CURSOR SHARING

The CURSOR_SHARING parameter can be set so as to instruct Oracle to transparently replace literals with bind variables. The parameter CURSOR_SHARING can take 3 values:

1. **EXACT** This is the default setting. No substitution of bind variables for literals will occur.

   a) if SQL statement uses literals: the optimizer will generate a new execution plan for every combination of literals - optimizer will not replace literals with binds. A new parent/child cursor is generated for every literal combination.

   b) if SQL statement uses bind variables: first time the statement is run, the optimizer will peek at the value of the bind variables and use those specific values to generate an execution plan - all future statements with those bind variables will use that same plan (even if the plan is suboptimal for other values of the bind variable).

2. **SIMILAR** Statements that differ in literals are consider identical *unless* the execution plan is different for the statements.
   Bind variables will be substituted for literal values only if this substitution could not change the execution plan. In some cases, different values of literals could result in different execution plans. If the optimizer determines that this is the case then substitution will not occur.

   a) No histogram: optimizer replaces all literals with binds -> same final effect as with 1(b) and 3(a).

   b) With histogram: optimizer replaces all literals with binds, but peeks at the bind variable EVERY time the statement is run (as opposed to just on the first run through) to see if there is a more optimal execution plan for that specific value of the bind variable (based on histogram statistics). Therefore, a new child cursor is effectively created for every distinct value of the bind variable that the optimizer encounters.

3. **FORCE** Bind variables will be substituted for literals values whenever possible.

   a) Optimizer will replace all literals with binds - and will basically use the same algorithm as scenario 1(b)

Preparing the Environment

```
SQL> create table test (id1 number, id2 number, txt char(1000));
SQL> Insert into test values (1,1, 'one');
SQL> commit;

begin
for i in 1..1000 loop
insert into test values (2,2, 'two');
insert into test values (3,3, 'three');
end loop;
end;
/

SQL> insert into test select * from test where id1=3;
SQL> commit;

SQL> create index test_idx1 on test(id1);
SQL> create index test_idx2 on test(id2);

SQL> select id1,id2, count(*) from test group by id1,id2;
       ID1        ID2   COUNT(*)
---------- ---------- ----------
         1          1          1
         2          2       1000
         3          3       2000
```

CURSOR SHARING = EXACT

In this case when the same statement is issued with different literals, multiple parent cursors will be created.

Parent   Parent   Parent

| | |

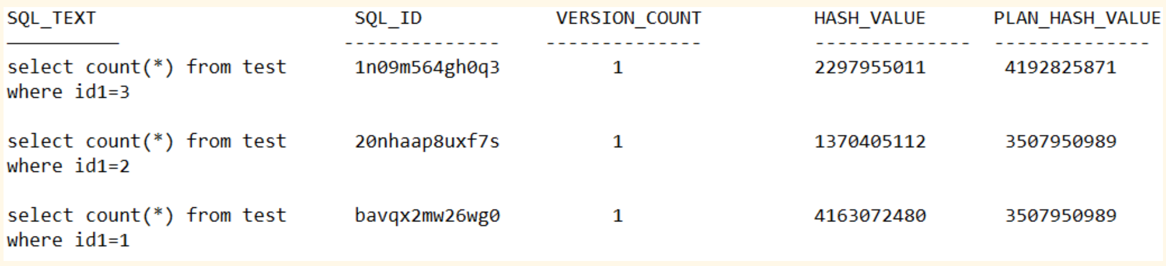
Child   Child   Child

```
SQL> alter system set CURSOR_SHARING='EXACT';
SQL> alter system flush shared_pool;
SQL> show parameter CURSOR_SHARING

NAME                  TYPE      VALUE
--------------        --        --
cursor_sharing        string    EXACT



— Issue identical statements with different values of literals
```

```
SQL> select count(*) from test where id1=1;
SQL> select count(*) from test where id1=2;
SQL> select count(*) from test where id1=3;
```

— Check that the 3 parent cursors have been created
— Note that there is one record for each statement in v$sqlarea as one parent cursor is created for each sql statement since each of these statements differ in their text.

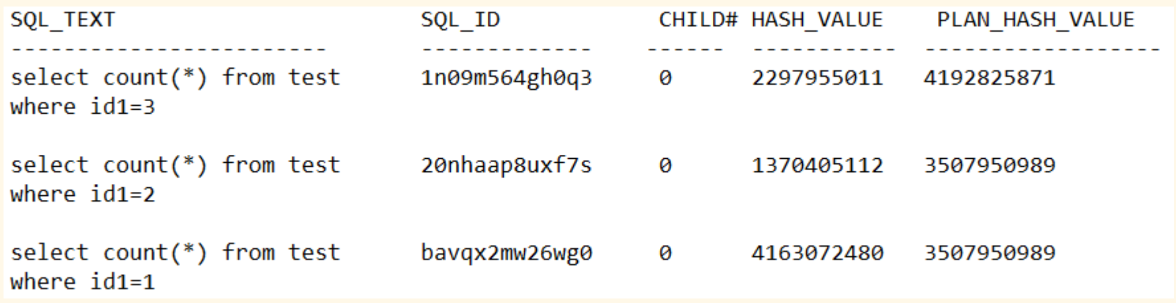
```
a) Each statement has different SQL_ID/HASH_VALUE
b) There is one child per parent cursor (version_count=1)
c) Execution plans for id = 2, 3 is same (full table scan) (same
PLAN_HASH_VALUE)
d) Execution plan for id = 1 is different (indexed access)


SQL> col sql_text for a30 word_wrapped
SQL> SELECT SQL_TEXT, SQL_ID, VERSION_COUNT, HASH_VALUE,
     PLAN_HASH_VALUE FROM v$SQLAREA
     WHERE LOWER(SQL_TEXT) LIKE 'select count(*) from test%'
     AND LOWER (SQL_TEXT) NOT LIKE '%HASH%';
```



```
SQL_TEXT                      SQL_ID          VERSION_COUNT          HASH_VALUE      PLAN_HASH_VALUE
______________               --------------  --------------          --------------  --------------
select count(*) from test     1n09m564gh0q3        1                 2297955011       4192825871
where id1=3

select count(*) from test     20nhaap8uxf7s        1                 1370405112       3507950989
where id1=2

select count(*) from test     bavqx2mw26wg0        1                 4163072480       3507950989
where id1=1
```

— Note that 3 child cursors have been created for the 3 statements

```
SQL> col child_number for 99
SQL> SELECT SQL_TEXT, SQL_ID, CHILD_NUMBER CHILD#, HASH_VALUE,
     PLAN_HASH_VALUE
     FROM v$SQL
     WHERE LOWER (SQL_TEXT) LIKE 'select count (*) from test%'
     AND LOWER (SQL_TEXT) NOT LIKE '%HASH%';
```



```
SQL_TEXT                         SQL_ID          CHILD# HASH_VALUE   PLAN_HASH_VALUE
------------------------         -------------   ------ -----------  ------------------
select count(*) from test        1n09m564gh0q3    0     2297955011   4192825871
where id1=3

select count(*) from test        20nhaap8uxf7s    0     1370405112   3507950989
where id1=2

select count(*) from test        bavqx2mw26wg0    0     4163072480   3507950989
where id1=1
```

1. We can see that in all 6 cursors have been created.
2. 3 parent cursors and 3 child cursors

Each of the cursor occupies memory.

Parent cursors contain sql text whereas child cursor contains execution plan, execution statistics and execution environment.

If we replace literal with a bind variable, all the 3 statements will be identical and hence only parent cursor needs to be created.

Multiple child cursors can be created for different values of the bind variables.
That's what CURSOR_SHARING=SIMILAR does. It replaces literals in the otherwise identical SQL statements with bind variables and only one parent cursor is created.

CURSOR_SHARING = SIMILAR

Thus **CURSOR_SHARING=SIMILAR** reduces the no. parent cursors.

1. If (histogram is not available) only one child cursor will be created.
2. Else (Data is not skewed) only one child cursor will be created.
3. If there is skew in data. Skew is basically the data is not evenly distributed
   If histogram on the column containing skewed data is there multiple child cursors may be created – one for each value of the bind variable

Now, since there is identical skewed data in id1 and id2 , we will create histogram on id1 with one bucket and on id2 with 4 buckets and see the difference.

Ideally we would like one child cursor to be created if execution plan is same for different values of the bind variable.

1. CURSOR_SHARING=SIMILAR WITHOUT HISTOGRAM

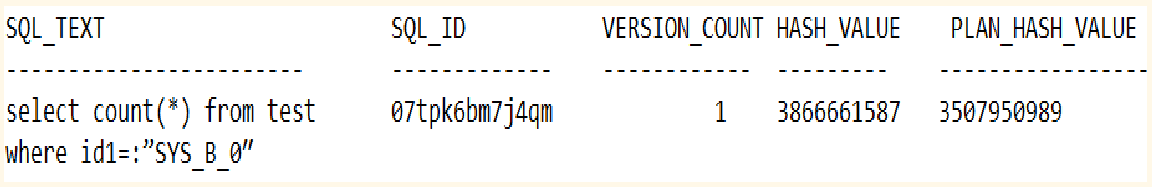
Parent

|

Child

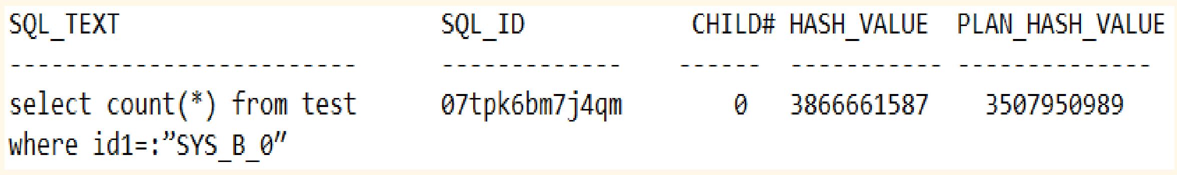
Create histogram only on id1 with one bucket so that optimizer does not know about the skew and hence only one child cursor will be created.

```
SQL> exec dbms_stats.gather_table_stats (OWNNAME => 'HR',-
                                          TABNAME => 'TEST',-
                                          ESTIMATE_PERCENT =>null,-
                           METHOD_OPT => 'FOR COLUMNS SIZE 1 ID1');

SQL> alter system set CURSOR_SHARING='SIMILAR';
SQL> alter system flush shared_pool;
SQL> show parameter CURSOR_SHARING;

— Issue identical statements with different values of literals for
the column on which histogram is not there (id1)

SQL> select count(*) from test where id1=1;
SQL> select count(*) from test where id1=2;
SQL> select count(*) from test where id1=3;

— Check that the only 1 parent cursor has been created and literal
has been replaced by bind variable. (1 record in v$SQLAREA).There
is only one child cursor (version_count=1) since the optimizer
does not know about skew in data

SQL> col sql_text for a30 word_wrapped
SQL> SELECT SQL_TEXT , SQL_ID, VERSION_COUNT,
     HASH_VALUE,PLAN_HASH_VALUE
     FROM V$SQLAREA
     WHERE LOWER(SQL_TEXT) LIKE 'select count(*) from test%'
     AND LOWER(SQL_TEXT) NOT LIKE '%HASH%';
```



```
SQL_TEXT                    SQL_ID        VERSION_COUNT HASH_VALUE   PLAN_HASH_VALUE
-----------------------     ------------  ------------  ---------   ----------------
select count(*) from test   07tpk6bm7j4qm             1  3866661587  3507950989
where id1=:"SYS_B_0"
```

```
— Note there is only one child cursor created i.e. same execution
plan will be used for different values of the bind variable

SQL> col child_number for 99
SQL> SELECT SQL_TEXT, SQL_ID, CHILD_NUMBER CHILD#, HASH_VALUE,
     PLAN_HASH_VALUE
     FROM V$SQL
     WHERE LOWER(SQL_TEXT) LIKE 'select count(*) from test%'
     AND LOWER(SQL_TEXT) NOT LIKE '%HASH%';
```



```
SQL_TEXT                     SQL_ID        CHILD# HASH_VALUE  PLAN_HASH_VALUE
-------------------------    ------------- ------ ----------- --------------
select count(*) from test    07tpk6bm7j4qm      0  3866661587   3507950989
where id1=:"SYS_B_0"
```

2. CURSOR_SHARING=SIMILAR WITH HISTOGRAM

```
      Parent
        +
   +—-- —+———+
   |          |    |
 Child   Child  Child
```

(Each Child has Different Execution Plan)

Create histogram on id2 with 4 buckets so that optimizer knows about the skew in data for each distinct value of the bind variable even if they have the same execution plan

```
SQL> exec dbms_stats.gather_table_stats (OWNNAME => 'HR',-
                                         TABNAME => 'TEST',-
                                         ESTIMATE_PERCENT =>null,-
```

```
                                  CASCADE => TRUE,-
                          METHOD_OPT => 'FOR COLUMNS SIZE 4 ID2');

— Issue identical statements with different values of literals for
the column on which histogram is there (id2)

SQL> alter system flush shared_pool;

SQL> select count (*) from test where id2=1;
SQL> select count (*) from test where id2=2;
SQL> select count (*) from test where id2=3;

— Check that the only 1 parent cursor has been created and literal
has been replaced by bind variable. (1 record in v$SQLAREA).
There are 3 child cursors (version_count=3)

SQL> col sql_text for a30 word_wrapped
SQL> SELECT SQL_TEXT , SQL_ID, VERSION_COUNT,
     HASH_VALUE,PLAN_HASH_VALUE
     FROM V$SQLAREA
     WHERE LOWER(SQL_TEXT) LIKE 'select count(*) from test%'
     AND LOWER(SQL_TEXT) NOT LIKE '%HASH%';
```

```
SQL_TEXT                       SQL_ID        VERSION_COUNT HASH_VALUE PLAN_HASH_VALUE
------------------------------ ------------- ------------- ---------- ---------------
select count(*) from test      3tcujqmqnqs8t 3             3981140249  2432738936
where id2=:"SYS_B_0"
```

```
a) Note that 3 child cursors have been created as optimizer
   realizes that data is skewed and different execution plans
   will  be more efficient for different values of the bind
   variable.

b) 2 children have same execution plan (PLAN_HASH_VALUE)
    (for id=2 and id=3 (Full table scan)


SQL> col child_number for 99
SQL> SELECT SQL_TEXT, SQL_ID, CHILD_NUMBER CHILD#, HASH_VALUE,
     PLAN_HASH_VALUE
     FROM v$SQL
     WHERE LOWER(SQL_TEXT) LIKE 'select count(*) from test%'
     AND LOWER(SQL_TEXT) NOT LIKE '%HASH%';
```

```
SQL_TEXT                       SQL_ID            CHILD#   HASH_VALUE  PLAN_HASH_VALUE
------------------------------ -------------   --------   ----------  ---------------
select count(*) from test      3tcujqmqnqs8t          0   3981140249   2432738936
where id2=:"SYS_B_0"

select count(*) from test      3tcujqmqnqs8t          1   3981140249   2432738936
where id2=:"SYS_B_0"

select count(*) from test      3tcujqmqnqs8t          2   3981140249   1489241381
where id2=:"SYS_B_0"
```

Hence, it can be seen that setting CURSOR_SHARING=SIMILAR replaces literals with bind variables in otherwise identical Sql statements.

1. Only one child cursor is created if optimizer does not know about skew in data. Hence only one execution plan which may not be ideal in some cases.

2. If optimizer is aware of the skew in data, multiple child cursors are created for each distinct value of the bind variable even if they have the same execution plan.

Ideally we would like one child cursor to be created if execution plan is same for different values of the bind variable or else multiple execution plan for each child cursor would be ideal in some cases.

a) Reduces memory usage in library cache as only one parent cursor is created.

b) If data is not skewed or the optimizer is not aware of the skew, optimizer peeks at the value of the bind variable on the first execution of the statement and that plan is used for all the values of the bind variable. Thus only one child cursor is created resulting in minimum memory usage by child cursors. In this case performance will be affected if there is skew in the data.

c) If data is skewed and the optimizer is aware of the skew, multiple child cursor are created – one for each distinct value of the bind variable. In this case performance will be the best as optimizer creates different execution plan for each value of the bind variable. But in this case we will have multiple child cursors created for the same execution plan.

CURSOR_SHARING = FORCE

Ideally we would like one child cursor to be created if execution plan is same for different values of the bind variable.

Setting CURSOR_SHARING=FORCE IN 11G does precisely this but only if the optimizer is aware about the skew in the data

1. CURSOR_SHARING=FORCE IN 11G WITHOUT HISTOGRAM

Parent
|
Child

```
SQL> alter system set CURSOR_SHARING='FORCE';
SQL> alter system flush shared_pool;

-- Issue identical statements with different values of literals
SQL> select count(*) from test where id1=1;
SQL> select count(*) from test where id1=2;
SQL> select count(*) from test where id1=3;
```

```
--Note that only one parent cursor is created
--One child cursor has been created (version_count=1)

SQL> col sql_text for a30 word_wrapped
SQL> SELECT SQL_TEXT , SQL_ID, VERSION_COUNT,
     HASH_VALUE,PLAN_HASH_VALUE
     FROM V$SQLAREA
     WHERE LOWER(SQL_TEXT) LIKE 'select count(*) from test%'
     AND LOWER(SQL_TEXT) NOT LIKE '%HASH%';

SQL_TEXT                        SQL_ID        VERSION_COUNT HASH_VALUE  PLAN_HASH_VALUE
------------------------        ------------- ------------ ----------- ----------------
select count(*) from test       07tpk6bm7j4qm        1      3866661587  3507950989
where id1=:"SYS_B_0"



-- Note that 1 child cursor has been created
SQL> col child_number for 99
SQL> SELECT SQL_TEXT, SQL_ID, CHILD_NUMBER CHILD#, HASH_VALUE,
     PLAN_HASH_VALUE
     FROM V$SQL
     WHERE LOWER(SQL_TEXT) LIKE 'select count(*) from test%'
     AND LOWER(SQL_TEXT) NOT LIKE '%HASH%';

SQL_TEXT                        SQL_ID              CHILD# HASH_VALUE   PLAN_HASH_VALUE
-------------------------       -------------       ------ ------------ ---------------
select count(*) from test       07tpk6bm7j4qm            0   3866661587    3507950989
where id1=:"SYS_B_0"
```

2. CURSOR_SHARING=FORCE IN 11G WITH HISTOGRAM

```
     Parent
        |
   +—-+—--+
   |        |
 Child   Child
```

(Each Child has different Execution Plan)

```
SQL> alter system set CURSOR_SHARING='FORCE';
SQL> alter system flush shared_pool;

-- Issue identical statements with different values of literals
SQL> select count(*) from test where id2=1;
SQL> select count(*) from test where id2=2;
SQL> select count(*) from test where id2=3;


-- Note that only one parent cursor is created
-- Two child cursors have been created (version_count=2)

SQL> col sql_text for a30 word_wrapped
SQL> SELECT SQL_TEXT , SQL_ID, VERSION_COUNT,
     HASH_VALUE,PLAN_HASH_VALUE
     FROM V$SQLAREA
     WHERE LOWER(SQL_TEXT) LIKE 'select count(*) from test%'
     AND LOWER(SQL_TEXT) NOT LIKE '%HASH%';


SQL_TEXT                       SQL_ID        VERSION_COUNT HASH_VALUE  PLAN_HASH_VALUE
------------------------------ ------------- ------------- ----------  ---------------
select count(*) from test      3tcujqmqnqs8t             2  3981140249   2432738936
where id2=:"SYS_B_0"


-- Note that 2 child cursors have been created and each child has
a distinct execution plan (PLAN_HASH_VALUE)

SQL> col child_number for 99
SQL> SELECT SQL_TEXT, SQL_ID, CHILD_NUMBER CHILD#, HASH_VALUE,
     PLAN_HASH_VALUE
     FROM V$SQL
     WHERE LOWER(SQL_TEXT) LIKE 'select count(*) from test%'
     AND LOWER(SQL_TEXT) NOT LIKE '%HASH%';

SQL_TEXT                       SQL_ID          CHILD# HASH_VALUE  PLAN_HASH_VALUE
------------------------------ --------------- ------ ----------- ---------------
select count(*) from test      3tcujqmqnqs8t        0  3981140249   2432738936
where id2=:"SYS_B_0"

select count(*) from test      3tcujqmqnqs8t        1  3981140249   1489241381
where id2=:"SYS_B_0"
```

Without histogram behaviour of cursor_sharing = SIMILAR & FORCE is essentially same. Note that optimizer can generate same execution plan for different values of bind variable. With histogram, cursor_sharing = SIMILAR will cause as many child cursors to be created as are the distinct values of bind variables whereas cursor_sharing = FORCE will cause one child cursor to be created for each distinct execution plan.

Hence, setting CURSOR_SHARING=FORCE in 11g will use the same child cursor if execution plan is same for different values of the bind variables which means saving in memory in the shared pool and saving in the time for scanning the hash chains in the library cache . This new feature of 11g is called ADAPTIVE CURSOR SHARING.

a) Reduces memory usage in library cache as only one parent cursor and only one child cursor are created.

b) If data is not skewed or the optimizer is not aware of the skew, optimizer peeks at the value of the bind variable on the first execution of the statement and that plan is used for all the values of the bind variable. Thus only one child cursor is created resulting in minimum memory usage by child cursors. In this case performance will be affected if there is skew in the data. (same scenario as cursor_sharing=similar).

c) If data is skewed and the optimizer is aware of the skew, multiple child cursor are created for different values of the bind variable – one for each distinct execution plan . In this case performance will be the best as optimizer creates different execution plans for different values of the bind variable. But in this case we will have only child cursor created for the same execution plan thereby resulting in optimum memory usage by child cursors.