SQL Query Performance Tuning Case Study

Tao Zuo, NPD Group

Introduction

SQL tuning is a phenomenally complex subject. There are many books and papers on this topic. From release to release, Oracle built more and more intelligence into its query optimizer to run a query in an optimal manner. That also says, to understand oracle's internal on how a query being run becomes more and more difficult. There are many tuning software and utilities in the market to try to make SQL tuning "a click away". However you still find times with feeling of helpless, and SQL tuning is still an art work.

Through this paper, I would like to share two cases encountered in real life practice that you may find the fundamental utilities are real gifts to crack the hard shell.

There are many factors affect a SQL performance: Hardware, Network, OS, DB, Application, etc. Bear in mind that system level tuning could benefit many queries. So don't forget your tuning path is from top to bottom: from system to query itself. Ensure the objects statistics is up to date is essential, as some query maybe very sensitive to the small changes on object statistics and most of the time refreshing the statistics solve the query performance problem immediately. If statistics staleness is not the issue, further SQL tuning analysis shall begin.

Case Study 1

Here is an example. We received a client's complaint that their report is running in a hour glass for almost 45 minutes. We exam database system load, and all other applications responses, all appeared normal, the issue was narrowed down to a specific query. Which query contributes to the hour glass then? In our practice, we found gv$sql_monitor is pretty handy:

```
SELECT inst_id,
         status,
         sid,
         session_serial#,
         username,
         module,
         action,
         service_name,
         client_identifier,
         client_info,
         program,
         cpu_time,
         queuing_time,
         elapsed_time, -- Elapsed time (in microseconds); updated as the statement executes
         sql_id,
         sql_text
    FROM gv$sql_monitor
   WHERE elapsed_time > 1000000 * 60 * 60
ORDER BY elapsed_time DESC;
```

By the elapsed_time, username, client_info, client_identifier, program, sql_text, etc. we may quickly locate the problematic query. It can also provide us some useful information on the INST_ID (instance ID for RAC database) that the query runs, its SID, etc. for later further analysis.

With the SID, and INST_ID gathered in the previous query, we may find SPID (OS process ID) from the following v$process and v$session:

```
select s.username, s.sid, s.osuser, s.program, s.machine, p.spid, s.status
from v$session s, v$process p
where s.paddr = p.addr
and s.sid in (922)
;
--------
30765
```

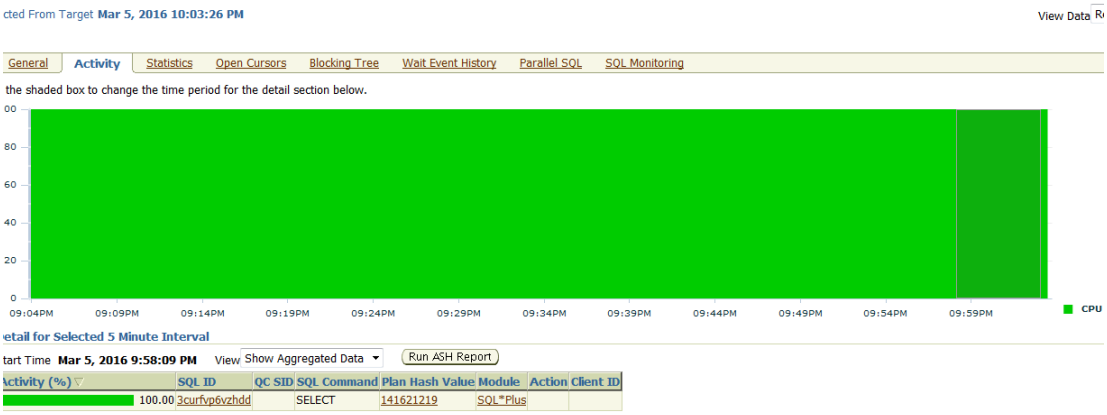
Note: with INST_ID, we can directly execute the above query in that instance.

With the SPID, we may take a look at how that process behaves on the server, using TOP utility:

```
top - 20:16:49 up 14 days, 11:55,  3 users,  load average: 1.92, 1.14, 0.92
Tasks: 730 total,   3 running, 727 sleeping,   0 stopped,   0 zombie
Cpu(s): 13.4%us,  0.2%sy,  0.0%ni, 86.4%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  132084668k total, 131218276k used,   866392k free,   264632k buffers
Swap: 24674288k total,  6000232k used, 18674056k free, 118870584k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
30765 oracle    25   0 38.4g 1.5g 1.5g R 100.0  1.2   0:50.22 oracle
30268 oracle    25   0 39.4g 2.3g 1.3g R 99.9  1.8   1:59.81 oracle
23122 oracle    16   0 38.4g 505m 500m S  4.3  0.4   0:04.22 oracle
 3112 oracle    15   0 38.4g 1.4g 1.4g S  2.0  1.1   0:19.43 oracle
```

In this case, the process 30765 is sitting on the first line of the TOP output. It is taking 100% of a single CPU. However overall server load is pretty low: load average is between 0.92 and 1.92, or there is almost no process needs to wait in the CPU run queue. In another word, the server does give sufficient resource to this process, and it still can't return with results in an expected time. From here we learned that this process is a CPU bound process.



This is how it looks in Oracle Enterprise Manager Session Detail page. We learned that the process was burning a whole CPU from 9:04PM - 9:59PM, over 50 minutes, with no return. From our experience, it is very possible that the query is running in a suboptimal plan with nested loops used in the wrong place killing the CPU.

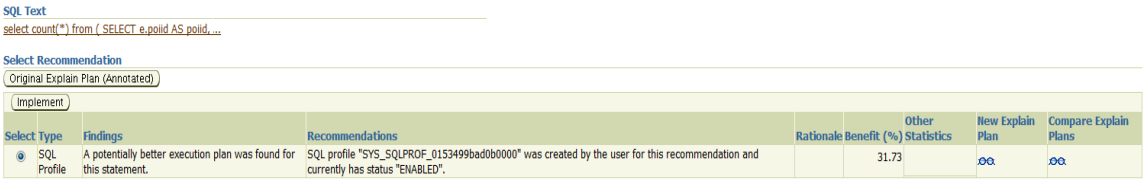
SQL Tuning Advisor has an interface embedded in Oracle Enterprise Manager. If you have Tuning Packs turned on, you may use it to tune the problematic SQL in your database.

To find out if Turning Pack is tuned on, use this:

```
SQL> show parameter control_management_pack_access
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
control_management_pack_access       string      DIAGNOSTIC+TUNING
```

Since we licensed with Diagnositc+Tuning Packs, and it is just a click away, it would be awesome if this click can get the problem solved.

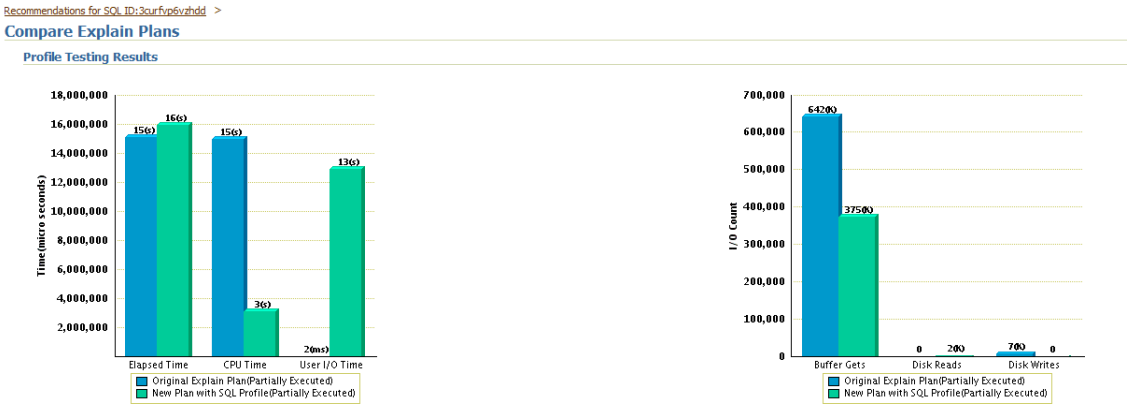
After running the SQL advisor, we got the recommendation as the following:



Select	Type	Findings	Recommendations	Rationale	Benefit (%)	Other Statistics	New Explain Plan	Compare Explain Plans
◉	SQL Profile	A potentially better execution plan was found for this statement.	SQL profile "SYS_SQLPROF_0153499bad0b0000" was created by the user for this recommendation and currently has status "ENABLED".		31.73			

Well, it found a better execution plan, which may improve the query performance by about 32%.

It also provided a nice graph on how the system resource usage would be reduced after applying its recommendation:



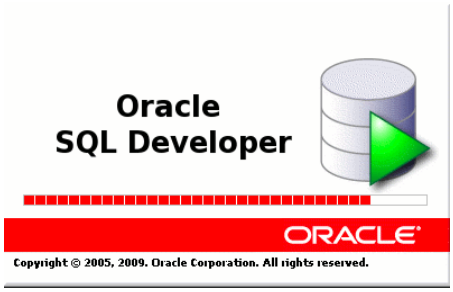
So we implement the recommendation, and run the query again. Here is how it looks from OEM:

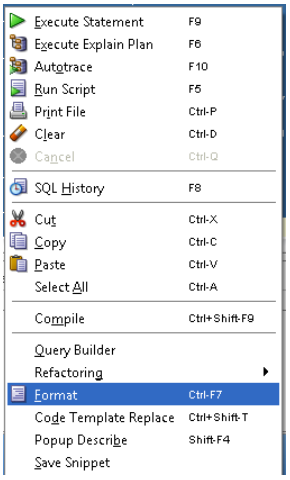
Query ran from 11:15AM till 12:08PM, about 53 minutes, no returns. Similar to its original execution plan, major time was pinning on CPU, no performance improvement. No silver bullet !

Here comes the SQL tuning fundamentals.

Format the query

Wait, does query format have anything to do with SQL performance? Well, No. Oracle can parse a query in any format, and execute it. However making the code readable is importance to appeal its structure, so we understand the logic and to make the tuning work easier. Either SQL developer or TOAD provides the feature of "Format the code" for the developers, so formatting the code is really just a click away.





The query that we are dealing with has 92 lines, many layers of subqueries, and a few views in addition to plenty tables.

Before formatting, the query looks cumbersome, and the logic is hard to catch.

```
SELECT e.poiid AS poiid, sopout101code AS
sop_out101_code, sopout101name AS sop_out101_name,
count11name AS count11, count11code AS count11_code, 0
AS matched_count11_code, out101code AS out101_code,
out101name AS out101_name, subchan102_name,
chan102_name, out101division AS out101_division,
out101de8888mmmm AS out101_de8888mmmm,
out101subde8888mmmm AS out101_sub_de8888mmmm,
out101class AS out101_class, out101subclass AS
out101_sub_class, uks AS uks, rfmcode AS rfm_code,
out101bf123 AS out101_bf123, out101itemnumber AS
out101_item_number, out101description AS
out101_description, rfmcodetype AS rfm_code_type,
manu4444tttrcodestatus AS manu4444tttr_code_status,
out101descriptionsupplement AS
out101_description_supplement, unitssold AS
```

…

```
categdogname, categdogcodeset, origin_count11code,
fld01code, fld01name, bf123code, bf123name) e, (SELECT
* FROM vw_djw_sopout101s WHERE sopout101code = 253355)
h WHERE h.sopout101code(+) = e.sopout101code AND
h.out101division(+) = NVL (e.out101division, '*') AND
h.out101de8888mmmm(+) = NVL (e.out101de8888mmmm, '*')
AND h.out101subde8888mmmm(+) = NVL
(e.out101subde8888mmmm, '*') AND h.out101class(+) =
NVL (e.out101class, '*') AND h.out101subclass(+) = NVL
(e.out101subclass, '*')) e,
vw_djw_categdoghierarchy_8_0 cat,
vw_djw_out101hierlist_8_0 OUT WHERE 1 = 1 AND
e.categdogcode = cat.code(+) AND e.out101code =
OUT.out101_code(+)) e, xyz_sopout101item_attributes a
WHERE e.sopout101code = a.sopout101(+) AND e.poiid =
a.poiid(+);
```

After we format the code, the query becomes 110 lines, however, the query structure is much more obvious. We can clearly see each subquery layer, and not being confused by those repeat used subquery block aliases.

```
SELECT e.poiid  AS poiid,
  sopout101code AS sop_out101_code,
  sopout101name AS sop_out101_name,
  count11name   AS count11,
  count11code   AS count11_code,
  0             AS matched_count11_code,
  out101code    AS out101_code,
  out101name    AS out101_name,
  subchan102_name,
  chan102_name,
  out101division              AS out101_division,
  out101de8888mmmm            AS out101_de8888mmmm,
  out101subde8888mmmm         AS out101_sub_de8888mmmm,
  out101class                 AS out101_class,
  out101subclass              AS out101_sub_class,
  uks                         AS uks,
  rfmcode                     AS rfm_code,
  out101bf123                 AS out101_bf123,
  out101itemnumber            AS out101_item_number,
  out101description           AS out101_description,
  rfmcodetype                 AS rfm_code_type,
  manu4444tttrcodestatus      AS manu4444tttr_code_status,
  out101descriptionsupplement AS out101_description_supplement,
  unitssold                   AS units_sold,
  totalvalue                  AS total_value,
  averagepriceAS average price,
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

…

```
        (SELECT calendar_id FROM xyz_ret999ers WHERE ret999er_id = sopoutl01code
        )                                   AS calendar_id,
        MAX (outl01descriptionsupplement ) AS outl01descriptionsupplement,
        MAX (privatelabel)                  AS privatelabel
      FROM
        (SELECT *
        FROM
          (SELECT a.*
          FROM vw_djw_noit55555s a
          WHERE sopoutl01code = 253355
          AND status NOT     IN (91, -2)
          AND status NOT     IN (99, 95, 94, 97)
          AND status NOT     IN (11, 12, 15)
          AND hidden_exp      = 0
          AND cutoff_exp      = 0
          AND ppmonth         = 549
          )
        )
      GROUP BY lrowid,
        poiid,
        sopoutl01code,
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

…

```
        categdogname,
        categdogcodeset,
        origin_countllcode,
        fld01code,
        fld01name,
        bfl23code,
        bfl23name
      ) e,
      (SELECT * FROM vw_djw_sopoutl01s WHERE sopoutl01code = 253355
      ) h
    WHERE h.sopoutl01code(+)      = e.sopoutl01code
    AND h.outl01division(+)       = NVL (e.outl01division, '*')
    AND h.outl01de8888mmmm(+)     = NVL (e.outl01de8888mmmm, '*')
    AND h.outl01subde8888mmmm(+) = NVL (e.outl01subde8888mmmm, '*')
    AND h.outl01class(+)          = NVL (e.outl01class, '*')
    AND h.outl01subclass(+)       = NVL (e.outl01subclass, '*')
    ) e,
    vw_djw_categdoghierarchy_8_0 cat,
    vw_djw_outl01hierlist_8_0 OUT
  WHERE 1              = 1
  AND e.categdogcode = cat.code(+)
  AND e.outl01code   = OUT.outl01_code(+)
  ) e,
  xyz_sopoutl01item_attributes a
WHERE e.sopoutl01code = a.sopoutl01(+)
AND e.poiid           = a.poiid(+);
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Here the 3 alias of “e” and 2 “a” represent different query blocks. This needs to be corrected, not only for the code readability, but also the query tuning. Since we have learned this query has 4 layers, therefore each layer block is labeled with the layer# as its alias’s suffix:

```
        )                                   AS calendar_id,
        MAX (outl01descriptionsupplement ) AS outl01descriptionsupplement,
        MAX (privatelabel)                  AS privatelabel
      FROM
        (SELECT *
        FROM
          (SELECT a4.*
          FROM vw_djw_exceptions a4
          WHERE sopoutl01code = 253355
          AND status NOT     IN (91, -2)
          AND status NOT     IN (99, 95, 94, 97)
          AND status NOT     IN (11, 12, 15)
          AND hidden_exp      = 0
          AND cutoff_exp      = 0
          AND ppmonth         = 549
          )
        )
      GROUP BY lrowid,
        poiid,
        sopoutl01code,
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

…

```
        bfl23name
      ) e3,
      (SELECT * FROM vw_djw_sopoutl01s WHERE sopoutl01code = 253355
      ) h
    WHERE h.sopoutl01code(+)      = e3.sopoutl01code
    AND h.outl01division(+)       = NVL (e3.outl01division, '*')
    AND h.outl01de8888mmmm(+)     = NVL (e3.outl01de8888mmmm, '*')
    AND h.outl01subde8888mmmm(+)  = NVL (e3.outl01subde8888mmmm, '*')
    AND h.outl01class(+)          = NVL (e3.outl01class, '*')
    AND h.outl01subclass(+)       = NVL (e3.outl01subclass, '*')
    ) e2,
    vw_djw_categdoghierarchy_8_0 cat,
    vw_djw_outl01hierlist_8_0 OUT
  WHERE 1              = 1
  AND e2.categdogcode = cat.code(+)
  AND e2.outl01code    = OUT.outl01_code(+)
  ) e,
  xyz_sopoutl01item_attributes a
WHERE e.sopoutl01code = a.sopoutl01(+)
AND e.poiid           = a.poiid(+);
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Explain Plan

Now, with the query formatted, we use the following to generate the query explain plan:

explain plan set statement_id='f' for
<QUERY>
;

And We use the following to show the explain plan output:

select * from table(dbms_xplan.display);

Here is the query explain plan:

```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------------------------------
Plan hash value: 3464502560

-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name                        | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                             | 94916 |  480M|       |  287K  (1)| 01:07:10 |       |       |
|   1 |  TABLE ACCESS BY GLOBAL INDEX ROWID          | XYZ_CODES                   |     1 |   25 |       |     3   (0)| 00:00:01 | ROWID | ROWID |
|*  2 |   INDEX UNIQUE SCAN                          | XYZ_CODES_UK01              |     1 |      |       |     2   (0)| 00:00:01 |       |       |
|   3 |  TABLE ACCESS BY GLOBAL INDEX ROWID          | XYZ_CODES                   |     1 |   25 |       |     3   (0)| 00:00:01 | ROWID | ROWID |
|*  4 |   INDEX UNIQUE SCAN                          | XYZ_CODES_UK01              |     1 |      |       |     2   (0)| 00:00:01 |       |       |
|   5 |    TABLE ACCESS BY INDEX ROWID               | XYZ_RET999ERS               |     1 |   10 |       |     2   (0)| 00:00:01 |       |       |
|*  6 |     INDEX UNIQUE SCAN                        | XYZ_RET999ERS_PK            |     1 |      |       |     1   (0)| 00:00:01 |       |       |
|   7 |  TABLE ACCESS BY INDEX ROWID                 | XYZ_RET999ERS               |     1 |   10 |       |     2   (0)| 00:00:01 |       |       |
|*  8 |   INDEX UNIQUE SCAN                          | XYZ_RET999ERS_PK            |     1 |      |       |     1   (0)| 00:00:01 |       |       |
|   9 |  TABLE ACCESS BY GLOBAL INDEX ROWID          | XYZ_CODES                   |     1 |   25 |       |     3   (0)| 00:00:01 | ROWID | ROWID |
|* 10 |   INDEX UNIQUE SCAN                          | XYZ_CODES_UK01              |     1 |      |       |     2   (0)| 00:00:01 |       |       |
|  11 |  TABLE ACCESS BY INDEX ROWID                 | XYZ_SOPDATASTATUSVALUES     |     1 |   36 |       |     1   (0)| 00:00:01 |       |       |
|* 12 |   INDEX UNIQUE SCAN                          | SYS_C002645765              |     1 |      |       |     0   (0)| 00:00:01 |       |       |
|  13 |  PARTITION LIST SINGLE                       |                             |     1 |   44 |       |     2   (0)| 00:00:01 |   KEY |   KEY |
|* 14 |   TABLE ACCESS BY LOCAL INDEX ROWID          | XYZ_CODES                   |     1 |   44 |       |     2   (0)| 00:00:01 |    59 |    59 |
|* 15 |    INDEX RANGE SCAN                          | XYZ_CODES_FK01              |     1 |      |       |     1   (0)| 00:00:01 |    59 |    59 |
|  16 |  PARTITION LIST SINGLE                       |                             |     1 |   28 |       |     2   (0)| 00:00:01 |   KEY |   KEY |
|  17 |   TABLE ACCESS BY LOCAL INDEX ROWID          | XYZ_CODES                   |     1 |   28 |       |     2   (0)| 00:00:01 |    59 |    59 |
|* 18 |    INDEX UNIQUE SCAN                         | XYZ_CODES_PK                |     1 |      |       |     1   (0)| 00:00:01 |    59 |    59 |
|  19 |  PARTITION LIST SINGLE                       |                             |     1 |   27 |       |     2   (0)| 00:00:01 |   KEY |   KEY |
|  20 |   TABLE ACCESS BY LOCAL INDEX ROWID          | XYZ_CODES                   |     1 |   27 |       |     2   (0)| 00:00:01 |    59 |    59 |
|* 21 |    INDEX UNIQUE SCAN                         | XYZ_CODES_PK                |     1 |      |       |     1   (0)| 00:00:01 |    59 |    59 |
|  22 |  TABLE ACCESS BY GLOBAL INDEX ROWID          | XYZ_CODES                   |     1 |   25 |       |     3   (0)| 00:00:01 | ROWID | ROWID |
|* 23 |   INDEX UNIQUE SCAN                          | XYZ_CODES_UK01              |     1 |      |       |     2   (0)| 00:00:01 |       |       |
```

```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------------------------------------------------
|  24 |  SORT UNIQUE                                |                                |     3 |    33 |        |    30   (4)| 00:00:01 |       |       |
|* 25 |   TABLE ACCESS BY INDEX ROWID               | XYZ_CALENDAR_WEEKS             |     3 |    33 |        |    29   (0)| 00:00:01 |       |       |
|* 26 |    INDEX RANGE SCAN                         | XYZ_CALENDAR_WEEKS_FK01        |  4918 |       |        |     5   (0)| 00:00:01 |       |       |
|* 27 |    TABLE ACCESS BY INDEX ROWID              | XYZ_CALENDAR_WEEKS             |     1 |    15 |        |     2   (0)| 00:00:01 |       |       |
|* 28 |     INDEX UNIQUE SCAN                       | XYZ_CALENDAR_WEEKS_PK          |     1 |       |        |     1   (0)| 00:00:01 |       |       |
|  29 |  SORT UNIQUE                                |                                |     3 |    30 |        |    30   (4)| 00:00:01 |       |       |
|* 30 |   TABLE ACCESS BY INDEX ROWID               | XYZ_CALENDAR_WEEKS             |     3 |    30 |        |    29   (0)| 00:00:01 |       |       |
|* 31 |    INDEX RANGE SCAN                         | XYZ_CALENDAR_WEEKS_FK01        |  4918 |       |        |     5   (0)| 00:00:01 |       |       |
|* 32 |    TABLE ACCESS BY INDEX ROWID              | XYZ_CALENDAR_WEEKS             |     1 |    14 |        |     2   (0)| 00:00:01 |       |       |
|* 33 |     INDEX UNIQUE SCAN                       | XYZ_CALENDAR_WEEKS_PK          |     1 |       |        |     1   (0)| 00:00:01 |       |       |
|* 34 |  TABLE ACCESS BY INDEX ROWID                | XYZ_CALENDAR_WEEKS             |     1 |    14 |        |     2   (0)| 00:00:01 |       |       |
|* 35 |   INDEX UNIQUE SCAN                         | XYZ_CALENDAR_WEEKS_PK          |     1 |       |        |     1   (0)| 00:00:01 |       |       |
|* 36 |  TABLE ACCESS BY INDEX ROWID                | XYZ_CALENDAR_WEEKS             |     1 |    19 |        |     2   (0)| 00:00:01 |       |       |
|* 37 |   INDEX UNIQUE SCAN                         | XYZ_CALENDAR_WEEKS_PK          |     1 |       |        |     1   (0)| 00:00:01 |       |       |
|* 38 |  TABLE ACCESS BY INDEX ROWID                | XYZ_CALENDAR_WEEKS             |     1 |    19 |        |     2   (0)| 00:00:01 |       |       |
|* 39 |   INDEX UNIQUE SCAN                         | XYZ_CALENDAR_WEEKS_PK          |     1 |       |        |     1   (0)| 00:00:01 |       |       |
|  40 |  TABLE ACCESS BY INDEX ROWID                | XYZ_RET999ERS                  |     1 |    10 |        |     2   (0)| 00:00:01 |       |       |
|* 41 |   INDEX UNIQUE SCAN                         | XYZ_RET999ERS_PK               |     1 |       |        |     1   (0)| 00:00:01 |       |       |
|* 42 |  TABLE ACCESS BY GLOBAL INDEX ROWID         | XYZ_CODES                      |     1 |    42 |        |     3   (0)| 00:00:01 | ROWID | ROWID |
|* 43 |   INDEX UNIQUE SCAN                         | XYZ_CODES_UK01                 |     1 |       |        |     2   (0)| 00:00:01 |       |       |
|  44 |  TABLE ACCESS BY INDEX ROWID                | XYZ_RET999ERS                  |     1 |     8 |        |     2   (0)| 00:00:01 |       |       |
|* 45 |   INDEX UNIQUE SCAN                         | XYZ_RET999ERS_PK               |     1 |       |        |     1   (0)| 00:00:01 |       |       |
|* 46 |  HASH JOIN RIGHT OUTER                      |                                | 94916 |  480M|        |   287K  (1)| 01:07:10 |       |       |
|  47 |   VIEW                                      | VW_DJW_OUT101HIERLIST_8_0      |     1 |   139 |        |     4   (0)| 00:00:01 |       |       |
|  48 |    NESTED LOOPS                             |                                |     1 |   149 |        |     4   (0)| 00:00:01 |       |       |
|  49 |     NESTED LOOPS                            |                                |     1 |   149 |        |     4   (0)| 00:00:01 |       |       |
|  50 |      NESTED LOOPS                           |                                |     1 |   121 |        |     3   (0)| 00:00:01 |       |       |
|  51 |       NESTED LOOPS                          |                                |     1 |    93 |        |     2   (0)| 00:00:01 |       |       |
|* 52 |        HASH JOIN                            |                                |     1 |    82 |        |     2   (0)| 00:00:01 |       |       |
|  53 |         PARTITION LIST SINGLE               |                                |     5 |   205 |        |     1   (0)| 00:00:01 |   KEY |   KEY |
|* 54 |          INDEX RANGE SCAN                   | XYZ_SUBCODES_PK                |     5 |   205 |        |     1   (0)| 00:00:01 |    59 |    59 |
|  55 |         PARTITION LIST SINGLE               |                                |   135 |  5535 |        |     1   (0)| 00:00:01 |   KEY |   KEY |
|* 56 |          INDEX RANGE SCAN                   | XYZ_SUBCODES_PK                |   135 |  5535 |        |     1   (0)| 00:00:01 |    59 |    59 |
|  57 |        PARTITION LIST SINGLE                |                                |     1 |    11 |        |     0   (0)| 00:00:01 |   KEY |   KEY |
|* 58 |         INDEX UNIQUE SCAN                   | XYZ_CODES_PK                   |     1 |    11 |        |     0   (0)| 00:00:01 |    59 |    59 |
|  59 |       PARTITION LIST SINGLE                 |                                |     1 |    28 |        |     1   (0)| 00:00:01 |   KEY |   KEY |
|  60 |        TABLE ACCESS BY LOCAL INDEX ROWID    | XYZ_CODES                      |     1 |    28 |        |     1   (0)| 00:00:01 |    59 |    59 |
|* 61 |         INDEX UNIQUE SCAN                   | XYZ_CODES_PK                   |     1 |       |        |     0   (0)| 00:00:01 |    59 |    59 |
|  62 |      PARTITION LIST SINGLE                  |                                |     1 |       |        |     0   (0)| 00:00:01 |   KEY |   KEY |
|* 63 |       INDEX UNIQUE SCAN                     | XYZ_CODES_PK                   |     1 |       |        |     0   (0)| 00:00:01 |    59 |    59 |
|  64 |      TABLE ACCESS BY LOCAL INDEX ROWID      | XYZ_CODES                      |     1 |    28 |        |     1   (0)| 00:00:01 |    59 |    59 |
|* 65 |    HASH JOIN RIGHT OUTER                    |                                | 94916 |  468M|        |   287K  (1)| 01:07:10 |       |       |
|  66 |     PARTITION LIST SINGLE                   |                                |    94 | 12502 |        |     5   (0)| 00:00:01 |   KEY |   KEY |
|  67 |      VIEW                                   |                                |    94 | 12502 |        |     5   (0)| 00:00:01 |       |       |
|  68 |       TABLE ACCESS BY LOCAL INDEX ROWID     | XYZ_BUSDE8888MMMMCLASSES       |    94 |  4700 |        |     5   (0)| 00:00:01 |    35 |    35 |
|* 69 |        INDEX RANGE SCAN                     | XYZ_BUSDE8888MMMMCLASSES_IE01  |     1 |       |        |     2   (0)| 00:00:01 |    35 |    35 |
|* 70 |     HASH JOIN RIGHT OUTER                   |                                | 94916 |  456M|        |   287K  (1)| 01:07:10 |       |       |
|  71 |      VIEW                                   |                                |     1 |    99 |        |    31   (4)| 00:00:01 |       |       |
|  72 |       HASH UNIQUE                           |                                |     1 |   156 |        |    31   (4)| 00:00:01 |       |       |
|  73 |        CONCATENATION                        |                                |       |       |        |            |          |       |       |
|  74 |         NESTED LOOPS                        |                                |     1 |   156 |        |    15   (0)| 00:00:01 |       |       |
|  75 |          NESTED LOOPS                       |                                |     1 |   156 |        |    15   (0)| 00:00:01 |       |       |
|* 76 |           HASH JOIN                         |                                |     1 |   112 |        |    14   (0)| 00:00:01 |       |       |
|* 77 |            HASH JOIN                        |                                |    32 |  2720 |        |    10   (0)| 00:00:01 |       |       |
|  78 |             PARTITION LIST SINGLE           |                                |    32 |  1312 |        |     1   (0)| 00:00:01 |   KEY |   KEY |
|* 79 |              INDEX RANGE SCAN               | XYZ_SUBCODES_PK                |    32 |  1312 |        |     1   (0)| 00:00:01 |    59 |    59 |
|  80 |             PARTITION LIST SINGLE           |                                |  2809 |  120K|        |     9   (0)| 00:00:01 |   KEY |   KEY |
|* 81 |              TABLE ACCESS FULL              | XYZ_CODES                      |  2809 |  120K|        |     9   (0)| 00:00:01 |    59 |    59 |
|  82 |            PARTITION LIST SINGLE            |                                |    43 |  1161 |        |     4   (0)| 00:00:01 |   KEY |   KEY |
|  83 |            INLIST ITERATOR                  |                                |       |       |        |            |          |       |       |
|* 84 |             TABLE ACCESS BY LOCAL INDEX ROWID | XYZ_CODES                      |    43 |  1161 |        |     4   (0)| 00:00:01 |    59 |    59 |
|* 85 |              INDEX RANGE SCAN               | XYZ_CODES_FK01                 |     1 |       |        |     2   (0)| 00:00:01 |    59 |    59 |
|  86 |          PARTITION LIST SINGLE              |                                |     1 |       |        |     0   (0)| 00:00:01 |   KEY |   KEY |
|* 87 |          INDEX UNIQUE SCAN                  | XYZ_CODES_PK                   |     1 |       |        |     0   (0)| 00:00:01 |    59 |    59 |
|* 88 |         TABLE ACCESS BY LOCAL INDEX ROWID   | XYZ_CODES                      |     1 |    44 |        |     1   (0)| 00:00:01 |    59 |    59 |
|  89 |        NESTED LOOPS                         |                                |     1 |   156 |        |    15   (0)| 00:00:01 |       |       |
|  90 |         NESTED LOOPS                        |                                |     1 |   156 |        |    15   (0)| 00:00:01 |       |       |
|* 91 |          HASH JOIN                          |                                |     1 |   112 |        |    14   (0)| 00:00:01 |       |       |
|* 92 |           HASH JOIN                         |                                |    32 |  2720 |        |    10   (0)| 00:00:01 |       |       |
|  93 |            PARTITION LIST SINGLE            |                                |    32 |  1312 |        |     1   (0)| 00:00:01 |   KEY |   KEY |
|* 94 |             INDEX RANGE SCAN                | XYZ_SUBCODES_PK                |    32 |  1312 |        |     1   (0)| 00:00:01 |    59 |    59 |
|  95 |            PARTITION LIST SINGLE            |                                |  2809 |  120K|        |     9   (0)| 00:00:01 |   KEY |   KEY |
|* 96 |             TABLE ACCESS FULL               | XYZ_CODES                      |  2809 |  120K|        |     9   (0)| 00:00:01 |    59 |    59 |
|  97 |           PARTITION LIST SINGLE             |                                |    43 |  1161 |        |     4   (0)| 00:00:01 |   KEY |   KEY |
|  98 |           INLIST ITERATOR                   |                                |       |       |        |            |          |       |       |
|* 99 |            TABLE ACCESS BY LOCAL INDEX ROWID | XYZ_CODES                      |    43 |  1161 |        |     4   (0)| 00:00:01 |    59 |    59 |
|*100 |             INDEX RANGE SCAN                | XYZ_CODES_FK01                 |     1 |       |        |     2   (0)| 00:00:01 |    59 |    59 |
| 101 |          PARTITION LIST SINGLE              |                                |     1 |       |        |     0   (0)| 00:00:01 |   KEY |   KEY |
|*102 |          INDEX UNIQUE SCAN                  | XYZ_CODES_PK                   |     1 |       |        |     0   (0)| 00:00:01 |    59 |    59 |
|*103 |         TABLE ACCESS BY LOCAL INDEX ROWID   | XYZ_CODES                      |     1 |    44 |        |     1   (0)| 00:00:01 |    59 |    59 |
|*104 |     HASH JOIN RIGHT OUTER                   |                                | 94916 |  447M|   91M| 287K  (1)| 01:07:09 |       |       |
| 105 |      PARTITION LIST ALL                     |                                  2277K|   65M|        |  7013   (1)| 00:01:39 |     1 |  3191 |
| 106 |       TABLE ACCESS FULL                     | XYZ_SOPOUT101ITEM_ATTRIBUTES   2277K|   65M|        |  7013   (1)| 00:01:39 |     1 |  3191 |
| 107 |      VIEW                                   |                                | 94916 |  444M|        |   257K  (1)| 01:00:10 |       |       |
| 108 |       TABLE ACCESS BY GLOBAL INDEX ROWID    | XYZ_CODES                      |     1 |    25 |        |     3   (0)| 00:00:01 | ROWID | ROWID |
|*109 |        INDEX UNIQUE SCAN                    | XYZ_CODES_UK01                 |     1 |       |        |     2   (0)| 00:00:01 |       |       |
| 110 |       HASH GROUP BY                         |                                | 94916 |  248M|  296M| 257K  (1)| 01:00:10 |       |       |
| 111 |        VIEW                                 |                                | 94916 |  248M|        |   223K  (1)| 00:52:15 |       |       |
| 112 |         NESTED LOOPS                        |                                | 94916 |   61M|        |   223K  (1)| 00:52:15 |       |       |
| 113 |          NESTED LOOPS                       |                                | 94916 |   61M|        |   223K  (1)| 00:52:15 |       |       |
|*114 |           HASH JOIN                         |                                | 94215 |   39M|   27M| 35414   (1)| 00:08:16 |       |       |
| 115 |            PART JOIN FILTER CREATE          | :BF0000                        | 94058 |   25M|        | 20176   (1)| 00:04:43 |       |       |
|*116 |             HASH JOIN                       |                                | 94058 |   25M|        | 20176   (1)| 00:04:43 |       |       |
| 117 |              PART JOIN FILTER CREATE        | :BF0001                        |     1 |   26 |        |     5   (0)| 00:00:01 |       |       |
| 118 |               PARTITION LIST SINGLE         |                                |     1 |   26 |        |     5   (0)| 00:00:01 |   KEY |   KEY |
```

```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------------------------------------------------------
|*119 |                TABLE ACCESS BY LOCAL INDEX ROWID| XYZ_SOPDATALOADSTATUS         |     1 |    26 |       |     5   (0)| 00:00:01 |    35 |    35 |
|*120 |                 INDEX RANGE SCAN                | XYZ_SOPDATALOADSTATUS_PK      |     1 |       |       |     1   (0)| 00:00:01 |    35 |    35 |
| 121 |              PARTITION RANGE SINGLE             |                               | 2818K|  706M|       | 20164   (1)| 00:04:43 |KEY(AP)|KEY(AP)|
|*122 |               TABLE ACCESS FULL                 | XYZ_SOPUNMATCHEDDATA          | 2818K|  706M|       | 20164   (1)| 00:04:43 |   249 |   249 |
| 123 |            PARTITION LIST SINGLE                |                               | 1353K|  188M|       |  5372   (1)| 00:01:16 |KEY(AP)|KEY(AP)|
|*124 |             TABLE ACCESS FULL                   | XYZ_SOPEXCEPTIONS             | 1353K|  188M|       |  5372   (1)| 00:01:16 |    35 |    35 |
|*125 |           INDEX UNIQUE SCAN                     | XYZ_SOPOUT101ITEMS_UK01       |     1 |       |       |     1   (0)| 00:00:01 |       |       |
|*126 |          TABLE ACCESS BY GLOBAL INDEX ROWID     | XYZ_SOPOUT101ITEMS            |     1 |   240 |       |     2   (0)| 00:00:01 | ROWID | ROWID |
-----------------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("CODE"=:B1)
…
```

The last section list all the predicate information. We wanted to make sure the effective filter has applied in the early stage of the query as much as possible, so the query cost can be as small as possible.

Let's look at the explain plan output from top to bottom. Line ID0 gives out the query last cumulative time in estimation. Here it says it is 1:07hr. We navigate down to find the last line with time close to this, which is line ID111, with estimation query time 52:15min. According to this, if we can make this query block execution time dropped, the whole query time shall be dropped. Therefore we know this is where we need to start to work with.

We learned that there are 4 tables participated in this query block. First things first, we want to verify the 4 tables statistics are not stale. With dictionary view DBA_TAB_MODIFICATIONS, we can check if these tables are updated since last statistics gathered. If the table statistics is stale for some reason, we can refresh it with DBMS_STATS package. By default, oracle use 10% change rule to define if an object has stale statistics or not. However, in our practice, that is just a general rule of thumb, you have to learn from your system, every object could have different stale statistics threshold.

After we clear the tables' statistics staleness, we move on to verify if the optimizer is correct by executing that query block.

Analysis

Now the question is what part of the query equivalent to this query block between line ID111 to line ID126 in the explain plan? Benefiting from the query formatting, this becomes much easier. The only little problem is those 4 tables don't show up in the query. We know there are a few views being used in the query, so they must embedded in those views.

```
| 111 |        VIEW                                 |                               | 94916 |  248M|       |  223K  (1)| 00:52:15 |       |       |
| 112 |         NESTED LOOPS                        |                               | 94916 |   61M|       |  223K  (1)| 00:52:15 |       |       |
| 113 |          NESTED LOOPS                       |                               | 94916 |   61M|       |  223K  (1)| 00:52:15 |       |       |
|*114 |           HASH JOIN                         |                               | 94215 |   39M|   27M| 35414  (1)| 00:08:16 |       |       |
| 115 |            PART JOIN FILTER CREATE          | :BF0000                       | 94058 |   25M|       | 20176  (1)| 00:04:43 |       |       |
|*116 |             HASH JOIN                       |                               | 94058 |   25M|       | 20176  (1)| 00:04:43 |       |       |
| 117 |              PART JOIN FILTER CREATE        | :BF0001                       |     1 |   26 |       |     5  (0)| 00:00:01 |       |       |
| 118 |               PARTITION LIST SINGLE         |                               |     1 |   26 |       |     5  (0)| 00:00:01 |   KEY |   KEY |
|*119 |                TABLE ACCESS BY LOCAL INDEX ROWID| XYZ_SOPDATALOADSTATUS         |     1 |   26 |       |     5  (0)| 00:00:01 |    35 |    35 |
|*120 |                 INDEX RANGE SCAN            | XYZ_SOPDATALOADSTATUS_PK      |     1 |       |       |     1  (0)| 00:00:01 |    35 |    35 |
| 121 |              PARTITION RANGE SINGLE         |                               | 2818K|  706M|       | 20164  (1)| 00:04:43 |KEY(AP)|KEY(AP)|
|*122 |               TABLE ACCESS FULL             | XYZ_SOPUNMATCHEDDATA          | 2818K|  706M|       | 20164  (1)| 00:04:43 |   249 |   249 |
| 123 |            PARTITION LIST SINGLE            |                               | 1353K|  188M|       |  5372  (1)| 00:01:16 |KEY(AP)|KEY(AP)|
|*124 |             TABLE ACCESS FULL               | XYZ_SOPEXCEPTIONS             | 1353K|  188M|       |  5372  (1)| 00:01:16 |    35 |    35 |
|*125 |           INDEX UNIQUE SCAN                 | XYZ_SOPOUT101ITEMS_UK01       |     1 |       |       |     1  (0)| 00:00:01 |       |       |
|*126 |          TABLE ACCESS BY GLOBAL INDEX ROWID | XYZ_SOPOUT101ITEMS            |     1 |  240 |       |     2  (0)| 00:00:01 | ROWID | ROWID |
-----------------------------------------------------------------------------------------------------------------------------------------------------
```

With this query, we can easily pull the view that those tables constructed with:

```
BEGIN
   FOR V
      IN (SELECT VIEW_NAME, TEXT
            FROM DBA_VIEWS
           WHERE     OWNER = 'DICT_MGR'
                 AND VIEW_NAME IN
                        (UPPER ('vw_djw_categdoghierarchy_8_0'),
                         UPPER ('vw_djw_out101hierlist_8_0'),
                         UPPER ('vw_djw_sopout101s'),
                         UPPER ('vw_djw_exceptions')))
   LOOP
      IF     INSTR (UPPER (V.TEXT), 'SOPDATALOADSTATUS') > 0
         AND INSTR (UPPER (V.TEXT), 'SOPUNMATCHEDDATA') > 0
         AND INSTR (UPPER (V.TEXT), 'SOPEXCEPTIONS') > 0
         AND INSTR (UPPER (V.TEXT), 'SOPOUT101ITEMS') > 0
      THEN
         DBMS_OUTPUT.PUT_LINE (V.VIEW_NAME);
      END IF;
   END LOOP;
END;
/
---------------
VW_DJW_EXCEPTIONS
```

With this we can find the query block (layer4) and execute it:

```
SELECT COUNT(*)
FROM
  (SELECT a4.*
  FROM vw_djw_exceptions a4
  WHERE sopout101code = 253355
  AND status NOT     IN (91, -2)
  AND status NOT     IN (99, 95, 94, 97)
  AND status NOT     IN (11, 12, 15)
  AND hidden_exp      = 0
  AND cutoff_exp      = 0
  AND ppmonth         = 549
  ) ;

COUNT(*)
----------
229021 rows

Elapsed:  23sec.
```

Well, looks like optimizer's estimation and the real scenario has a big discrepancy. This query block's execution is fine.

We expend to its outer block:

```
SELECT CONT(*) FROM (
SELECT lrowid,
        poiid,
        …
      FROM
        (SELECT *
        FROM
          (<layer4>
          )
        )
      GROUP BY lrowid,
        poiid,
        …
      );

COUNT(*)
----------
162869 rows

Elapsed:  19sec
```

Again, the problem is not in the layer.

We extended to its outer block (layer 3):

```
  SELECT COUNT(*) FROM(
  SELECT e3.*,
        …
        h.out101code,
        …
      FROM
        (<layer3>) e3,
        (SELECT * FROM vw_djw_sopout101s WHERE sopout101code = 253355
        ) h
      WHERE h.sopout101code(+)     = e3.sopout101code
      AND h.out101division(+)      = NVL (e3.out101division, '*')
      AND h.out101de8888mmmm(+)    = NVL (e3.out101de8888mmmm, '*')
      AND h.out101subde8888mmmm(+) = NVL (e3.out101subde8888mmmm, '*')
      AND h.out101class(+)         = NVL (e3.out101class, '*')
      AND h.out101subclass(+)      = NVL (e3.out101subclass, '*')
      );

COUNT(*)
----------
162869 rows

Elapsed:  6sec
```

This block runs fine.

Further extended out (layer2):

```
SELECT COUNT(*) FROM (
SELECT
  (SELECT code
  FROM xyz_codes
  WHERE bus466ssid = 471
  AND codeset      = 'djw_a_subcategdog'
  AND NAME         = categdogname
  ) AS selected_categdog_code,
  e2.*,
  cat.*,
  OUT.*
FROM (layer 2)
) e2,
vw_djw_categdoghierarchy_8_0 cat,
vw_djw_out101hierlist_8_0 OUT
  WHERE 1            = 1
  AND e2.categdogcode = cat.code(+)
  AND e2.out101code   = OUT.out101_code(+)
 ) ;

COUNT(*)
----------
162869 rows

Elapsed: 7sec
```

Now we learn up to this query block there is no performance problem. The problem is sitting on the top layer.

10046 Trace and tkprof

Let's generate a 10046 trace for the query. It provides the statistics for the real run. Before we start the trace, we need to make sure timed_statistics is turned to true. The statistics_level is set to typical. To make the trace file easier to be found, we set the tracefile_identifier. After that, we run the query. The trace file is generated in the same directory where the database instance alert.log is.

- ALTER SESSION SET TIMED_STATISTICS = TRUE;
- ALTER SESSION SET STATISTICS_LEVEL=TYPICAL;
- alter session set tracefile_identifier = 'SQL_badsql';
- alter session set events '10046 trace name context forever, level 12';
- <run query>
- alter session set events '10046 trace name context off';
- $ORACLE_BASE/diag/rdbms/<db>/<inst>/trace/<inst>_ora_xxxx_SQL_badsql.trc

Tkprof is the utility to interpret 10046 trace file. To get more details of this tool, you may refer to metalink doc: "tkprof Interpretation (9i and above)" (Doc ID 760786.1).

Here is how you get the tkprof output:

```
$ tkprof <inst>_ora_xxxx_SQL_badsql.trc  <inst>_ora_xxxx_SQL_badsql.out
```

This is the plan statistics captured of this query in the tkprof output:

```
……
Line 1179:     162869     162869     162869   NESTED LOOPS OUTER (cr=396510507 pr=41355 pw=6690 time=1912696184 us cost=10384 size=192840 card=1607)
Line 1180:     162869     162869     162869    HASH JOIN RIGHT OUTER (cr=575968 pr=38937 pw=6690 time=68106257 us cost=9967 size=171949 card=1607)
Line 1181:        135        135        135     VIEW  VW_DJW_OUT101HIERLIST_8_0 (cr=16 pr=5 pw=0 time=50107 us cost=3 size=28 card=4)
Line 1182:        135        135        135      NESTED LOOPS  (cr=16 pr=5 pw=0 time=49903 us cost=3 size=476 card=4)
Line 1183:        135        135        135       NESTED LOOPS  (cr=12 pr=5 pw=0 time=48348 us cost=3 size=540 card=5)
Line 1184:        135        135        135        NESTED LOOPS  (cr=8 pr=5 pw=0 time=46650 us cost=3 size=485 card=5)
Line 1185:          5          5          5         NESTED LOOPS  (cr=5 pr=4 pw=0 time=31551 us cost=1 size=216 card=4)
Line 1186:          5          5          5          PARTITION LIST SINGLE PARTITION: KEY KEY (cr=3 pr=3 pw=0 time=21969 us cost=1 size=215 card=5)
Line 1187:          5          5          5           INDEX RANGE SCAN XYZ_SUBCODES_PK PARTITION: 59 59 (cr=3 pr=3 pw=0 time=21953 us cost=1 size=215 card=5)(object id 2493853)
Line 1188:          5          5          5          PARTITION LIST SINGLE PARTITION: KEY KEY (cr=2 pr=1 pw=0 time=9557 us cost=0 size=11 card=1)
Line 1189:          5          5          5           INDEX UNIQUE SCAN XYZ_CODES_PK PARTITION: 59 59 (cr=2 pr=1 pw=0 time=9531 us cost=0 size=11 card=1)(object id 2402829)
Line 1190:        135        135        135         PARTITION LIST SINGLE PARTITION: KEY KEY (cr=3 pr=1 pw=0 time=14958 us cost=1 size=43 card=1)
Line 1191:        135        135        135          TABLE ACCESS BY LOCAL INDEX ROWID XYZ_SUBCODES PARTITION: 59 59 (cr=3 pr=1 pw=0 time=14682 us cost=1 size=43 card=1)
Line 1192:        135        135        135           INDEX RANGE SCAN XYZ_SUBCODES_FK02 PARTITION: 59 59 (cr=2 pr=0 pw=0 time=44 us cost=0 size=0 card=1)(object id 2493907)
Line 1193:        135        135        135        PARTITION LIST SINGLE PARTITION: KEY KEY (cr=4 pr=0 pw=0 time=925 us cost=0 size=11 card=1)
Line 1194:        135        135        135         INDEX UNIQUE SCAN XYZ_CODES_PK PARTITION: 59 59 (cr=4 pr=0 pw=0 time=344 us cost=0 size=11 card=1)(object id 2402829)
Line 1195:        135        135        135       PARTITION LIST SINGLE PARTITION: KEY KEY (cr=4 pr=0 pw=0 time=882 us cost=0 size=11 card=1)
Line 1196:        135        135        135        INDEX UNIQUE SCAN XYZ_CODES_PK PARTITION: 59 59 (cr=4 pr=0 pw=0 time=359 us cost=0 size=11 card=1)(object id 2402829)
……
```

Line: 1179, Top layer: elapsed time=1912696184 us or 32 min.

Line: 1180, 2rd layer: elapsed time=68106257 us or 1 min.

From the tkprof output, we can see the top layer in this run took 32min, while the second layer only took 1 min. This matches what we have found in the earlier analysis.

So now we know where the problem is. What is the fix? Obviously the join method of using nested loop is a wrong choice. We add a hint to force it to use hash join method. Since hint is using the query block alias, we can see the importance of labeling different query block with different alias.

```
SELECT /*+ USE_HASH(e a) */
  e.poiid  AS poiid,
…
FROM
  (SELECT
    (SELECT code
    FROM xyz_codes
    WHERE bus466ssid = 471
    …
    e2.*,
    cat.*,
    OUT.*
  FROM
    (SELECT e3.*,
     …
    FROM
      (SELECT lrowid,
        poiid,
        …
      FROM
        (SELECT *
        FROM
          (SELECT a4.*
          FROM vw_djw_exceptions a4
          …
          )
        )
      GROUP BY lrowid,
        poiid,
        …
      ) e3,
      (SELECT * FROM vw_djw_sopout101s WHERE sopout101code = 253355
      ) h
    WHERE h.sopout101code(+)     = e3.sopout101code
    …
    ) e2,
    vw_djw_categdoghierarchy_8_0 cat,
    vw_djw_out101hierlist_8_0 OUT
  WHERE 1            = 1
  AND e2.categdogcode = cat.code(+)
  AND e2.out101code   = OUT.out101_code(+)
  ) e,
  xyz_sopout101item_attributes a
WHERE e.sopout101code = a.sopout101(+)
AND e.poiid           = a.poiid(+);
```

Now we check the explain plan again:

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                           |     1 |   118 |       | 11132   (2)| 00:02:36 |       |       |
|   1 |  SORT AGGREGATE                          |                           |     1 |   118 |       |            |          |       |       |
|*  2 |   HASH JOIN OUTER                        |                           |  1498 |  172K|       | 11132   (2)| 00:02:36 |       |       |
|*  3 |    HASH JOIN RIGHT OUTER                 |                           |  1498 |  153K|       |  9811   (1)| 00:02:18 |       |       |
|   4 |     VIEW                                 | VW_DJW_OUT101HIERLIST_8_0 |     5 |    35 |       |     6  (17)| 00:00:01 |       |       |
…
```

The estimation Time is 2:36 min., much more promising than the old one with Time as 1:07hr.

Then we run the query with the hint we added. It returns in a half minute. We can compare the 10046 trace file tkprof output here.

- Tkprof - After

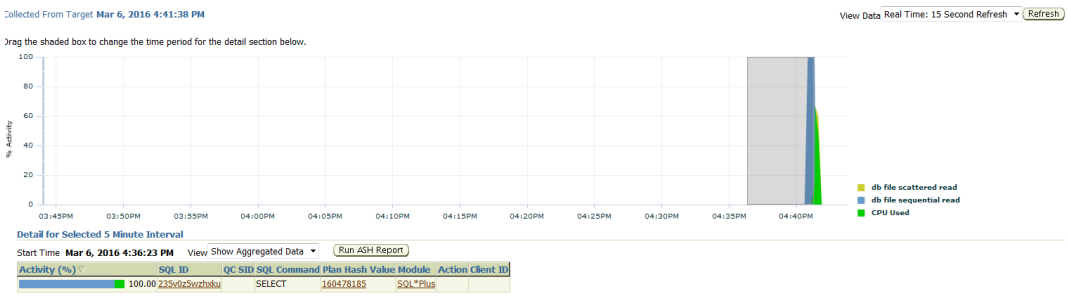
```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.51       0.58          0         48          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        2     16.18      31.17      16905     578400          0           1
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4     16.69      31.76      16905     578448          0           1

Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
         1          1          1  SORT AGGREGATE (cr=578400 pr=16905 pw=6690 time=31172401 us)
    162869     162869     162869   HASH JOIN OUTER (cr=578400 pr=16905 pw=6690 time=31137179 us cost=10817 size=192840 card=1607)
    162869     162869     162869    HASH JOIN RIGHT OUTER (cr=575969 pr=16905 pw=6690 time=31278389 us cost=9967 size=171949 card=1607)
       135        135        135     VIEW  VW_DJW_OUT101HIERLIST_8_0 (cr=16 pr=0 pw=0 time=5063 us cost=3 size=28 card=4)
```

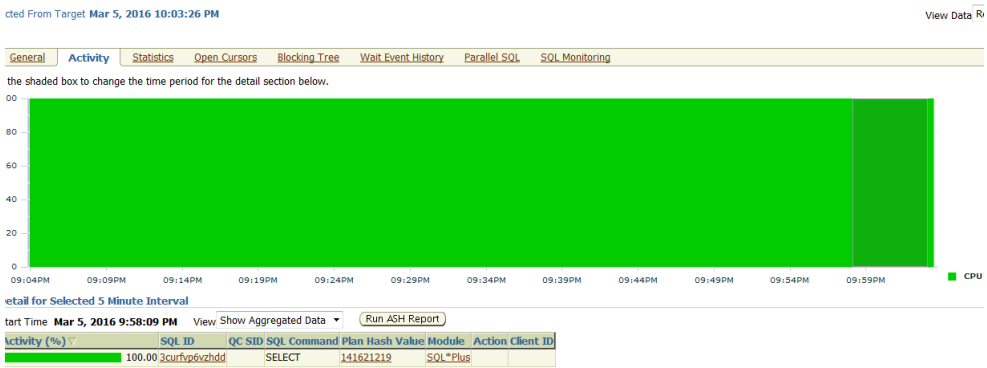
- Tkprof - before

```
Line 1179:     162869     162869     162869   NESTED LOOPS OUTER (cr=396510507 pr=41355 pw=6690 time=1912696184 us cost=10384 size=192840 card=1607)
Line 1180:     162869     162869     162869    HASH JOIN RIGHT OUTER (cr=575968 pr=38937 pw=6690 time=68106257 us cost=9967 size=171949 card=1607)
Line 1181:        135        135        135     VIEW  VW_DJW_OUT101HIERLIST_8_0 (cr=16 pr=5 pw=0 time=50107 us cost=3 size=28 card=4)
```

Here is the new execution activities, some db file sequential read, some db file scattered read and some cpu used. Looks healthy compare to the previous activity, which is nothing but burning cpu!



Before:



SQL Profile and Hint

You may say, this is great, but what if we can't change the query with hint?

We can use dbms_sqltune package to avoid code changes. (need diag+tuning pack licenses)

The idea is to create a sql profile with the hint we added, and attach the sql profile to the query.

This time we use a slight different syntax to retrieve the explain plan output.

```
explain plan set statement_id='f' for
<QUERY hinted>
;
select * from table(dbms_xplan.display(null,null,'ADVANCED'));
```

or

```
<Run the query hinted>
select * from table(dbms_xplan.display _cursor(format => 'ADVANCED'));
```

We should find the outline section from the output report:

```
Outline Data
-------------
 /*+
      BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('11.2.0.3')
      DB_VERSION('11.2.0.4')
      OUTLINE_LEAF(@"SEL$19")
      OUTLINE_LEAF(@"SEL$43F09110")
      …
      USE_HASH_AGGREGATION(@"SEL$43F09110")
      INDEX_RS_ASC(@"SEL$19" "XYZ_RET999ERS"@"SEL$19" ("XYZ_RET999ERS"."RET999ER_ID"))
      END_OUTLINE_DATA
  */
```

Then we use dbms_sqltune package to create a sql profile with the outline.

```
DECLARE
     l_sql               clob;
     BEGIN
     l_sql := q'!select count(*) from (
SELECT e.poiid  AS poiid,
…)
!';
    dbms_sqltune.import_sql_profile( sql_text => l_sql,
                                     name => 'SQLPROFILE_01',
                                     profile => sqlprof_attr(
q'! IGNORE_OPTIM_EMBEDDED_HINTS!',
q'! OPTIMIZER_FEATURES_ENABLE('11.2.0.3')!',
…
q'! USE_HASH_AGGREGATION(@"SEL$43F09110") !',
q'! INDEX_RS_ASC(@"SEL$19" "XYZ_RET999ERS"@"SEL$19" ("XYZ_RET999ERS"."RET999ER_ID"))
),
             force_match => true );
end;
/
```

Let's verify that the sql profile is generated and does work for the original query. As you can see, dbms_sqltune is doing a great job.

```
SQL> select * from table(dbms_xplan.display);
-----------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                     | Name                      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                              |                           |     1 |   118 |       | 11132   (2)| 00:02:36 |       |       |
|   1 |  SORT AGGREGATE                               |                           |     1 |   118 |       |            |          |       |       |
|*  2 |   HASH JOIN OUTER                             |                           |  1498 |   172K|       | 11132   (2)| 00:02:36 |       |       |
|*  3 |    HASH JOIN RIGHT OUTER                      |                           |  1498 |   153K|       |  9811   (1)| 00:02:18 |       |       |
|   4 |     VIEW                                      | VW_DJW_OUT101HIERLIST_8_0 |     5 |    35 |       |     6  (17)| 00:00:01 |       |       |
…
Note
-----
   - SQL profile "SQLPROFILE_01" used for this statement
```

Case Study 2

The Query has 107 lines, and it calls several functions.

Here is the query explain plan:

```
Plan hash value: 1228438649
-----------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                            | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                                 |     9 |  3528 |   100   (1)| 00:00:02 |       |       |
|   1 |  TABLE ACCESS BY INDEX ROWID             | LTE_TYPES                       |     1 |    18 |     1   (0)| 00:00:01 |       |       |
|*  2 |   INDEX UNIQUE SCAN                      | LTE_TYPES_PK                    |     1 |       |     0   (0)| 00:00:01 |       |       |
|   3 |  SORT UNIQUE                             |                                 |     1 |    21 |     3  (34)| 00:00:01 |       |       |
|   4 |   TABLE ACCESS BY INDEX ROWID            | LTE_PPUS_CODE_DEFS              |     1 |    21 |     2   (0)| 00:00:01 |       |       |
|*  5 |    INDEX RANGE SCAN                      | LTE_PPUS_CODE_DEFS_PK           |     1 |       |     1   (0)| 00:00:01 |       |       |
|   6 |  TABLE ACCESS BY INDEX ROWID             | LTE_TYPES                       |     1 |    18 |     1   (0)| 00:00:01 |       |       |
|*  7 |   INDEX UNIQUE SCAN                      | LTE_TYPES_PK                    |     1 |       |     0   (0)| 00:00:01 |       |       |
|   8 |  NESTED LOOPS                            |                                 |     1 |    22 |     3   (0)| 00:00:01 |       |       |
|   9 |   TABLE ACCESS BY INDEX ROWID            | LTE_RPT_PPUSRESSION_LEVELS      |     1 |    13 |     2   (0)| 00:00:01 |       |       |
|* 10 |    INDEX UNIQUE SCAN                     | LTE_RPT_PPUSRESSION_LEVELS_PK   |     1 |       |     1   (0)| 00:00:01 |       |       |
|* 11 |   TABLE ACCESS BY INDEX ROWID            | LTE_PPUS_FIELD_DEFS             |     1 |     9 |     1   (0)| 00:00:01 |       |       |
|* 12 |    INDEX UNIQUE SCAN                     | LTE_PPUS_FIELD_DEFS_PK          |     1 |       |     0   (0)| 00:00:01 |       |       |
|  13 |  TABLE ACCESS BY GLOBAL INDEX ROWID      | XYZ_CODES                       |     1 |    25 |     3   (0)| 00:00:01 | ROWID | ROWID |
|* 14 |   INDEX UNIQUE SCAN                      | XYZ_CODES_UK01                  |     1 |       |     2   (0)| 00:00:01 |       |       |
|  15 |  SORT ORDER BY                           |                                 |     9 |  3528 |   100   (1)| 00:00:02 |       |       |
|* 16 |   HASH JOIN OUTER                        |                                 |     9 |  3528 |    99   (0)| 00:00:02 |       |       |
|  17 |    NESTED LOOPS OUTER                    |                                 |     3 |  1029 |    96   (0)| 00:00:02 |       |       |
|* 18 |     HASH JOIN OUTER                      |                                 |     1 |   289 |    94   (0)| 00:00:02 |       |       |
|  19 |      NESTED LOOPS                        |                                 |     1 |   237 |    82   (0)| 00:00:02 |       |       |
|  20 |       NESTED LOOPS                       |                                 |     8 |   237 |    82   (0)| 00:00:02 |       |       |
|  21 |        NESTED LOOPS                      |                                 |     8 |  1792 |    74   (0)| 00:00:02 |       |       |
|  22 |         NESTED LOOPS                     |                                 |     1 |    82 |     4   (0)| 00:00:01 |       |       |
|  23 |          NESTED LOOPS                    |                                 |     1 |    73 |     3   (0)| 00:00:01 |       |       |
|  24 |           NESTED LOOPS                   |                                 |     1 |    61 |     2   (0)| 00:00:01 |       |       |
|  25 |            TABLE ACCESS BY INDEX ROWID| LTE_PPUS_FIELD_DEFS             |     1 |    25 |     1   (0)| 00:00:01 |       |       |
|* 26 |             INDEX UNIQUE SCAN            | LTE_PPUS_FIELD_DEFS_PK          |     1 |       |     0   (0)| 00:00:01 |       |       |
|  27 |            TABLE ACCESS BY INDEX ROWID| LTE_RPT_PPUSRESSION_RULES       |     1 |    36 |     1   (0)| 00:00:01 |       |       |
|* 28 |             INDEX UNIQUE SCAN            | LTE_RPT_PPUSRESSION_RULES_UK    |     1 |       |     0   (0)| 00:00:01 |       |       |
|* 29 |           INDEX RANGE SCAN               | LTE_RPT_RULESET_VERSIONS_PK     |     1 |    12 |     1   (0)| 00:00:01 |       |       |
|* 30 |          TABLE ACCESS BY INDEX ROWID     | LTE_RULESETS                    |     1 |     9 |     1   (0)| 00:00:01 |       |       |
|* 31 |           INDEX UNIQUE SCAN              | LTE_RULESETS_PK                 |     1 |       |     0   (0)| 00:00:01 |       |       |
|* 32 |         TABLE ACCESS FULL                | LTE_RPT_PPUS_PSTEDT_RULES       |     8 |  1136 |    70   (0)| 00:00:01 |       |       |
|* 33 |        INDEX UNIQUE SCAN                 | LTE_RPT_PPUSRESSION_LEVELS_PK   |     1 |       |     0   (0)| 00:00:01 |       |       |
|* 34 |       TABLE ACCESS BY INDEX ROWID        | LTE_RPT_PPUSRESSION_LEVELS      |     1 |    13 |     1   (0)| 00:00:01 |       |       |
|  35 |      TABLE ACCESS FULL                   | LTE_RPT_PPUS_PSTEDT_CASES       |  5781 |   293K|    12   (0)| 00:00:01 |       |       |
|  36 |     TABLE ACCESS BY INDEX ROWID          | LTE_RPT_PPUS_PSTEDT_FLT_VALUES  |     3 |   162 |     2   (0)| 00:00:01 |       |       |
|* 37 |      INDEX RANGE SCAN                    | PPUS_PSTEDT_FLT_VALS_UNIQUE     |     3 |       |     1   (0)| 00:00:01 |       |       |
|* 38 |    TABLE ACCESS BY INDEX ROWID           | LTE_RPT_PPUS_PSTEDT_FIELD_DEFS  |     3 |   147 |     3   (0)| 00:00:01 |       |       |
|* 39 |     INDEX RANGE SCAN                     | PPUS_PSTEDT_FIELD_DEFS_PK       |     3 |       |     2   (0)| 00:00:01 |       |       |
-----------------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
```

…

Here is 10046 trace:

```
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.02       0.02          0          3          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       36      0.17       0.17          0       1529      11404          35
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       38      0.20       0.20          0       1532      11404          35

Rows (1st) Rows (avg) Rows (max)  Row Source Operation
---------- ---------- ----------  ---------------------------------------------------
         1          1          1  TABLE ACCESS BY INDEX ROWID LTE_TYPES (cr=2 pr=0 pw=0 time=26 us cost=1 size=18 card=1)
         1          1          1   INDEX UNIQUE SCAN LTE_TYPES_PK (cr=1 pr=0 pw=0 time=14 us cost=0 size=0 card=1)(object id 14779233)
         0          0          0  SORT UNIQUE (cr=0 pr=0 pw=0 time=15 us cost=3 size=21 card=1)
         0          0          0   TABLE ACCESS BY INDEX ROWID LTE_PPUS_CODE_DEFS (cr=0 pr=0 pw=0 time=6 us cost=2 size=21 card=1)
         0          0          0    INDEX RANGE SCAN LTE_PPUS_CODE_DEFS_PK (cr=0 pr=0 pw=0 time=2 us cost=1 size=0 card=1)(object id 15000131)
         1          1          1  TABLE ACCESS BY INDEX ROWID LTE_TYPES (cr=2 pr=0 pw=0 time=10 us cost=1 size=18 card=1)
         1          1          1   INDEX UNIQUE SCAN LTE_TYPES_PK (cr=1 pr=0 pw=0 time=4 us cost=0 size=0 card=1)(object id 14779233)
         2          2          2  NESTED LOOPS  (cr=9 pr=0 pw=0 time=66 us cost=3 size=22 card=1)
         2          2          2   TABLE ACCESS BY INDEX ROWID LTE_RPT_PPUSRESSION_LEVELS (cr=5 pr=0 pw=0 time=31 us cost=2 size=13 card=1)
         2          2          2    INDEX UNIQUE SCAN LTE_RPT_PPUSRESSION_LEVELS_PK (cr=3 pr=0 pw=0 time=17 us cost=1 size=0 card=1)(object id 15261028)
         2          2          2   TABLE ACCESS BY INDEX ROWID LTE_PPUS_FIELD_DEFS (cr=4 pr=0 pw=0 time=20 us cost=1 size=9 card=1)
         2          2          2    INDEX UNIQUE SCAN LTE_PPUS_FIELD_DEFS_PK (cr=2 pr=0 pw=0 time=7 us cost=0 size=0 card=1)(object id 14779217)
         3          3          3  TABLE ACCESS BY GLOBAL INDEX ROWID XYZ_CODES PARTITION: ROW LOCATION ROW LOCATION (cr=11 pr=0 pw=0 time=90 us cost=3 size=25 card=1)
         3          3          3   INDEX UNIQUE SCAN XYZ_CODES_UK01 (cr=8 pr=0 pw=0 time=55 us cost=2 size=0 card=1)(object id 15278521)
        35         35         35  SORT ORDER BY (cr=1277054 pr=3 pw=0 time=143806498 us cost=98 size=3096 card=8)
        35         35         35   HASH JOIN OUTER (cr=268 pr=0 pw=0 time=7117 us cost=97 size=3096 card=8)
         7          7          7    NESTED LOOPS OUTER (cr=265 pr=0 pw=0 time=4533 us cost=94 size=1020 card=3)
         5          5          5     HASH JOIN OUTER (cr=254 pr=0 pw=0 time=6370 us cost=92 size=288 card=1)
         5          5          5      NESTED LOOPS  (cr=228 pr=0 pw=0 time=1379 us cost=82 size=236 card=1)
        23         23         23       NESTED LOOPS  (cr=205 pr=0 pw=0 time=1728 us cost=82 size=236 card=8)
        23         23         23        NESTED LOOPS  (cr=196 pr=0 pw=0 time=803 us cost=74 size=1784 card=8)
         1          1          1         NESTED LOOPS  (cr=9 pr=0 pw=0 time=165 us cost=4 size=81 card=1)
         1          1          1          NESTED LOOPS  (cr=6 pr=0 pw=0 time=114 us cost=3 size=72 card=1)
         1          1          1           NESTED LOOPS  (cr=5 pr=0 pw=0 time=87 us cost=2 size=61 card=1)
         1          1          1            TABLE ACCESS BY INDEX ROWID LTE_PPUS_FIELD_DEFS (cr=2 pr=0 pw=0 time=41 us cost=1 size=25 card=1)
         1          1          1             INDEX UNIQUE SCAN LTE_PPUS_FIELD_DEFS_PK (cr=1 pr=0 pw=0 time=19 us cost=0 size=0 card=1)(object id 14779217)
         1          1          1            TABLE ACCESS BY INDEX ROWID LTE_RPT_PPUSRESSION_RULES (cr=3 pr=0 pw=0 time=40 us cost=1 size=36 card=1)
         1          1          1             INDEX UNIQUE SCAN LTE_RPT_PPUSRESSION_RULES_UK (cr=2 pr=0 pw=0 time=28 us cost=0 size=0 card=1)(object id 15546053)
         1          1          1          INDEX RANGE SCAN LTE_RPT_RULESET_VERSIONS_PK (cr=1 pr=0 pw=0 time=20 us cost=1 size=11 card=1)(object id 15260996)
         1          1          1          TABLE ACCESS BY INDEX ROWID LTE_RULESETS (cr=3 pr=0 pw=0 time=43 us cost=1 size=9 card=1)
         1          1          1          INDEX UNIQUE SCAN LTE_RULESETS_PK (cr=2 pr=0 pw=0 time=27 us cost=0 size=0 card=1)(object id 14779176)
        23         23         23         TABLE ACCESS FULL LTE_RPT_PPUS_PSTEDT_RULES (cr=187 pr=0 pw=0 time=627 us cost=70 size=1136 card=8)
        23         23         23        INDEX UNIQUE SCAN LTE_RPT_PPUSRESSION_LEVELS_PK (cr=9 pr=0 pw=0 time=576 us cost=0 size=0 card=1)(object id 15261028)
         5          5          5       TABLE ACCESS BY INDEX ROWID LTE_RPT_PPUSRESSION_LEVELS (cr=23 pr=0 pw=0 time=62 us cost=1 size=13 card=1)
      5266       5266       5266      TABLE ACCESS FULL LTE_RPT_PPUS_PSTEDT_CASES (cr=26 pr=0 pw=0 time=5546 us cost=10 size=266968 card=5134)
         6          6          6     TABLE ACCESS BY INDEX ROWID LTE_RPT_PPUS_PSTEDT_FLT_VALUES (cr=11 pr=0 pw=0 time=1459 us cost=2 size=156 card=3)
         6          6          6      INDEX RANGE SCAN PPUS_PSTEDT_FLT_VALS_UNIQUE (cr=7 pr=0 pw=0 time=403 us cost=1 size=0 card=3)(object id 17373910)
         5          5          5    TABLE ACCESS BY INDEX ROWID LTE_RPT_PPUS_PSTEDT_FIELD_DEFS (cr=3 pr=0 pw=0 time=872 us cost=3 size=141 card=3)
         5          5          5     INDEX RANGE SCAN PPUS_PSTEDT_FIELD_DEFS_PK (cr=2 pr=0 pw=0 time=371 us cost=2 size=0 card=3)(object id 17373906)
```

The Query plan doesn't show any problem. The 10046 trace says it returns in 0.2 sec. Where is the time spent? We need to look at all parts of the 10046 trace this time.

At the bottom of the tkprof report there is a section with the overall total for non-recursive statements and recursive statements executed by the query:

```
OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.02       0.02          0          3          0           0
Execute      1      0.00       0.00          0          0          0           0
Fetch       36      0.17       0.17          0       1529      11404          35
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total       38      0.20       0.20          0       1532      11404          35

OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse      272      0.03       0.03          0          0          8           0
Execute   4147      0.36       0.37          0          1          5           1
Fetch     4313    143.19     143.23          3    1275525          0        3655
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     8732    143.59     143.63          3    1275526         13        3656
```

The top part is from our query. It executed once, fetched 35 rows, spent 0.2 sec. However let's look at the bottom part. This part sums up all the recursive calls by the query. The total time is about 2 min. What is this?

Let's look through the total time spend of all the calls in 10046 trace file.

```
$ grep -n total DXYZ1_ora_16499_SQL_badsql.out

47:total       38      0.20       0.20          0       1532      11404          35
122:total      315      0.00       0.00          0        210          0          90
153:total      181      0.00       0.00          0        630          0          90
192:total      181      0.01       0.01          0        900          2         120
234:total      241      0.01       0.01          0        240          0         120
265:total      341      0.02       0.02          0       1870          0         170
308:total      241      0.00       0.00          0        240          0         120
337:total      451      0.01       0.01          0        630          0         150
365:total      301      0.00       0.01          0        390          0          90
400:total      141      0.03       0.02          0       2160          0          50
461:total       71      0.28       0.30          0       8785          0          35
505:total     1681      0.04       0.05          0       2520          0         840
539:total     1681      0.05       0.04          0       2520          0         840
568:total     1681      0.05       0.05          0       2730          0         350
627:total      701    142.97     143.00          0    1250690          0           0
722:total      105      0.00       0.00          0         70          0          35
752:total      105      0.00       0.00          0        385          0          35
785:total       21      0.00       0.00          0         14          0           7
815:total       21      0.00       0.00          0         77          0           7
848:total       21      0.00       0.00          0         14          0           7
884:total       21      0.00       0.00          0         77          0           7
923:total       21      0.00       0.00          0         14          0           7
953:total       21      0.00       0.00          0         77          0           7
986:total       21      0.00       0.00          0         14          0           7
1016:total       21      0.00       0.00          0         77          0           7
1049:total       21      0.00       0.00          0         14          0           7
1079:total       21      0.00       0.00          0         77          0           7
1113:total       38      0.20       0.20          0       1532      11404          35
1137:total     8732    143.59     143.63          3    1275526         13        3656
```

We found the top time consuming call. Line 627 in 10046 trace file shows its total time spent is almost the same as the total recursive calls in line 1137. From line 588 to line 627 in 10046 trace file is this call's detail.

```
588 SELECT DISTINCT 1
589 FROM
590  (SELECT TO_CHAR (OCC.CODE) AS CODE, OCC.NAME FROM
591   ETL_MGR.ETL_SUPP_FIELD_DEFS FD, ETL_MGR.ETL_BUS_DATASRC_DATAELEMENTS DD,
592   ETL_MGR.ETL_CODESET_MAPS CM, (SELECT SCD.CODE, SCD.CODESET, C.NAME FROM
593   ETL_MGR.ETL_SUPP_CODE_DEFS SCD, ODS.ODS_CODES C WHERE SCD.CODE = C.CODE AND
594   ( C.NO_DELETE_FLAG = 1 AND LOWER (C.NAME) LIKE '%other%') UNION SELECT
595   SCD.CODE, SCD.CODESET, CI.PARENT_NAME||'#'||CI.MODELNUMBER FROM
596   ETL_MGR.ETL_SUPP_CODE_DEFS SCD, ODS.ODS_CODE_ITEMNUMBERS CI WHERE SCD.CODE =
597    CI.CODE AND ( CI.NO_DELETE_FLAG = 1 AND LOWER (CI.MODELNUMBER) LIKE
598   '%other%' AND LOWER(CI.PARENT_NAME) LIKE '%other%')) OCC WHERE FD.DIM_LEVEL
599   IN (SELECT DIMENSIONVIEW_LEVEL_ID FROM ODS.ODS_DIMENSIONVIEW_LEVELS WHERE
600   DIMENSIONVIEW = :B1 ) AND FD.INPUT_DATA_ELEMENT_ID = DD.BUSDS_ELEMENT_ID
601   AND DD.CODESET_MAP_ID = CM.CODESET_MAP_ID AND CM.CODESET = OCC.CODESET
602   UNION SELECT TO_CHAR (OCC.CODE) AS CODE, OCC.NAME FROM
603   ETL_MGR.ETL_SUPP_FIELD_DEFS FD, ETL_MGR.ETL_BUS_DATASRC_DATAELEMENTS DD,
604   ETL_MGR.ETL_CODESET_MAPS CM, ODS.ODS_CODESETS CC, ODS.ODS_DERIVEDCODESETS
605   DC, (SELECT CV.CODE, C.CODESET, C.NAME FROM ODS.ODS_CODES C,
606   ODS.ODS_CODEVIEWS CV, ETL_MGR.ETL_SUPP_CODE_DEFS SCD WHERE C.CODE = CV.CODE
607   AND CV.CODE = SCD.CODE AND C.NO_DELETE_FLAG = 1 AND LOWER (C.NAME) LIKE
608   '%other%' UNION SELECT CVI.CODE, CI.CODESET,
609   CI.PARENT_NAME||'#'||CI.MODELNUMBER AS NAME FROM ODS.ODS_CODE_ITEMNUMBERS
610   CI, ODS.ODS_CODEVIEW_ITEMNUMBERS CVI, ETL_MGR.ETL_SUPP_CODE_DEFS SCD WHERE
611   CI.CODE = CVI.CODE AND CVI.CODE = SCD.CODE AND CI.NO_DELETE_FLAG = 1 AND
612   LOWER (CI.MODELNUMBER) LIKE '%other%' AND LOWER(CI.PARENT_NAME) LIKE
613   '%other%') OCC WHERE FD.DIM_LEVEL IN (SELECT DIMENSIONVIEW_LEVEL_ID FROM
614   ODS.ODS_DIMENSIONVIEW_LEVELS WHERE DIMENSIONVIEW = :B1 ) AND
615   FD.INPUT_DATA_ELEMENT_ID = DD.BUSDS_ELEMENT_ID AND DD.CODESET_MAP_ID =
616   CM.CODESET_MAP_ID AND CM.CODESET = CC.CODESET AND CC.DERIVEDCODESET = 1 AND
617   CC.CODESET = DC.DERIVEDCODESET AND DC.CODESET = OCC.CODESET ) WHERE CODE =
618   :B2
619
620
621 call     count       cpu    elapsed       disk      query    current        rows
622 ------- ------  -------- ---------- ---------- ---------- ----------  ----------
623 Parse        1      0.00       0.00          0          0          0           0
624 Execute    350      0.16       0.16          0          0          0           0
625 Fetch      350    142.80     142.83          0    1250690          0           0
626 ------- ------  -------- ---------- ---------- ---------- ----------  ----------
627 total      701    142.97     143.00          0    1250690          0           0
```

From this we can see that this call was executed 350 times, although consumed 143 seconds, but no rows returns. This is found later from one of the function call embedded in the query, and is determined the call is unnecessary. This function call was removed from the query, problem is resolved.

Summary:

- Take advantage of the available diagnostic and tuning tools
- Don’t forget the fundamental tool kit:
  - Top
  - Explain plan
  - 10046 trace and tkprof
  - Optimizer hint and SQL profile
- Prerequisite: Statistics is up to date