Using Dynamic SQL

Execution Flow of SQL

- All SQL statements go through some or all of the following stages:
  - Parse
  - Bind
  - Execute
  - Fetch

- Parse : check the statement syntax , validating the statement

Ensure all referencing objects are correct , The privileges exists

- Bind : check the bind variable if the statement contains Bind Var.
- Execute: execute the statement(non Queries statements)
- Fetch : retrieve the rows (Queries statements)

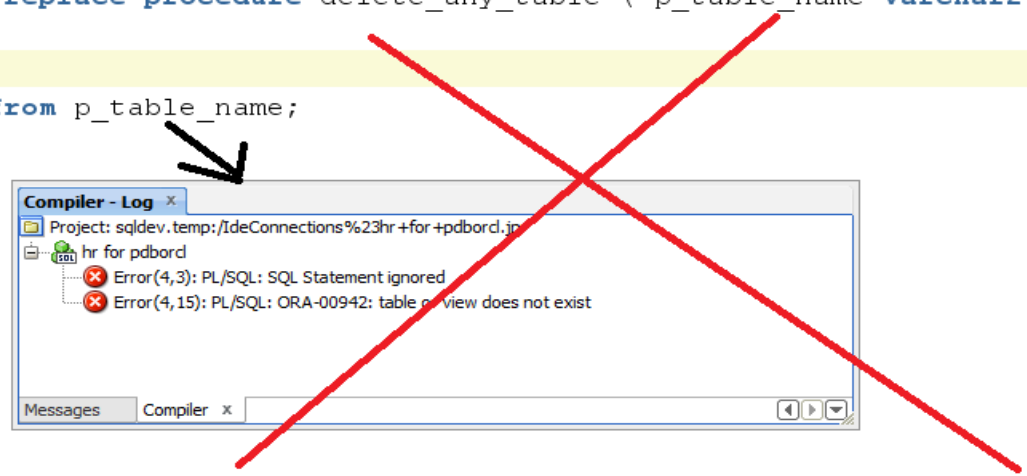
What Is Dynamic SQL???

The SQL statements are created dynamically at run time
(not compile time)

Example: creating a procedure for deleting any table

You may think to do this:

```
create or replace procedure delete_any_table ( p_table_name varchar2)
is
begin
  delete from p_table_name;
  commit;
end;
```



The solution is to use dynamic SQL

- execute immediate (native Dynamic)
- DBMS_SQL

What Is Dynamic SQL?

Use dynamic SQL to create a SQL statement whose structure may change during run time. Dynamic SQL:

- Is constructed and stored as a character string, string variable, or string expression within the application.
- Is a SQL statement with varying column data, or different conditions with or without placeholders (bind variables).
- Enables DDL, DCL, or session-control statements to be written and executed from PL/SQL.
- Is executed with Native Dynamic SQL statements or the `DBMS_SQL` package.

What Is Dynamic SQL?

- The full text of the dynamic SQL statement might be unknown until run time; therefore, its syntax is checked at *run time* rather than at *compile time*.
- You can use dynamic SQL statements to make your PL/SQL programs more general and flexible.

When Do You Need Dynamic SQL?

In PL/SQL, you need dynamic SQL in order to execute the following:

- SQL whose full text is unknown at compile time such as:
  - A `SELECT` statement that includes an identifier that is unknown at compile time (such as a table name)
  - A `WHERE` clause in which the column name is unknown at compile time

Native Dynamic SQL (NDS)

- Provides native support for dynamic SQL directly in the PL/SQL language.
- Provides the ability to execute SQL statements whose structure is unknown until execution time.
- If the dynamic SQL statement is a `SELECT` statement that returns multiple rows, NDS gives you the following choices:
  - Use the `EXECUTE IMMEDIATE` statement with the `BULK COLLECT INTO` clause
  - Use the `OPEN-FOR`, `FETCH`, and `CLOSE` statements
- In Oracle Database 11*g*, NDS supports statements larger than 32 KB by accepting a `CLOB` argument.

Use the `EXECUTE IMMEDIATE` statement for NDS or PL/SQL anonymous blocks:

```
EXECUTE IMMEDIATE dynamic_string
  [INTO {define_variable
      [, define_variable] ... | record}]
  [USING [IN|OUT|IN OUT] bind_argument
      [, [IN|OUT|IN OUT] bind_argument] ... ];
```

- `INTO` is used for single-row queries and specifies the variables or records into which column values are retrieved.
- `USING` is used to hold all bind arguments. The default parameter mode is `IN`.

Execute immediate as dynamic string

```
create or replace procedure delete_any_table
( p_table_name varchar2)
is
v_no_rec number;
begin
execute immediate 'delete from '||p_table_name;
commit; --same rules for commit and rollback
v_no_rec:=sql%rowcount;
dbms_output.put_line(v_no_rec ||' record(s) deleted form '||p_table_name );
end;
```

```
execute delete_any_table('empl');
select * from empl;
```

maxvlearn

Execute immediate as dynamic string -USING

```
create or replace procedure add_rows
( p_table_name varchar2,p_value number )
is
begin
EXECUTE IMMEDIATE 'insert into '||p_table_name ||' values(:1) ' using p_value;
end;
```

```
execute add_rows ('emp1',10);
```

maxvlearn

dynamic SQL with single row query

```
--dynamic sql with single row query
create or replace function get_emp2
(p_id number)
return employees%rowtype
is
emp_rec employees%rowtype;
v_query varchar2(1000);
begin
  v_query:='select * from employees where employee_id=:1';
  execute immediate v_query into emp_rec using p_id;
  return emp_rec;
end;
```

```
declare
emp_rec employees%rowtype;
begin
emp_rec:=get_emp2(105);
dbms_output.put_line(emp_rec.employee_id||' '||emp_rec.first_name );
end;
```

dynamic SQL with multi row query

```
create or replace procedure emp_list ( p_dept_id number default null )
is
type  c_emp_dept is ref cursor;
d_cursor c_emp_dept;
v_empno employees.employee_id%type;
v_first_name employees.first_name%type;
v_sql varchar2(1000):='select employee_id, first_name from employees';
begin
   if p_dept_id is null then
   open d_cursor for v_sql;
   else
   v_sql:=v_sql||'  where department_id=:id';
   open d_cursor for v_sql using p_dept_id;
   end if;

     loop
      fetch d_cursor into v_empno, v_first_name;
      exit when d_cursor%notfound;
      dbms_output.put_line(v_empno||' '||v_first_name);
    end loop;
close d_cursor;

end;
```

```
--to get all the employees
execute emp_list;

--to get all the employees in specific dept
execute emp_list (30);
```

dynamic SQL to execute anonymous Block

```
declare
     v_code varchar2(100):=
    'begin
    dbms_output.put_line(''welcome'');
    end;
    ';
begin
execute immediate v_code;
end;
```

Using the DBMS_SQL Package

- The DBMS_SQL package is used to write dynamic SQL in stored procedures and to parse DDL statements.
- You must use the DBMS_SQL package to execute a dynamic SQL statement that has an unknown number of input or output variables, also known as Method 4.
- In most cases, NDS is easier to use and performs better than DBMS_SQL except when dealing with Method 4.
- For example, you must use the DBMS_SQL package in the following situations:
  - You do not know the SELECT list at compile time
  - You do not know how many columns a SELECT statement will return, or what their data types will be

Using the DBMS_SQL Package Subprograms

Examples of the package procedures and functions:

- OPEN_CURSOR Open a new cursor – return cursor ID number
- PARSE check the syntax – associates with the cursor
- BIND_VARIABLE In case of bind variables
- EXECUTE Execute and return number of rows processed
- FETCH_ROWS
- CLOSE_CURSOR

Note: Using the DBMS_SQL package to execute DDL statements can result in a deadlock. For example, the most likely reason is that the package is being used to drop a procedure that you are still using.

Using the DBMS_SQL Package Subprograms (continued)

The PARSE Procedure Parameters

The LANGUAGE_FLAG parameter of the PARSE procedure determines how Oracle handles the SQL statement—that is, using behavior associated with a specific Oracle database version. Using NATIVE (or 1) for this parameter specifies using the normal behavior associated with the database to which the program is connected.

If the LANGUAGE_FLAG parameter is set to V6 (or 0), that specifies version 6 behavior. If the LANGUAGE_FLAG parameter is set to V7 (or 2), that specifies Oracle database version 7 behavior.

```
dbms_sql.parse(v_cur_id,'delete from '||p_table_name ,dbms_sql.native);
```

maxvlearn

2 codes doing the same (Execute immediate VS DBMS_SQL)

1- Using Execute immediate

```
create or replace procedure delete_any_table
( p_table_name varchar2)
is
v_no_rec number;
begin
execute immediate 'delete from '||p_table_name;
commit; --same rules for commit and rollback
v_no_rec:=sql%rowcount;
dbms_output.put_line(v_no_rec ||' record(s) deleted form '||p_table_name );
end;
```

2- Using DBMS_SQL

```
create or replace procedure delete_any_table2
( p_table_name varchar2)
is
v_no_rec number;
v_cur_id number;
begin
v_cur_id:=dbms_sql.open_cursor;
dbms_sql.parse(v_cur_id,'delete from '||p_table_name ,dbms_sql.native);
v_no_rec:=dbms_sql.execute(v_cur_id);
dbms_output.put_line(v_no_rec ||' record(s) deleted form '||p_table_name );
commit;
end;
```

Thank You