

# BIND VARIABLES

## PERFORMANCE AND SECURITY IN ORACLE

---

### Quick Terms:

**Parsing:** Before Oracle executes a SQL statement, first it analyzes the sql statement and checks whether it is valid or not, and determines how to access tables and join them together. This is called parsing.

**Execution plan :** The optimiser chooses the which table access and join method to use. This produces an execution plan which describes how the sql statement is executed along with its CPU usage, cost and so on.

**Shared pool:** A shared pool is a RAM area within RAM heap that is created at startup time, a component of System Global Area (SGA).

**Hard Parse:** Each time the query is submitted, Oracle first checks in the shared pool to see whether this statement has been submitted before. If it has, the execution plan that this statement previously used is retrieved, and the SQL is executed. If the statement cannot be found in the shared pool, Oracle has to go through the process of parsing the statement, working out the various execution paths and coming up with an optimal access plan before it can be executed. This process is known as a Hard Parse.

Hard Parse is very **CPU intensive**.

**Soft Parse:** Unlike hard parse, Soft Parse doesn't require loading into shared pool because the execution plan is already present in the shared pool from preceding sql statement executions.

The code for this exercise can be found in [github](#) and [gist](#).

---

---

## How does BIND VARIABLE improves the performance ?

Consider an example,

```
SELECT fname, lname, pcode FROM cust WHERE id = 674;  
SELECT fname, lname, pcode FROM cust WHERE id = 234;  
SELECT fname, lname, pcode FROM cust WHERE id = 332;
```

Every time a query is submitted, Oracle first checks in the **shared pool** to see whether this statement has been submitted before. If it has, the execution plan that this statement previously used is retrieved, and the SQL is executed. If the statement cannot be found in the shared pool, Oracle has to go through the process of parsing the statement, working out the various execution paths and coming up with an optimal access plan before it can be executed.

The way to get Oracle to reuse the execution plans for these statements is to use **bind variables**. Bind variables are substitution variables that are used in place of literals (such as 674, 234, 332) and that have the effect of sending exactly the same SQL to Oracle every time the query is executed. For example, in our application, we would just submit.

```
SELECT fname, lname, pcode FROM cust WHERE id = :cust_no;
```

and this time we would be able to reuse the execution plan every time, reducing the latch activity in the SGA, and therefore the total CPU activity, which has the effect of allowing our application to scale up to many users on a large dataset.

Following PL/SQL block proves that Bind variable is a great performance booster:

```
alter system flush shared_pool;
set serveroutput on;

-----No BIND VARIABLE-----

declare
    type rc is ref cursor;
    l_rc rc;
    l_dummy all_objects.object_name%type;
    l_start number default dbms_utility.get_time;
begin
    for i in 1..100
    loop
        open l_rc for
            'select object_name from all_objects where
            object_id= '||i;
        fetch l_rc into l_dummy;
        close l_rc;
        --dbms_output.put_line(l_dummy);
    end loop;

    dbms_output.put_line(round((dbms_utility.get_time - l_start)/100, 2) || ' Seconds');
end;
/
```

Script Output x

Task completed in 76.429 seconds

76.14 Seconds

PL/SQL procedure successfully completed.

Fig : Performance when not using bind variable

```
-----BIND VARIABLE-----
declare
    type rc is ref cursor;
    l_rc rc;
    l_dummy all_objects.object_name%type;
    l_start number default dbms_utility.get_time;
begin
    for i in 1..100
    loop
        open l_rc for
            'select object_name from all_objects where
            object_id=:x' using i;
        fetch l_rc into l_dummy;
        close l_rc;
        --dbms_output.put_line(l_dummy);
    end loop;

    dbms_output.put_line(round((dbms_utility.get_time-l_start)/100, 2) || ' Seconds');
end;
/
```

Script Output x

Task completed in 1.466 seconds

1.34 Seconds

PL/SQL procedure successfully completed.

Fig : Performance while using BIND VARIABLE

### BIND VARIABLE IN SQL\*PLUS:

In SQL\*PLUS we use bind variables as follows:

```
SQL> var myname varchar2(25);
SQL> exec :myname := 'Saman Munikar'

PL/SQL procedure successfully completed.

SQL> print :myname;

MYNAME
-----
Saman Munikar
```

Fig : BIND VARIABLE in SQL\*PLUS

## STEPS:

- Declare bind variable
- Executing bind variable
- Display the value stored in bind variable
  - We could print AFTER the bind variable gets executed just by:
    - SET AUTOPRINT ON;

## BIND VARIABLE IN PL/SQL:

There is a good news in PL/SQL, static SQL in PL/SQL blocks (i.e. not as part of an execute immediate or dbms\_sql statement) is already using bind variables.

For instance,

```

/
-----PL/SQL with builtin bind variable-----
declare
  l_dummy all_objects.object_name%type;
  l_start number default dbms_utility.get_time;
begin
  for i in 1 .. 100
  loop
    begin
      select object_name
      into   l_dummy
      from   all_objects
      where  object_id = i;
    exception
      when no_data_found then null;
    end;
    --dbms_output.put_line(l_dummy);
  end loop;
  dbms_output.put_line(round((dbms_utility.get_time-l_start)/100, 2) || ' Seconds');
end;
/

Script Output x
Task completed in 0.301 seconds

.17 Seconds

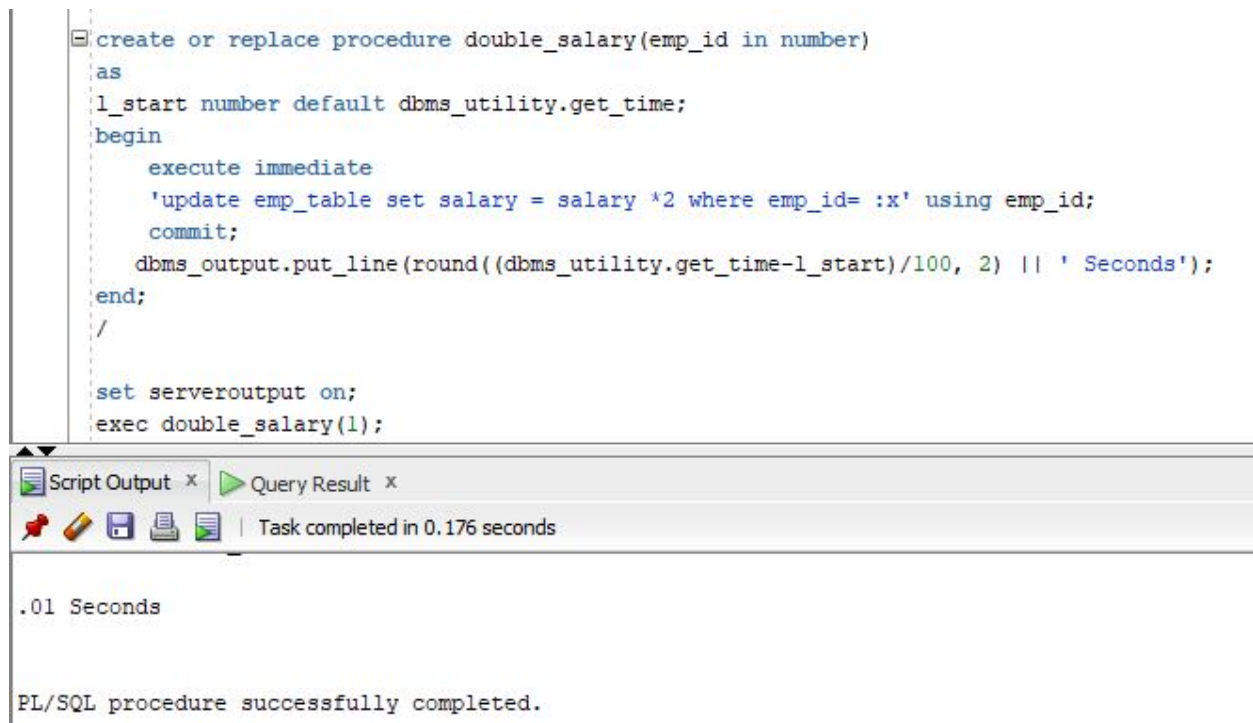
PL/SQL procedure successfully completed.
```

---

Fig : Performance PL/SQL with build-in bind variable

### DYNAMIC SQL:

We just saw that PL/SQL block is very efficient (in static sql); however, we should consider using bind variable when we are using DYNAMIC SQL (i.e. part of an execute immediate or dbms\_sql statement ).



```
create or replace procedure double_salary(emp_id in number)
as
l_start number default dbms_utility.get_time;
begin
    execute immediate
    'update emp_table set salary = salary *2 where emp_id= :x' using emp_id;
    commit;
    dbms_output.put_line(round((dbms_utility.get_time-l_start)/100, 2) || ' Seconds');
end;
/

set serveroutput on;
exec double_salary(1);
```

Script Output x Query Result x

Task completed in 0.176 seconds

.01 Seconds

PL/SQL procedure successfully completed.

Fig: Dynamic SQL with bind variable

### Security:

This is a controversial topic that BIND VARIABLE fully protects our database from SQL Injection (SQLi). It does protect to our tables to some extents.

SQL Injection (SQLi) refers to an injection attack wherein an attacker can execute malicious SQL statements. By saying malicious SQL statement i meant

*SELECT email, password FROM Users\_records WHERE username= samanmunikar or 1=1;*

---

## References:

1. [https://www.akadia.com/services/ora\\_bind\\_variables.html](https://www.akadia.com/services/ora_bind_variables.html)
2. <https://blogs.oracle.com/sql/improve-sql-query-performance-by-using-bind-variables>