2013

CBO – SQL TRANSFORMER

Document describes a few examples of transformations made by CBO.

Environment description

- OS - Oracle Linux Server release 6.3 x64
- Database – Oracle Database 11.2.0.3 EE with sample schemas

Article details

Oracle Cost Based Optimizer consists of three main components: Query Transformer, Esitmator, Plan Generator. In this article I will try to show you some interesting features of the first component – Query Transformer. This component has some very powerful features and most of them remain in shadows, unrevealed while other new features gains all the credit on all conferences and presentations. I think that the understanding of CBO mechanisms is crucial for writing good queries and resolving of many bugs.

Quoting after Oracle Documentation:

> "For some statements, the query transformer determines whether it is advantageous to rewrite the original SQL statement into a semantically equivalent SQL statement with a lower cost. When a viable alternative exists, the database calculates the cost of the alternatives separately and chooses the lowest-cost alternative."

To understand the path of parsing and query transformation, we will use the 10053 database event. For basic usage we will be executing the two important queries:

For finding the default trace file for my dedicated process:

```
select value from v$diag_info where name='Default Trace File';
```

To set the event, for generating the hard-parsing report:

```
alter session set events '10053 trace name context forever, level 1';
```

This document will not explain how to read the whole report – I will focus only on transformation examples.

CASE 1 – Join Factorization

Join factorization is used by Query Transformer in case of UNION or UNION ALL queries – this transformation is useful for statements in which the optimizer can reduce table scans by "pulling" one of the tables to outer query.

Let's assume that we want to find all the employees (first name, last name and department name), which have salary greater than 2000$ or have salary greater than minimal salary assigned to their job identifier. We can achieve this with the following query:

```
select /*TEST1*/ first_name, last_name, department_name
from employees e, departments d
where e.department_id=d.department_id
and   e.salary>=2000
union all
select first_name, last_name, department_name
from employees e, jobs j, departments d
where e.job_id=j.job_id
and  e.department_id=d.department_id
and  e.salary>j.min_salary;
```

There are two common tables for this UNION – EMPLOYEES and DEPARTMENTS. The EMPLOYEES table is used in both queries but with different join predicates. The DEPARTMENTS table however, is used with the same join predicate each time. Let's enable the 10053 event and trace the parsing process to find out what will happen with this SQL statement.

```
SQL> select value from v$diag_info where name='Default Trace File';

VALUE
--------------------------------------------------------------------------------
/u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_3991.trc

SQL> alter session set events '10053 trace name context forever, level 1';

SQL> ;
  1  select /*TEST1*/ first_name, last_name, department_name
  2  from employees e, departments d
  3  where e.department_id=d.department_id
  4  and   e.salary>=2000
  5  union all
  6  select first_name, last_name, department_name
  7  from employees e, jobs j, departments d
  8  where e.job_id=j.job_id
  9  and  e.department_id=d.department_id
 10* and  e.salary>j.min_salary;
/
```

At the head part of the trace file we can see a legend, which should ease us the interpreting. When I've seen it for the first time in 11g database I was surprised by the number of transformation, which can be made by CBO. In fact most of those names meant nothing to me ☺

Here is the output of the first part of the legend:

```
Legend
The following abbreviations are used by optimizer trace.
CBQT - cost-based query transformation
JPPD - join predicate push-down
OJPPD - old-style (non-cost-based) JPPD
FPD - filter push-down
PM - predicate move-around
CVM - complex view merging
SPJ - select-project-join
SJC - set join conversion
SU - subquery unnesting
OBYE - order by elimination
OST - old style star transformation
ST - new (cbqt) star transformation
CNT - count(col) to count(*) transformation
JE - Join Elimination
JF - join factorization ← This is what we are looking for ☺
SLP - select list pruning
DP - distinct placement
```

OK, after miles of text on which consists the list of parameters used by CBO (ca. 319) and the information about bug fix control environment (ca. 652), we can see some actual action.

Our queries where marked as follows (qb stands for “query block”):

- qb_name=SEL$1 – for the first query in the UNION
- qb_name=SEL$2 – for the second query in the UNION
- qb_name=SET$1 – for the whole UNION

A few screens of text later we can see the following info:

```
JF: Checking validity of join factorization for query block SEL$1 (#3)
JF: Bypassed: not a UNION or UNION-ALL query block.
JF: Checking validity of join factorization for query block SEL$2 (#2)
JF: Bypassed: not a UNION or UNION-ALL query block.
JF: Checking validity of join factorization for query block SET$1 (#1)
```

Here we can see, that Join Factorization was rejected for the first two query blocks but SET$1 will be accepted and will proceed to transformation mechanizm.

```
Registered qb: SET$1 0xb6a62edc (COPY SET$1)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature(): NULL
Registered qb: SEL$2 0xb6a631f8 (COPY SEL$2)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature(): NULL
Registered qb: SEL$1 0xb68fd1ec (COPY SEL$1)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature(): NULL
*********************************
Cost-Based Join Factorization
*********************************
```

```
Join-Factorization on query block SET$1 (#1)
JF: Using search type: exhaustive
JF: Checking validity of join factorization for query block SET$1 (#1)

JF: Generate basic transformation units
Validating JF unit: (branch: {2, 3} table: {E, E})
  rejected: table filter predicates do not match
```

The table “EMPLOYEES” was rejected, because (as I explained earlier) the predicates are not the same for each statement.

```
Validating JF unit: (branch: {2, 3} table: {D, D})
  passed JF validation
```

But here we can see that table “DEPARTMENTS” was accepted by the parser.

```
JF: Generate transformation units from basic units
```

OK. Let’s try to find the final query after all transformations:

```
Final query after transformations:******* UNPARSED QUERY IS *******
 SELECT   "VW_JF_SET$BB08C903"."ITEM_2"   "FIRST_NAME","VW_JF_SET$BB08C903"."ITEM_3"
"LAST_NAME","D"."DEPARTMENT_NAME"    "DEPARTMENT_NAME"    FROM        (    (SELECT
"E"."DEPARTMENT_ID"   "ITEM_1","E"."FIRST_NAME"   "ITEM_2","E"."LAST_NAME"   "ITEM_3"
FROM   "HR"."EMPLOYEES"   "E"   WHERE   "E"."SALARY">=2000)   UNION   ALL     (SELECT
"E"."DEPARTMENT_ID"   "ITEM_1","E"."FIRST_NAME"   "ITEM_2","E"."LAST_NAME"   "ITEM_3"
FROM  "HR"."EMPLOYEES"  "E","HR"."JOBS"  "J"  WHERE  "E"."SALARY">"J"."MIN_SALARY"  AND
"E"."JOB_ID"="J"."JOB_ID"))   "VW_JF_SET$BB08C903","HR"."DEPARTMENTS"   "D"   WHERE
"VW_JF_SET$BB08C903"."ITEM_1"="D"."DEPARTMENT_ID"
```

After formatting the query, we will see beauty of this transformation ☺

```
SELECT "VW_JF_SET$BB08C903"."ITEM_2" "FIRST_NAME",
  "VW_JF_SET$BB08C903"."ITEM_3" "LAST_NAME",
  "D"."DEPARTMENT_NAME" "DEPARTMENT_NAME"
FROM (
  (SELECT "E"."DEPARTMENT_ID" "ITEM_1",
    "E"."FIRST_NAME" "ITEM_2",
    "E"."LAST_NAME" "ITEM_3"
  FROM "HR"."EMPLOYEES" "E"
  WHERE "E"."SALARY">=2000
  )
UNION ALL
  (SELECT "E"."DEPARTMENT_ID" "ITEM_1",
    "E"."FIRST_NAME" "ITEM_2",
    "E"."LAST_NAME" "ITEM_3"
  FROM "HR"."EMPLOYEES" "E",
    "HR"."JOBS" "J"
  WHERE "E"."SALARY">"J"."MIN_SALARY"
  AND "E"."JOB_ID"  ="J"."JOB_ID"
  )) "VW_JF_SET$BB08C903",
  "HR"."DEPARTMENTS" "D"
WHERE "VW_JF_SET$BB08C903"."ITEM_1"="D"."DEPARTMENT_ID";
```

As you can see, the table “DEPARTMENTS” was “pulled out” of the statement and used in a join – thanks to this, the CBO reduced segment scans on that particular table.

CASE 2 – Set To Join Conversion

While reading the output from the previous report, you could see the following sentence:

```
Set-Join Conversion (SJC)
************************
SJC:   Checking validity of SJC on query block SET$1 (#1)
SJC:     SJC bypassed: Not enabled by hint/parameter.
```

Really interesting ☺ So that means, that if I change some parameter I will see some additional transformation… Well, there is one parameter that is promising: _convert_set_to_join with default value set to FALSE. Let's see what will happen when I'll change it and run the previous query once again.

```
SQL> select value from v$diag_info where name='Default Trace File';

VALUE
--------------------------------------------------------------------------------
/u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_6198.trc

SQL> alter session set "_convert_set_to_join"=true;

SQL> select /*TEST2*/ first_name, last_name, department_name
from employees e, departments d
where e.department_id=d.department_id
and   e.salary>=2000
union all
select first_name, last_name, department_name
from employees e, jobs j, departments d
where e.job_id=j.job_id
and  e.department_id=d.department_id
and  e.salary>j.min_salary  2    3    4    5    6    7    8    9   10
 11  /
```

Great! Let's see at the trace:

```
SJC: Considering set-join conversion in query block SET$1 (#1)
************************
Set-Join Conversion (SJC)
************************
SJC:   Checking validity of SJC on query block SET$1 (#1)
SJC:   Passed validity checks.
SJC: SJC: Applying SJC on query block SET$1 (#1)
SJC: Considering set-join conversion in query block SEL$2 (#2)
************************
Set-Join Conversion (SJC)
************************
SJC: not performed
SJC: Considering set-join conversion in query block SEL$1 (#3)
************************
Set-Join Conversion (SJC)
************************
SJC: not performed
SJC: not performed
```

Well, something has happened but, unfortunately, this query is not suitable for this type of transformation. But UNION is not the only type of SET statements – let's try INTERSECT.

```
select /*TEST3*/ first_name, last_name, department_name
from employees e, departments d
where e.department_id=d.department_id
and   e.salary>=2000
intersect
select first_name, last_name, department_name
from employees e1, jobs j, departments d1
where e1.job_id=j.job_id
and  e1.department_id=d1.department_id
and  e1.salary>j.min_salary
```

The trace file looks like this:

```
SJC: Considering set-join conversion in query block SET$1 (#0)
************************
Set-Join Conversion (SJC)
************************
SJC:   Checking validity of SJC on query block SET$1 (#0)
SJC:   Passed validity checks.
SJC: SJC: Applying SJC on query block SET$1 (#0)
Registered qb: SET$09AAA538 0xb6b5139c (SET QUERY BLOCK SET$1; SET$1)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SET$09AAA538 nbfros=2 flg=0
    fro(0): flg=1 objn=0 hint_alias="NULL_HALIAS"@"SET$09AAA538"
    fro(1): flg=1 objn=0 hint_alias="NULL_HALIAS"@"SET$09AAA538"
```

Great! It looks like, the transformation finally has happened! Now I will look for the final query after transformation and format the output for clarity:

```
SELECT DISTINCT "E"."FIRST_NAME" "FIRST_NAME",
  "E"."LAST_NAME" "LAST_NAME",
  "D"."DEPARTMENT_NAME" "DEPARTMENT_NAME"
FROM "HR"."EMPLOYEES" "E1",
  "HR"."JOBS" "J",
  "HR"."DEPARTMENTS" "D1",
  "HR"."EMPLOYEES" "E",
  "HR"."DEPARTMENTS" "D"
WHERE SYS_OP_MAP_NONNULL("E"."FIRST_NAME")=SYS_OP_MAP_NONNULL("E1"."FIRST_NAME")
AND "E"."LAST_NAME"                        ="E1"."LAST_NAME"
AND "D"."DEPARTMENT_NAME"                  ="D1"."DEPARTMENT_NAME"
AND "E"."DEPARTMENT_ID"                    ="D"."DEPARTMENT_ID"
AND "E"."SALARY"                          >=2000
AND "E1"."JOB_ID"                          ="J"."JOB_ID"
AND "E1"."DEPARTMENT_ID"                   ="D1"."DEPARTMENT_ID"
AND "E1"."SALARY"                          >"J"."MIN_SALARY";
```

The output is a little bit strange. Please notice the usage of undocumented function SYS_OP_MAP_NONNULL. You can read about it here:

http://www.oracle-base.com/articles/misc/null-related-functions.php#sys_op_map_nonnull

There is a funny bug associated with this function in 11.2.0.2 database – please see Metalink DocID 12346165.8

To decide which method is better in our current environment, we can compare the execution plans for query with and without this SJC transformation.

Transformed query:

```
SQL> set pagesize 100
SQL> set linesize 250
SQL> alter session set statistics_level=all;

Sesja zostala zmieniona.

SQL> get intersect
  1  select /*TEST4*/ first_name, last_name, department_name
  2  from employees e, departments d
  3  where e.department_id=d.department_id
  4  and   e.salary>=2000
  5  intersect
  6  select first_name, last_name, department_name
  7  from employees e1, jobs j, departments d1
  8  where e1.job_id=j.job_id
  9  and  e1.department_id=d1.department_id
 10* and  e1.salary>j.min_salary
/
(….output removed for clarity…)

SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------
SQL_ID  49q50cufc0mu3, child number 0
-------------------------------------
select /*TEST4*/ first_name, last_name, department_name from employees
e, departments d where e.department_id=d.department_id and
e.salary>=2000 intersect select first_name, last_name, department_name
from employees e1, jobs j, departments d1 where e1.job_id=j.job_id and
e1.department_id=d1.department_id and  e1.salary>j.min_salary

Plan hash value: 388878752

-------------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                          | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |             |      1 |        |    106 |00:00:00.01 |     343 |     17 |       |       |          |
|   1 |  HASH UNIQUE                       |             |      1 |      1 |    106 |00:00:00.01 |     343 |     17 |   786K|   786K| 1277K (0)|
|   2 |   NESTED LOOPS                     |             |      1 |        |    106 |00:00:00.01 |     343 |     17 |       |       |          |
|   3 |    NESTED LOOPS                    |             |      1 |      1 |    106 |00:00:00.01 |     237 |     17 |       |       |          |
|   4 |     NESTED LOOPS                   |             |      1 |      1 |    106 |00:00:00.01 |     233 |     17 |       |       |          |
|   5 |      NESTED LOOPS                  |             |      1 |      1 |    107 |00:00:00.01 |     123 |      8 |       |       |          |
|*  6 |       HASH JOIN                    |             |      1 |      1 |    107 |00:00:00.01 |      12 |      6 |   768K|   768K| 1242K (0)|
|   7 |        TABLE ACCESS FULL           | EMPLOYEES   |      1 |    107 |    107 |00:00:00.01 |       6 |      6 |       |       |          |
|*  8 |        TABLE ACCESS FULL           | EMPLOYEES   |      1 |    107 |    107 |00:00:00.01 |       6 |      0 |       |       |          |
|*  9 |       TABLE ACCESS BY INDEX ROWID  | JOBS        |    107 |      1 |    107 |00:00:00.01 |     111 |      2 |       |       |          |
|* 10 |        INDEX UNIQUE SCAN           | JOB_ID_PK   |    107 |      1 |    107 |00:00:00.01 |       4 |      1 |       |       |          |
|  11 |      TABLE ACCESS BY INDEX ROWID   | DEPARTMENTS |    107 |      1 |    106 |00:00:00.01 |     110 |      9 |       |       |          |
|* 12 |       INDEX UNIQUE SCAN            | DEPT_ID_PK  |    107 |      1 |    106 |00:00:00.01 |       4 |      1 |       |       |          |
|* 13 |     INDEX UNIQUE SCAN              | DEPT_ID_PK  |    106 |      1 |    106 |00:00:00.01 |       4 |      0 |       |       |          |
|* 14 |    TABLE ACCESS BY INDEX ROWID     | DEPARTMENTS |    106 |      1 |    106 |00:00:00.01 |     106 |      0 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   6 - access(SYS_OP_MAP_NONNULL("FIRST_NAME")=SYS_OP_MAP_NONNULL("FIRST_NAME") AND "LAST_NAME"="LAST_NAME")
   8 - filter("E"."SALARY">=2000)
   9 - filter("E1"."SALARY">"J"."MIN_SALARY")
  10 - access("E1"."JOB_ID"="J"."JOB_ID")
  12 - access("E1"."DEPARTMENT_ID"="D1"."DEPARTMENT_ID")
  13 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
  14 - filter("DEPARTMENT_NAME"="DEPARTMENT_NAME")
```

Query without transformation:

```
SQL> set pagesize 100
SQL> set linesize 250
SQL> alter session set statistics_level=all;

Sesja zostala zmieniona.

SQL> alter session set "_convert_set_to_join"=false;

Sesja zostala zmieniona.

SQL> get intersect
  1  select /*TEST5*/ first_name, last_name, department_name
  2  from employees e, departments d
  3  where e.department_id=d.department_id
  4  and   e.salary>=2000
  5  intersect
  6  select first_name, last_name, department_name
  7  from employees e1, jobs j, departments d1
  8  where e1.job_id=j.job_id
  9  and  e1.department_id=d1.department_id
 10* and  e1.salary>j.min_salary
/
(….output removed for clarity…)

SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));
```

```
SQL> select * from table(dbms_xplan.display_cursor(null,null,'ALLSTATS LAST'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------
SQL_ID  3xyk82mbny8u6, child number 0
-------------------------------------
select /*TEST5*/ first_name, last_name, department_name from employees
e, departments d where e.department_id=d.department_id and
e.salary>=2000 intersect select first_name, last_name, department_name
from employees e1, jobs j, departments d1 where e1.job_id=j.job_id and
e1.department_id=d1.department_id and  e1.salary>j.min_salary

Plan hash value: 1392547086

------------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |  OMem |  1Mem | Used-Mem |
------------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |      1 |        |    106 |00:00:00.01 |      22 |     18 |       |       |          |
|   1 |  INTERSECTION                 |             |      1 |        |    106 |00:00:00.01 |      22 |     18 |       |       |          |
|   2 |   SORT UNIQUE                 |             |      1 |    106 |    106 |00:00:00.01 |       8 |     15 |  6144 |  6144 | 6144  (0)|
|   3 |    MERGE JOIN                 |             |      1 |    106 |    106 |00:00:00.01 |       8 |     15 |       |       |          |
|   4 |     TABLE ACCESS BY INDEX ROWID | DEPARTMENTS |      1 |     27 |     27 |00:00:00.01 |       2 |      9 |       |       |          |
|   5 |      INDEX FULL SCAN          | DEPT_ID_PK  |      1 |     27 |     27 |00:00:00.01 |       1 |      1 |       |       |          |
|*  6 |     SORT JOIN                 |             |     27 |    107 |    106 |00:00:00.01 |       6 |      6 |  9216 |  9216 | 8192  (0)|
|*  7 |      TABLE ACCESS FULL        | EMPLOYEES   |      1 |    107 |    107 |00:00:00.01 |       6 |      6 |       |       |          |
|   8 |   SORT UNIQUE                 |             |      1 |     63 |    106 |00:00:00.01 |      14 |      3 |  6144 |  6144 | 6144  (0)|
|*  9 |    HASH JOIN                  |             |      1 |     63 |    106 |00:00:00.01 |      14 |      3 |   848K|   848K|  829K (0)|
|  10 |     MERGE JOIN                |             |      1 |     64 |    107 |00:00:00.01 |       8 |      2 |       |       |          |
|  11 |      TABLE ACCESS BY INDEX ROWID| JOBS        |      1 |     19 |     19 |00:00:00.01 |       2 |      2 |       |       |          |
|  12 |       INDEX FULL SCAN         | JOB_ID_PK   |      1 |     19 |     19 |00:00:00.01 |       1 |      1 |       |       |          |
|* 13 |      FILTER                   |             |     19 |        |    107 |00:00:00.01 |       6 |      0 |       |       |          |
|* 14 |       SORT JOIN               |             |     19 |    107 |    107 |00:00:00.01 |       6 |      0 |  9216 |  9216 | 8192  (0)|
|  15 |        TABLE ACCESS FULL      | EMPLOYEES   |      1 |    107 |    107 |00:00:00.01 |       6 |      0 |       |       |          |
|  16 |     TABLE ACCESS FULL         | DEPARTMENTS |      1 |     27 |     27 |00:00:00.01 |       6 |      1 |       |       |          |
------------------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   6 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
       filter("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   7 - filter("E"."SALARY">=2000)
   9 - access("E1"."DEPARTMENT_ID"="D1"."DEPARTMENT_ID")
  13 - filter("E1"."SALARY">"J"."MIN_SALARY")
  14 - access("E1"."JOB_ID"="J"."JOB_ID")
       filter("E1"."JOB_ID"="J"."JOB_ID")
```

Well, the amount of the data in this example is to small to decide which plan is better – the one with no SJC transformation uses less buffer gets, but a lot of additional sorting can cause suboptimal workarea executions, leading to excessive TEMP usage.

OK, but what if my production environment would require transforming this INTERSECT query into a join? How to do it if I can't change the query in my application? Clearly setting the "_convert_set_to_join" parameter for the whole database is not the solution. Well, we could set this parameter in our session, run the query and after that – save SQL plan in the SQL baseline.

```
SQL> alter session set "_convert_set_to_join"=true;
SQL> ed
Wrote file afiedt.buf

  1  select first_name, last_name, department_name
  2  from employees e, departments d
  3  where e.department_id=d.department_id
  4  and   e.salary>=2000
  5  intersect
  6  select first_name, last_name, department_name
  7  from employees e1, jobs j, departments d1
  8  where e1.job_id=j.job_id
  9  and  e1.department_id=d1.department_id
 10* and  e1.salary>j.min_salary;

(output removed for clarity)

SQL> declare
  2      x number;
  3  begin
  4     x:=dbms_spm.load_plans_from_cursor_cache(sql_id=>'1dvcfdaxj6292',
  5                                              plan_hash_value=>388878752,
  6                                              enabled=>'YES',fixed=>'YES');
  7  end;
  8  /

PL/SQL procedure successfully completed.
```

Let's see what will happen in the new session, if I will run this query once again?

```
SQL> get intersect
  1  select first_name, last_name, department_name
  2  from employees e, departments d
  3  where e.department_id=d.department_id
  4  and   e.salary>=2000
  5  intersect
  6  select first_name, last_name, department_name
  7  from employees e1, jobs j, departments d1
  8  where e1.job_id=j.job_id
  9  and  e1.department_id=d1.department_id
 10* and  e1.salary>j.min_salary
```

(output removed for clarity)

```
SQL> set pagesize 100
SQL> set linesize 250
SQL> select * from table(dbms_xplan.display_cursor(null,null));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
------------------------------------------
SQL_ID  1dvcfdaxj6292, child number 1
-------------------------------------
select first_name, last_name, department_name from employees e,
departments d where e.department_id=d.department_id and
e.salary>=:"SYS_B_0" intersect select first_name, last_name,
department_name from employees e1, jobs j, departments d1 where
e1.job_id=j.job_id and  e1.department_id=d1.department_id and
e1.salary>j.min_salary

Plan hash value: 388878752

-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |             |       |       |    11 (100)|          |
|   1 |  HASH UNIQUE                  |             |     1 |    97 |    11  (19)| 00:00:01 |
|   2 |   NESTED LOOPS                |             |       |       |            |          |
|   3 |    NESTED LOOPS               |             |     1 |    97 |    10  (10)| 00:00:01 |
|   4 |     NESTED LOOPS              |             |     1 |    81 |     9  (12)| 00:00:01 |
|   5 |      NESTED LOOPS             |             |     1 |    65 |     8  (13)| 00:00:01 |
|*  6 |       HASH JOIN               |             |     1 |    53 |     7  (15)| 00:00:01 |
|   7 |        TABLE ACCESS FULL      | EMPLOYEES   |   107 |  3317 |     3   (0)| 00:00:01 |
|*  8 |        TABLE ACCESS FULL      | EMPLOYEES   |   107 |  2354 |     3   (0)| 00:00:01 |
|*  9 |       TABLE ACCESS BY INDEX ROWID| JOBS        |     1 |    12 |     1   (0)| 00:00:01 |
|* 10 |        INDEX UNIQUE SCAN      | JOB_ID_PK   |     1 |       |     0   (0)|          |
|  11 |      TABLE ACCESS BY INDEX ROWID | DEPARTMENTS |     1 |    16 |     1   (0)| 00:00:01 |
|* 12 |       INDEX UNIQUE SCAN       | DEPT_ID_PK  |     1 |       |     0   (0)|          |
|* 13 |     INDEX UNIQUE SCAN         | DEPT_ID_PK  |     1 |       |     0   (0)|          |
|* 14 |    TABLE ACCESS BY INDEX ROWID| DEPARTMENTS |     1 |    16 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   6 - access(SYS_OP_MAP_NONNULL("FIRST_NAME")=SYS_OP_MAP_NONNULL("FIRST_NAME") AND
              "LAST_NAME"="LAST_NAME")
   8 - filter("E"."SALARY">=:SYS_B_0)
   9 - filter("E1"."SALARY">"J"."MIN_SALARY")
  10 - access("E1"."JOB_ID"="J"."JOB_ID")
  12 - access("E1"."DEPARTMENT_ID"="D1"."DEPARTMENT_ID")
  13 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
  14 - filter("DEPARTMENT_NAME"="DEPARTMENT_NAME")

Note
-----
   - SQL plan baseline SQL_PLAN_506855699q6g266b56ef0 used for this statement


47 rows selected.
```

As you can see – because of baseline usage – the query is using the execution plan with joins. But now there is no need of setting the hidden parameter ☺

CASE 3 – Subquery unnesting

This case is quite simple – subquery is transformed into a join. At the beginning let's try to compare two queries – with IN and EXISTS syntax.

Query with "IN":

```
SQL> select /*TEST6*/ department_name
  2  from departments d
  3  where department_id in (select department_id
  4                          from employees);
```

And after transformation:

```
SELECT "D"."DEPARTMENT_NAME" "DEPARTMENT_NAME"
FROM "HR"."EMPLOYEES" "EMPLOYEES",
```

```
  "HR"."DEPARTMENTS" "D"
WHERE "D"."DEPARTMENT_ID"="EMPLOYEES"."DEPARTMENT_ID";
```

Query with „EXISTS”:

```
SQL> select /*TEST7*/ department_name
  2  from departments d
  3  where exists (select 1
  4            from employees e
  5            where e.department_id=d.department_id);
```

And after transformation:

```
SELECT "D"."DEPARTMENT_NAME" "DEPARTMENT_NAME"
FROM "HR"."EMPLOYEES" "E",
  "HR"."DEPARTMENTS" "D"
WHERE "E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID";
```

It looks quite similar, isn’t it? ☺

Regardless of using normal subquery or Common Table Expression, this transformation can lead to remove the inner query. Nevertheless you should remember that using, for example, rownum or window functions inside subquery, prevents CBO from unnesting.

Check out this CTE query:

```
with v_emps as
(
 select first_name, last_name, department_name, job_id, salary
 from employees e, departments d
 where e.department_id=d.department_id
)
select e1.*, j.job_title, j.max_salary
from v_emps e1, jobs j
where e1.job_id=j.job_id
and   e1.salary>=2000;
```

After the transformation, the final query looks like this:

```
SELECT "E"."FIRST_NAME" "FIRST_NAME",
  "E"."LAST_NAME" "LAST_NAME",
  "D"."DEPARTMENT_NAME" "DEPARTMENT_NAME",
  "E"."JOB_ID" "JOB_ID",
  "E"."SALARY" "SALARY",
  "J"."JOB_TITLE" "JOB_TITLE",
  "J"."MAX_SALARY" "MAX_SALARY"
FROM "HR"."EMPLOYEES" "E",
  "HR"."DEPARTMENTS" "D",
  "HR"."JOBS" "J"
WHERE "E"."JOB_ID"     ="J"."JOB_ID"
AND "E"."SALARY"      >=2000
AND "E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID";
```

But when we'll make a little change to the query:

```
with v_emps as
(
 select first_name, last_name, department_name, job_id, salary, rownum as x
 from employees e, departments d
 where e.department_id=d.department_id
)
select /*TEST9*/ e1.*, j.job_title, j.max_salary
from v_emps e1, jobs j
where e1.job_id=j.job_id
and   e1.salary>=2000;
```

The final query will be executed with subquery – the unnesting can't be done because of ROWNUM usage. Unnesting would have changed the meaning of the query.

```
SELECT "E1"."FIRST_NAME" "FIRST_NAME",
  "E1"."LAST_NAME" "LAST_NAME",
  "E1"."DEPARTMENT_NAME" "DEPARTMENT_NAME",
  "E1"."JOB_ID" "JOB_ID",
  "E1"."SALARY" "SALARY",
  "E1"."X" "X",
  "J"."JOB_TITLE" "JOB_TITLE",
  "J"."MAX_SALARY" "MAX_SALARY"
FROM
  (SELECT "E"."FIRST_NAME" "FIRST_NAME",
    "E"."LAST_NAME" "LAST_NAME",
    "D"."DEPARTMENT_NAME" "DEPARTMENT_NAME",
    "E"."JOB_ID" "JOB_ID",
    "E"."SALARY" "SALARY",
    ROWNUM "X"
  FROM "HR"."EMPLOYEES" "E",
    "HR"."DEPARTMENTS" "D"
  WHERE "E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID"
  ) "E1",
  "HR"."JOBS" "J"
WHERE "E1"."JOB_ID"="J"."JOB_ID"
AND "E1"."SALARY" >=2000;
```

One of the best features of the CTE is possibility of transformation. When the CTE is used more than once in the statement, the CBO can make decision to create a temporary table. Let's look at the following SQL:

```
with v_emps_tree as
(
  select first_name, last_name, department_name, salary,
         level as lv, employee_id
  from employees e, departments d
  where e.department_id=d.department_id
  start with e.manager_id is null
  connect by e.manager_id=prior e.employee_id
)
select /*TEST*/ *
from v_emps_tree e
where lv=(select lv
          from v_emps_tree
          where employee_id=(select manager_id
                             from departments
                             where department_name='Shipping'));
```

Explain plan will show us, that the TEMP TABLE TRANSFORMATION was used.

```
SQL> select * from table(dbms_xplan.display_cursor(null,null));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------
------------------------------------------
SQL_ID  8qkhzn84xwnnw, child number 0
-------------------------------------
with v_emps_tree as (   select first_name, last_name, department_name,
salary,         level as lv, employee_id   from employees e,
departments d   where e.department_id=d.department_id   start with
e.manager_id is null   connect by e.manager_id=prior e.employee_id )
select /*TEST*/ * from v_emps_tree e where lv=(select lv           from
v_emps_tree          where employee_id=(select manager_id
             from departments                          where
department_name=:"SYS_B_0"))

Plan hash value: 769911149

------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                       | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                            |       |       |    14 (100)|          |
|   1 |  TEMP TABLE TRANSFORMATION               |                            |       |       |            |          |
|   2 |   LOAD AS SELECT                         |                            |       |       |            |          |
|*  3 |    CONNECT BY NO FILTERING WITH START-WITH|                            |       |       |            |          |
|   4 |     MERGE JOIN                           |                            |   106 |  4876 |     6  (17)| 00:00:01 |
|   5 |      TABLE ACCESS BY INDEX ROWID         | DEPARTMENTS                |    27 |   432 |     2   (0)| 00:00:01 |
|   6 |       INDEX FULL SCAN                    | DEPT_ID_PK                 |    27 |       |     1   (0)| 00:00:01 |
|*  7 |      SORT JOIN                           |                            |   107 |  3210 |     4  (25)| 00:00:01 |
|   8 |       TABLE ACCESS FULL                  | EMPLOYEES                  |   107 |  3210 |     3   (0)| 00:00:01 |
|*  9 |   VIEW                                   |                            |     7 |   574 |     2   (0)| 00:00:01 |
|  10 |    TABLE ACCESS FULL                     | SYS_TEMP_0FD9D6617_10B49D  |     7 |   336 |     2   (0)| 00:00:01 |
|* 11 |    VIEW                                  |                            |     7 |   182 |     2   (0)| 00:00:01 |
|  12 |     TABLE ACCESS FULL                    | SYS_TEMP_0FD9D6617_10B49D  |     7 |   336 |     2   (0)| 00:00:01 |
|* 13 |     TABLE ACCESS FULL                    | DEPARTMENTS                |     1 |    15 |     3   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("E"."MANAGER_ID"=PRIOR NULL)
       filter("E"."MANAGER_ID" IS NULL)
   7 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
       filter("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   9 - filter("LV"=)
  11 - filter("EMPLOYEE_ID"=)
  13 - filter("DEPARTMENT_NAME"=:SYS_B_0)
```

This transformation was made, because the “v_emps_tree” named statement was used more than once and CBO has decided, that materializing the result of the query will be cheaper, than executing it many times.

Tracing the 10046 event, will even show the statement used for this temporary table creation:

```
CREATE GLOBAL TEMPORARY TABLE "SYS"."SYS_TEMP_0FD9D6617_10B49D" ("C0"
  VARCHAR2(20),"C1" VARCHAR2(25),"C2" VARCHAR2(30),"C3" NUMBER(8,2),"C4"
  NUMBER,"C5" NUMBER(6) ) IN_MEMORY_METADATA CURSOR_SPECIFIC_SEGMENT STORAGE
  (OBJNO 4254950936 ) NOPARALLEL
```

In 10053 trace, we can see the usage of this temporary table:

```
kkoqbc: finish optimizing query block SEL$1 (#0)
Copy query block qb# -1 (<unnamed>) : SELECT /*+ CACHE_TEMP_TABLE(T1) */   "C0",
"C1", "C2", "C3", "C4", "C5" FROM "SYS"."SYS_TEMP_0FD9D661A_10B49D" T1
Registered qb: SEL$D67CB2D2 0xf14b1d48 (MAP QUERY BLOCK SEL$1)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SEL$D67CB2D2 nbfros=1 flg=0
    fro(0): flg=6 objn=0 hint_alias="T1"@"SEL$D67CB2D2"
```