else
         DBMS_SQL.COLUMN_VALUE(handle, i, y2(i));
       end if;
     end loop;
     counter := counter + 1;
     temp(counter) := '';

     for i in 1 .. ncols loop
       if ((column_types(i) = 'VARCHAR2') or
           (column_types(i) = 'CHAR')) then
         temp(counter) := temp(counter) ||
                          rtrim(y1(i)) || '|';
       else
         temp(counter) := temp(counter) ||
                          rtrim(y2(i)) || '|';
       end if;
     end loop;
 end if;
end loop;
```

```
  nrows := counter;
  result := temp;
  DBMS_SQL.CLOSE_CURSOR(handle);

  EXCEPTION
    WHEN OTHERS THEN
      DBMS_SQL.CLOSE_CURSOR(handle);
      nrows := -1;
END get_query_results;
end dsql;
/
show errors
```

The following package, called `dsql_driver` contains two procedures to drive the two procedures in the `dsql` package.

```
-- p21.sql
CREATE OR REPLACE PACKAGE dsql_driver AS
procedure drive_get_columns(startch IN char);
procedure drive_get_query_results(
    query_string IN varchar2,
    ncols IN number,
    column_types IN dsql.string9_table);
END dsql_driver;
/

CREATE OR REPLACE PACKAGE BODY dsql_driver AS
procedure drive_get_columns(startch IN char) AS
  dd_table dsql.dd_table_type;
  i binary_integer;
  n number;
  prev varchar2(30);
  newentry boolean;
BEGIN
  dsql.get_columns(startch,dd_table,n);
  dbms_output.put_line('N =  ' || n);

  dbms_output.put_line('Tables Starting with ' ||
                       startch);
  dbms_output.put_line('------------------------');
```

```
  prev := '';
  newentry := false;
  for i in 1 .. dd_table.count loop
    if (i = 1) then
      dbms_output.put(dd_table(i).table_name || '(');
      prev := dd_table(i).table_name;
    elsif (prev != dd_table(i).table_name) then
      dbms_output.put_line(');');
      dbms_output.put(dd_table(i).table_name || '(');
      prev := dd_table(i).table_name;
      newentry := false;
    end if;
    if (newentry) then
      dbms_output.put(', ');
    end if;
    newentry := true;
    dbms_output.put(dd_table(i).column_name || ':');
    dbms_output.put(dd_table(i).data_type);
  end loop;
  dbms_output.put_line(');');
END;
```

```
procedure drive_get_query_results
    (query_string IN varchar2,
     ncols IN number,
     column_types IN dsql.string9_table) AS
result dsql.string1024_table;
nrows number;
i binary_integer;

BEGIN
  dsql.get_query_results(query_string,ncols,column_types,
                         result,nrows);
  for i in 1 .. nrows loop
    dbms_output.put_line(result(i));
  end loop;
END;

END dsql_driver;
/
show errors
```

The following anonymous PL/SQL block calls the driver procedure for the `get_query_results` procedure.

```
-- p22.sql
DECLARE
 query_string varchar2(256);
 ncols number;
 column_types dsql.string9_table;
BEGIN
  query_string := 'select cno,cname,city ' ||
                  'from customers,zipcodes ' ||
                  'where customers.zip = zipcodes.zip';
  ncols := 3;
  column_types(1) := 'NUMBER';
  column_types(2) := 'VARCHAR2';
  column_types(3) := 'VARCHAR2';

  dsql_driver.drive_get_query_results(query_string,
                                      ncols,
                                      column_types);
END;
/
```

The following is a sample SQL*PLUS session to illustrate the above example.

```
SQL> execute display_table('customers');
1111 Charles 123 Main St. 67226 316-636-5555
2222 Bertram 237 Ash Avenue 67226 316-689-5555
3333 Barbara 111 Inwood St. 60606 316-111-1234

PL/SQL procedure successfully completed.

SQL> execute display_table('abcd');
NOT A VALID TABLE NAME

PL/SQL procedure successfully completed.

SQL> start p22
1111|Charles|Wichita|
2222|Bertram|Wichita|
3333|Barbara|Fort Dodge|

PL/SQL procedure successfully completed.

SQL>
```

PL/SQL Cartridge: `OWA_UTIL` package

- `tablePrint` procedure: used to print database tables in a Web page.

```
bool := owa_util.tablePrint(
  ctable in varchar2
  cattributes in varchar2 DEFAULT NULL
  ntable_type in integer DEFAULT HTML_TABLE
  ccolumns in varchar2 DEFAULT '*'
  cclauses in varchar2 DEFAULT NULL
  ccol_aliases in varchar2 DEFAULT NULL
  nrow_min in number DEFAULT 0
  nrow_max in number DEFAULT NULL);
```

`ctable` : the name of the table to be printed,
`cattributes` : a HTML table attribute,
`ntable_type` : either `owa_util.html_table` or `owa_util.pre_table`,
`ccolumns` is a comma separated list of columns to be printed,
`cclauses` is a selection condition,
`ccol_aliases` is any aliases for the columns,
`nrow_min` : minimum number of rows to be printed
`nrow_max` : maximum number of rows to be printed.

The function returns True or False as to whether there are more rows available

beyond the `nrow_max` requested.

```
create or replace package was_tprint as
  procedure tprint1;
  procedure tprint2;
  procedure tprint3;
end was_tprint;
create or replace package body was_tprint as
procedure tprint1 as
   ignore boolean;
begin
 ignore := owa_util.tablePrint
      ('courses', 'BORDER', OWA_UTIL.PRE_TABLE,
       'cno, lineno',
       'where term=''F96'' order by lineno',
       'Course Number, Line Number');
end;
procedure tprint2 as
   ignore boolean;
begin
 ignore := owa_util.tablePrint
      ('courses', 'BORDER', OWA_UTIL.HTML_TABLE,
       'cno, lineno',
       'where term=''F96'' order by lineno',
       'Course Number, Line Number');
end;
procedure tprint3 as
   ignore boolean;
begin
 ignore := owa_util.tablePrint
      ('courses', 'BORDER', OWA_UTIL.HTML_TABLE);
end;
end was_tprint;
```

OWA_UTIL: bind_variables

The `bind_variables` function in the `OWA_UTIL` package prepares a SQL query (binding variables to it, if any), and stores the output in an opened cursor. The integer returned by the function is a unique identifier for that cursor. The syntax is as follows:

```
n := owa_util.bind_variables(
  theQuery,
  bv1Name, bv1Value,
  ...
  bv25Name, bv25Value, bv25Name);
```

where `theQuery` is a string variable containing a SQL query, possibly with bind variables, and the remaining parameters are the bind variable names and their values (there is a maximum limit of 25 such pairs).

The results of the query can be printed using the `cellsprint` procedure.

```
owa_util.cellsprint(
  p_theCursor in integer
  p_max_rows in number DEFAULT 100
  p_format_numbers in varchar2 DEFAULT NULL);
```

This procedure generates an HTML table from the output of a SQL query. SQL atomic data items are mapped to HTML cells and SQL rows to HTML rows. The code to begin and end the HTML table must be written separately. The `p_theCursor` is the output of a call made to the bind variables procedure.
There are 4 versions of this procedure; please see online documentation.

- Web page with a pull down list of courses.
- The user may select one of the courses
- Upon submission of the form, the second procedure prints the students enrolled in the particular course.
- The query to obtain the enrolled students is assigned to a string variable and has three bind variables.

```
create or replace package was_dsql as
  procedure dsql1;
  procedure dsql2(tcl in varchar2);
end was_dsql;
/
show errors;
```

```
create or replace package body was_dsql as
procedure dsql1 as
  cursor c1 is
    select cno,term,lineno
    from   courses;
begin
   htp.htmlOpen;
   htp.headOpen;
   htp.title('Select Course');
   htp.header(1,'Display Enrolled Students');
   htp.headClose;
   htp.bodyOpen;
   htp.FormOpen(owa_util.get_owa_service_path ||
                'was_dsql.dsql2');
   htp.nl;
   htp.formSelectOpen('tcl','Select Course: ');
   for c1_rec in c1 loop
     htp.formSelectOption('(' ||c1_rec.term||','||
                          c1_rec.cno||','||
                          c1_rec.lineno || ')');
   end loop;
   htp.formSelectClose;
   htp.formSubmit(null,'Display Enrolled Students');
   htp.formClose;
   htp.bodyClose;
   htp.htmlClose;
end dsql1;
```

```
procedure dsql2(tcl in varchar2) as
  query varchar2(1024);
  cursor_id integer;
  p1 integer;
  p2 integer;
  trm  courses.term%type;
  cnum courses.cno%type;
  lnum courses.lineno%type;
begin
   p1 := instr(tcl,',',1,2)+1;
   p2 := instr(tcl,')',1,1)-1;
   trm := substr(tcl,2,instr(tcl,',',1)-2);
   lnum := to_number(substr(tcl,p1,p2-p1+1));
   cnum := substr(tcl,instr(tcl,',',1,1)+1,
                     p1-instr(tcl,',',1,1)-2);
   htp.htmlOpen;
   htp.headOpen;
   htp.title('Dynamic SQL Utilities');
   htp.headClose;
   htp.bodyOpen;
   htp.line;
   htp.header(1,'Students enrolled in');
   htp.print(cnum); htp.br;
   htp.print(trm); htp.br;
   htp.print(lnum); htp.br;
   htp.line;
```

```
   query :=
     'select students.sid,fname,lname ' ||
     'from enrolls,courses,students ' ||
     'where enrolls.lineno = courses.lineno and ' ||
     '      enrolls.term = courses.term and ' ||
     '      students.sid = enrolls.sid and ' ||
     '      cno = :cc and ' ||
     '      courses.term = :tt and ' ||
     '      courses.lineno = :ll';
   cursor_id :=
     owa_util.bind_variables(query,
        'cc',cnum,'tt',trm,'ll',lnum);
   htp.tableOpen(cborder => 'BORDER=1');
   owa_util.cellsprint(cursor_id);
   htp.tableClose;
   htp.bodyClose;
   htp.htmlClose;
end dsql2;
end was_dsql;
/
show errors
```

Notice that explicit statements are written to open and close the HTML table, as the `cellsprint` procedure generates only the code for the rows of the HTML table.