
Chapter 24**NATIVE DYNAMIC SQL**

- ❑ What is dynamic SQL?
- ❑ Why do we need dynamic SQL?
- ❑ An Example of Dynamic SQL
- ❑ Execute Immediate Statement
- ❑ Using Placeholders
- ❑ Execute a Query Dynamically
- ❑ Executing multi-row query dynamically
- ❑ Dynamic PL/SQL Blocks

What is Dynamic SQL?

Generally programmer knows the SQL statements that are to be executed at the time of writing program. However, in some cases the programmer may not precisely know the command that is to be executed at the time of writing program. Instead the command may vary from execution to execution and it is to be constructed dynamically (at runtime). An SQL statement that is constructed "on the fly" is dynamic SQL statement.

Until Oracle8 the only way to execute dynamic SQL was by using DBMS_SQL package. Oracle8i introduced native dynamic SQL, where you can directly place dynamic SQL commands into PL/SQL blocks. Native dynamic SQL is faster than using DBMS_SQL package.

Dynamic SQL should be considered only when static SQL is not possible. Dynamic SQL will impact performance.

Why do we need dynamic SQL?

We should use dynamic SQL where static SQL doesn't support the operation that we want to perform.

The following are the situations where you may want to use Dynamic SQL:

- ❑ You want to execute an SQL DDL or DCL command that cannot be executed statically.
That means the command is not allowed in the PL/SQL block. For example, you cannot execute DROP TABLE command from a PL/SQL block.
- ❑ You want to access a table or a view but do not know the name of the object until you run the program.

An Example of Dynamic SQL

Let's look at an example. Assume we have to drop a table. But the name of the table to be dropped is not known. The name of the table is formed using the word SALES and four digits year. For example, if the current year is 2000 then table name would be SALES2000.

That means depending upon the current year the name of the table changes. So the DROP TABLE command should drop the table of the current year and the current year is to be taken from the system date.

The solution to this problem is to construct DROP TABLE command as follows:

```
cmd := 'DROP TABLE sales' || to_char(sysdate,'yyyy');
```

The above statement will put the required command into a char variable and then the command in the char variable is to be executed with EXECUTE IMMEDIATE command.

Execute Immediate Statement

This statement is used to execute a command that is a char variable. In the above example we put the required command in a variable called *cmd*. Now, we can execute the command as follows:

```
EXECUTE IMMEDIATE cmd;
```

Now let us examine the complete syntax of Execute Immediate statement.

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

Dynamic_string	Is the string that contains the command to be executed.
Variable record	Is a variable into which values retrieved by SELECT command are to be copied. If a record is given then complete row is copied into the record. Record must be a variable of user-defined data type or %ROWTYPE% record.
Bind_argument	Is the value that is to be passed to SQL statement that is being executed. This value will replace a placeholder in the command (more on this later).

The following stored procedure takes the name of the table and drops the table.

```
create or replace procedure droptable ( tablename varchar2) is  
  cmd varchar2(100);  
begin  
  cmd := 'drop table ' || tablename;  
  execute immediate cmd;  
end;
```

You can now call the above procedure to drop a table. You have to supply the name of the table to be dropped. Remember that in PL/SQL DDL commands are not allowed in static SQL, so dynamic SQL is required.

```
execute droptable('oldsales');
```

The following is another stored function that takes the year and deletes all rows from the tables whose name is formed as *SALESyear*. Where *year* is the value passed to the procedure.

```
create or replace function deletesalesrows (year number)
return number is
cmd varchar2(100);
begin
    cmd := 'delete from sales' || year;
    execute immediate cmd;
    return sql%rowcount;
end;
```

The function takes year number and deletes all rows from sales table of that year. It also returns the number of rows deleted from the table.

You can invoke the function to delete rows of 2000 sales table and display the number of rows deleted as follows:

```
begin
    dbms_output.put_line( deletesalesrows(2000));
end;
```

Using Placeholders

Placeholders can be used to replace a missing value at runtime. A placeholder is a name preceded by : (colon). The placeholder is to be replaced with a value through USING clause of EXECUTE IMMEDIATE statement.

The following example deletes all rows of the given table where PRODID is equal to the product id passed.

```
create or replace procedure DeleteSalesRows
    (tablename varchar2, prodid number) is
cmd varchar2(100);
begin
    cmd := 'delete rows from ' || tablename || ' where prodid = :prodid';
    execute immediate cmd using prodid;
end;
/
```

At the time of executing a command that has placeholders, we must pass values that replace placeholders through USING clause of EXECUTE IMMEDIATE command. In the above example, the value of variable PRODID will replace the placeholder :PRODID and then the command is executed.

Note: The number of placeholders and the number of values passed by USING clause must be equal.

Placeholders and parameters are matched by the position and not by name. That means if the same placeholder is used for twice or more then for each occurrence the value is to be passed by USING clause as show below.

```
cmd := 'INSERT INTO sometable VALUES (:x, :x, :y, :x)';  
EXECUTE IMMEDIATE sql_stmt USING a, a, b, a;
```

You have to pass four values to the above command although there are only two placeholders – X and Y. This is because of the fact that bind arguments are associated with placeholders by position and not by name. That means the first bind argument is associated with first placeholder and second bind argument with second placeholder and so on.

Note: Placeholders cannot be used for names of schema objects.

The following dynamic SQL statement is INVALID.

```
cmd := 'drop table :table_name';  
execute immediate cmd using tablename;
```

Execute a Query Dynamically

So far whatever we have seen is related to non-query SQL statements. Now, let us see how to execute an SQL query dynamically. The difference between executing a non-query and a query is; a query returns one or more rows. In the beginning we will confine our examples to executing single-row query statements.

When a query in executed dynamically using EXECUTE IMMEDIATE command then we have to supply INTO clause to catch the values retrieved by the query. The following example shows how to use SELECT command with dynamic SQL to retrieve highest and lowest prices at which the given product is sold in the current year.

```
declare  
    cmd varchar2(100);  
    prodid number(5) := 10;  
    lprice number(5);  
    hprice number(5);  
begin  
    cmd := 'select min(price), max(price) from sales';  
    cmd := cmd || to_char(sysdate, 'yyyy') || ' where prodid = :1';  
  
    -- execute the command by sending product id  
  
    execute immediate cmd into lprice, hprice using prodid;  
  
    dbms_output.put_line( 'lowest price is : ' || lprice);
```

```
dbms_output.put_line('highest price is : ' || hprice);  
  
end;
```

While a query is executed using EXECUTE IMMEDIATE command the following rules will apply:

- ❑ The number of columns to be retrieved should be known at the time of writing program.
- ❑ Only one record can be retrieved as only one set of variable can be passed using INTO.

The following function is used to return salary of the employee from the given table.

```
create or replace function GetSalary(tablename varchar2, empno number) return  
number is  
    cmd varchar2(100);  
    v_sal number(5);  
begin  
    cmd := 'select sal from ' || tablename || ' where empno = :empno';  
    execute immediate cmd into v_sal using empno;  
  
    return v_sal;  
  
exception  
    when no_data_found then  
        return null;  
end;
```

The above function returns NULL if the given employee number is not found in the given table.

Executing multi-row query dynamically

Using simple EXECUTE IMMEDIATE statement we can execute a query that retrieves only one row. However, when we need to execute a query that has the potential to retrieve multiple rows, we need to follow a different process.

To deal with multi-row query dynamically, we have to use a Cursor. All the rows retrieved by the query are copied into cursor. And then using FETCH statement each row will be fetched and processed.

The following are the statements used to execute a multi-row query dynamically.

Open-For statement to open the cursor

The first step is to open a cursor with the query statement. This statement associates a cursor variable with the given query and executes the query. This statement has USING option to pass values to placeholders.

The complete syntax of OPEN-FOR statement is as follows:

```
OPEN {cursor_variable}
  FOR dynamic_string
  [USING bind_argument[, bind_argument]...];
```

Cursor_variable Is a weakly typed cursor variable, i.e. a variable that doesn't have any return type.

Dynamic_string The SELECT command to be executed.

Bind_argument Is the value to be passed to placeholders of the command.

The following is an example where all rows of SALES table of the current year are retrieved.

```
DECLARE
  TYPE SalesCursorType IS REF CURSOR; -- define weak REF CURSOR type
  salescursor SalesCursorType; -- declare cursor variable

BEGIN
  OPEN salescursor FOR -- open cursor variable
    'SELECT prodid, qty, price from sales' || to_char(sysdate,'yyyy');
  .
  .
  .

END;
```

Fetching row from cursor

Fetches one row from the cursor and copies the values of the columns into corresponding variables.

```
FETCH {cursor_variable } INTO {define_variable[, define_variable] ...
| record};
```

Cursor_variable is the cursor variable with which the result of the query is associated.

Define_variable is the variable into which the value the corresponding columns should be copied.

Record is the variable of a user-defined record type or %ROWTYPE%.

To fetch rows from SALESCURSOR that we have defined in the previous step, we have to use Fetch statement as follows:

```
Loop
  Fetch salescursor into prodid, qty, price;
  Exit when sales_cursor%notfound; --exit when no more records exist
  -- process the record here
End loop
```

Closing the cursor

The last step in the process is closing the cursor after it is processed.

```
CLOSE {cursor_variable};
```

A sample program

The following program is consolidating all that we have seen regarding how to execute a multi-row query dynamically.

```
declare
  type salescursortype is ref cursor; -- define weak ref cursor type
  salescursor salescursortype; -- declare cursor variable

  prodid number(5);
  qty number(5);
  price number(5);

begin

  open salescursor for -- open cursor variable
    'select proid, qty, price from sales' || to_char(sysdate,'yyyy');

  loop
    fetch salescursor into prodid, qty, price;
    exit when sales_cursor%notfound; --exit when no more records exist
    -- process the record here
  end loop

  close salescursor; -- close the cursor after it is processed.

end; -- end of the block
```

Dynamic PL/SQL Blocks

You can execute an anonymous PL/SQL block dynamically using EXECUTE IMMEDIATE statement. It is useful in cases where the procedure to be invoked is known only at runtime.

The following procedure invokes one of the two procedure based on the name of the company. For this we assume we already have two procedures with names HIKESAL_A and HIKESAL_B where A and B are company names.

```
create or replace procedure callhikesal(company varchar2)is
begin
  execute immediate
    'begin
      hikesal_' || company;
    'end; '
end;
```

The above procedure calls the procedure with the name HIKESAL_A or HIKESAL_B depending on the name of the company – A or B – passed to the procedure.

The same can be done without dynamic SQL but it becomes lengthy and needs to be updated if more there is any change in number of companies.

The following is non-dynamic SQL version of the same procedure.

```
create or replace procedure callhikesal(company varchar2) is
begin
  if company = 'A' then
    hikesal_a;
  else
    hikesal_b;
  end if;
end;
```

But as you can notice, if we add one more company to the list then the procedure is to be modified to accommodate another company.

Summary

Dynamic SQL is the process of constructing SQL commands dynamically and executing them. This was done using DBMS_SQL package prior to Oracle8i but in Oracle8i Native Dynamic SQL was introduced making dynamic SQL less cumbersome and faster.

EXECUTE IMMEDIATE statement is used to execute an SQL command that is constructed at runtime. You can execute DML, single-row SELECT and even multi-row SELECT.

While executing multi-row query, we have to use a cursor to retrieve the data into cursor and then fetch one row at a time from user.

It is also possible to execute a PL/SQL block dynamically.

Exercises

1. _____ option is used to pass bind arguments to placeholders.
2. When there are 2 placeholders used three times in the command then how many bind arguments are to be passed?
3. Create a function that takes table name and a condition and returns the number of rows in the table that satisfy the given condition.