Oracle Diagnostics

Hemant K Chitale

Hemant K Chitale

- whoami ?
- Oracle 5 to Oracle 10gR2 ： DOS, Xenix,8 flavours of Unix, Linux, Windows
- Financial Services, Govt/Not-for-Profit, ERP, Custom
- Production Support, Consulting, Development
- A DBA, not a Developer
- *My Oracle Blog* *http://hemantoracledba.blogspot.com*

Explain Plans -- simple

- Explain Plan is a method of displaying the *expected* OR *actual* SQL Execution Plan
- Since 9i, Oracle has provided the DBMS_XPLAN package with various procedures

Method 1 : Without Executing the Query

- NOT to be used if the query has Binds – particularly because "Explain is blind to Binds"
- Use "EXPLAIN PLAN FOR …. Query …." followed by "DBMS_XPLAN.DISPLAY".

Given this query :

```
select sale_id, cust_id, remarks
from sales where
sale_date between to_date('01-NOV-10','DD-MON-RR')
              and to_date('04-NOV-10','DD-MON-RR')
```

```
SQL> explain plan for
  2  select sale_id, cust_id, remarks
  3  from sales where
  4  sale_date between to_date('01-NOV-10','DD-MON-RR')
  5                and to_date('04-NOV-10','DD-MON-RR')
  6  /

Explained.

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
----------------------------------------------
Plan hash value: 1231079358
-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |  3000 |   166K|   244   (2)| 00:00:03 |
|*  1 |  FILTER            |       |       |       |            |          |
|*  2 |   TABLE ACCESS FULL| SALES |  3000 |   166K|   244   (2)| 00:00:03 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(TO_DATE('01-NOV-10','DD-MON-RR')<=TO_DATE('04-NOV-10','DD-
              MON-RR'))
   2 - filter("SALE_DATE"<=TO_DATE('04-NOV-10','DD-MON-RR') AND
              "SALE_DATE">=TO_DATE('01-NOV-10','DD-MON-RR'))

17 rows selected.
```

Understand the components :

```
Plan hash value: 1231079358
-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |  3000 |   166K|   244   (2)| 00:00:03 |
|*  1 |  FILTER            |       |       |       |            |          |
|*  2 |   TABLE ACCESS FULL| SALES |  3000 |   166K|   244   (2)| 00:00:03 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter(TO_DATE('01-NOV-10','DD-MON-RR')<=TO_DATE('04-NOV-10','DD-
              MON-RR'))
   2 - filter("SALE_DATE"<=TO_DATE('04-NOV-10','DD-MON-RR') AND
              "SALE_DATE">=TO_DATE('01-NOV-10','DD-MON-RR'))
```

Every Execution Plan has a Hash Value. (Just as every SQL has a Hash Value and SQL_ID). We'll see later where the Hash Value is an important clue.

Step 2 is indented --- we normally think that it is executed before Step 1.
A Filter for SALE_DATE between two dates is applied when doing a FullTableScan ("TABLE ACCESS FULL" and "filter") at Step 2. Oracle expects to return 3000 rows after applying the "filter" to the FullTableScan. (The Explain Plan shows the number of rows expected to be returned by the step, not the number of rows that the FTS will read {which is actually 100,000 !}). These 3000 rows will be 166KBytes to be returnd to the client (SQLPlus session).
Step 1 is a filter that Oracle applies for validation.

What is the filter in Step 1 ? Why does Oracle “apply it for validation” ?

```
SQL> l
  1  explain plan for
  2  select sale_id, cust_id, remarks
  3  from sales where
  4  sale_date between to_date('01-NOV-10','DD-MON-RR')
  5*               and to_date('25-OCT-10','DD-MON-RR')
SQL> /

Explained.

SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
----------------------------------------------
Plan hash value: 1231079358
-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |     1 |    57 |   244   (2)| 00:00:03 |
|*  1 |  FILTER            |       |       |       |            |          |
|*  2 |   TABLE ACCESS FULL| SALES |     1 |    57 |   244   (2)| 00:00:03 |
-------------------------------------------------------------------------------

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

   1 - filter(TO_DATE('01-NOV-10','DD-MON-RR')<=TO_DATE('25-OCT-10','DD-
              MON-RR'))
   2 - filter("SALE_DATE"<=TO_DATE('25-OCT-10','DD-MON-RR') AND
              "SALE_DATE">=TO_DATE('01-NOV-10','DD-MON-RR'))

SQL>
```

```
SQL> set autotrace on
SQL> select sale_id, cust_id, remarks
  2  from sales where
  3  sale_date between to_date('01-NOV-10','DD-MON-RR')
  4                and to_date('25-OCT-10','DD-MON-RR')
  5  /

no rows selected

Execution Plan
----------------------------------------------------------
Plan hash value: 1231079358
--------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |     1 |    57 |   244   (2)| 00:00:03 |
|*  1 |  FILTER            |       |       |       |            |          |
|*  2 |   TABLE ACCESS FULL| SALES |     1 |    57 |   244   (2)| 00:00:03 |
--------------------------------------------------------------------------------

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

   1 - filter(TO_DATE('01-NOV-10','DD-MON-RR')<=TO_DATE('25-OCT-10','DD-
              MON-RR'))
   2 - filter("SALE_DATE"<=TO_DATE('25-OCT-10','DD-MON-RR') AND
              "SALE_DATE">=TO_DATE('01-NOV-10','DD-MON-RR'))
Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
          0  consistent gets
          0  physical reads
          0  redo size
        409  bytes sent via SQL*Net to client
        408  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client

SQL>
```

You can now see that the FILTER in step 1 was a method for validation. Since

```
filter(TO_DATE('01-NOV-10','DD-MON-RR')<=TO_DATE('25-OCT-10','DD-MON-RR'))
```

will be *FALSE* (because '01-NOV-10' is *not less than* '25-OCT-10'), the next step (Step 2) does not get executed at all.

That is why the AUTOTRACE shows 0 block gets

```
Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
          0  consistent gets
          0  physical reads
```

inspite of Step 2 supposedly being "TABLE ACCESS FULL" !

Therefore, when you see a FILTER as a step in an Execution Plan, remember that it may be a "logic filter". Oracle may be using it to decide if the "indented" step below it needs to be executed at all.
If this FILTER returns FALSE, the "indented" "child" step is NOT executed.

Method 2 : Without Query Execution

- This is useful to “validate” an Execution Plan
- This is an alternative to executing with SQL Tracing enabled and writing to a trace file
- This method uses the Hint “GATHER_PLAN_STATISTICS”. (Another option is to set the session parameter STATISTICS_LEVEL to “ALL” just before execution of the SQL).

Actually executing the query, with the Hint added :

```
SQL> select /*+ gather_plan_statistics */ sale_id, cust_id, remarks
  2  from sales where
  3  sale_date between to_date('01-NOV-10','DD-MON-RR')
  4                and to_date('04-NOV-10','DD-MON-RR')
  5  /

   SALE_ID    CUST_ID REMARKS
---------- ---------- ----------------------------------------
---
     24099         40 BIK9SBLPJKGKA8UINWX20O64KAD210CMW4Z7AZUL
     24100         44 TXE1BTQM1631FJVTYCUBYBGOFRL7O32QVG20ZV6C
…….
…….

 27097         77 11KBENL0XE4T2OXAWXF9DAW2HHSQBG696BHJ5F79
     27098         17 MHEK21ILW79EHUIJC1Q3RA4PLC844K2KXNXCYL3E

3000 rows selected.

Elapsed: 00:00:04.20
SQL>
```

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

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------
------------------------------------
SQL_ID  2xk5b0ypks53n, child number 0
------------------------------------
select /*+ gather_plan_statistics */ sale_id, cust_id, remarks from
sales where sale_date between to_date('01-NOV-10','DD-MON-RR')
     and to_date('04-NOV-10','DD-MON-RR')

Plan hash value: 1231079358

------------------------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |
------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |   3000 |00:00:00.24 |    1064 |    835 |
|*  1 |  FILTER            |       |      1 |        |   3000 |00:00:00.24 |    1064 |    835 |
|*  2 |   TABLE ACCESS FULL| SALES |      1 |   3000 |   3000 |00:00:00.15 |    1064 |    835 |
------------------------------------------------------------------------------------------------------

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

   1 - filter(TO_DATE('01-NOV-10','DD-MON-RR')<=TO_DATE('04-NOV-10','DD-MON-RR'))
   2 - filter(("SALE_DATE"<=TO_DATE('04-NOV-10','DD-MON-RR') AND
              "SALE_DATE">=TO_DATE('01-NOV-10','DD-MON-RR')))
```

I use the DISPLAY_CURSOR procedure.
The first two Parameters are SQL_ID and CHILD_NO. The ‘’ (NULL) values passed tell Oracle to evaluate for the SQL just (previously) executed in my *current* session.
The last parameter is for display format. ‘ALLSTATS LAST’ is to show all statistics for the Last execution of the same SQL.

```
-------------------------------------------------------------------------------------------------
| Id  | Operation          | Name  | Starts | E-Rows | A-Rows |   A-Time   | Buffers | Reads  |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |      1 |        |   3000 |00:00:00.24 |    1064 |    835 |
|*  1 |  FILTER            |       |      1 |        |   3000 |00:00:00.24 |    1064 |    835 |
|*  2 |   TABLE ACCESS FULL| SALES |      1 |   3000 |   3000 |00:00:00.15 |    1064 |    835 |
-------------------------------------------------------------------------------------------------

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

   1 - filter(TO_DATE('01-NOV-10','DD-MON-RR')<=TO_DATE('04-NOV-10','DD-MON-RR'))
   2 - filter(("SALE_DATE"<=TO_DATE('04-NOV-10','DD-MON-RR') AND
              "SALE_DATE">=TO_DATE('01-NOV-10','DD-MON-RR')))
```

Here Oracle shows the Expected-Rows from each Step versus the Actual-Rows. (We already know the Step 1 for this plan is a “validation” step so we ignore it). Step 2 was expected to return 3000 rows and did return 3000 rows (after applying the filters on all the 100,000 rows returned by the FullTableScan).
The Actual-Time and Buffer Gets and Physical Reads (number of Oracle Blocks, not number of Read Calls to the OS) are also displayed.

In a complex Execution Plan, the A-Rows, A-Time and Buffers (and, possibly, Reads) columns are the ones to pay attention to – they will indicate whether the expected cardinality for the particular step was correct (if A-Rows is different from E-Rows) and how long the step took.

NOTE : When you have a query that is a join of two tables using a Nested Loop, you might mis-read the E-Rows and A-Rows. You will have to pay attention to another column “Starts”.

Method 3 : From AWR Historical Information

- This is useful to monitor a query's execution profile over time
- It can be done only if the SQL has been captured by AWR
- Remember : AWR only captures the Top 'N' SQLs present in the Shared Pool when a Snapshot is created. If the SQL had been invalidated or aged out or was not in the Top 'N', at the time of the Snapshot, it wouldn't have bee captured by AWR !

NOTE : This output spans this slide *and* the next two slides :

SQL> select * from table(dbms_xplan.display_awr('3pat1z2qx4gyg')) ;

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
----------------
```

```
SQL_ID 3pat1z2qx4gyg
--------------------
select sale_id, cust_id, remarks from sales where sale_date between
to_date('01-NOV-10','DD-MON-RR')              and
to_date('04-NOV-10','DD-MON-RR')


Plan hash value: 909439854

-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                 |       |       |    35 (100)|          |
|   1 |  FILTER                      |                 |       |       |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| SALES           |  3000 |   166K|    35   (0)| 00:00:01 |
|   3 |    INDEX RANGE SCAN          | SALES_DATES_NDX |  3000 |       |     9   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

```
SQL_ID 3pat1z2qx4gyg
--------------------
select sale_id, cust_id, remarks from sales where sale_date between
to_date('01-NOV-10','DD-MON-RR')               and
to_date('04-NOV-10','DD-MON-RR')

Plan hash value: 1231079358

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |       |       |   244 (100)|          |
|   1 |  FILTER            |       |       |       |            |          |
|   2 |   TABLE ACCESS FULL| SALES |  3000 |   166K|   244   (2)| 00:00:03 |
-------------------------------------------------------------------------------
```

We see two different Execution Plans with two different Plan Hash Values.

Why is that so ?

Apparently, the Execution Plan earlier had used an Index Range Scan, using Index “SALES_DATES_NDX”.

Later the Execution Plan for the same SQL (because the SQL_ID is the same) has changed to a Full Table Scan.

What had happened was that I had dropped the Index “SALES_DATES_NDX” between the first execution and the second one !

So, this display from AWR shows that the Execution Plans can and have changed over time, with the Plan Hash Value changing accordingly !

Another way is to run $ORACLE_HOME/rdbms/admin/awrsqrpt.sql :

```
Specify the Begin and End Snapshot Ids
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Enter value for begin_snap: 1152
Begin Snapshot Id specified: 1152

Enter value for end_snap: 1156
End   Snapshot Id specified: 1156



Specify the SQL Id
~~~~~~~~~~~~~~~~~~
Enter value for sql_id: 3pat1z2qx4gyg
SQL ID specified:  3pat1z2qx4gyg

Specify the Report Name
~~~~~~~~~~~~~~~~~~~~~~~
The default report file name is awrsqlrpt_1_1152_1156.txt.  To use this name,
press <return> to continue, otherwise enter an alternative.

Enter value for report_name:

Using the report name awrsqlrpt_1_1152_1156.txt
```

WORKLOAD REPOSITORY SQL Report

Snapshot Period Summary

DB Name	DB Id	Instance	Inst Num	Startup Time	Release	RAC
ORCL	1229390655	orcl	1	21-Apr-11 22:33	11.2.0.1.0	NO

	Snap Id	Snap Time	Sessions	Curs/Sess
Begin Snap:	1152	21-Apr-11 22:40:02	29	1.4
End Snap:	1156	22-Apr-11 00:00:19	31	1.5
Elapsed:		80.29 (mins)		
DB Time:		1.76 (mins)		

SQL Summary DB/Inst: ORCL/orcl Snaps: 1152-1156

SQL Id	Elapsed Time (ms)
3pat1z2qx4gyg	518

Module: sqlplus@localhost.localdomain (TNS V1-V3)

```
select sale_id, cust_id, remarks from sales where sale_date between to_date('01-
NOV-10','DD-MON-RR') and to_date('04-NOV-10','DD-MON-RR')
```

```
SQL ID: 3pat1z2qx4gyg                    DB/Inst: ORCL/orcl  Snaps: 1152-1156
-> 1st Capture and Last Capture Snap IDs
   refer to Snapshot IDs witin the snapshot range
-> select sale_id, cust_id, remarks from sales where sale_date between to...

    Plan Hash           Total Elapsed                 1st Capture   Last Capture
#   Value                    Time(ms)    Executions       Snap ID        Snap ID
--- ---------------- ---------------- ------------ ------------ --------------
1   1231079358                  400             1         1155           1155
2   909439854                   118             1         1153           1153
          ------------------------------------------------------------
```

This shows that there have been two executions, one of 400ms and the other of 118ms, with two different Plan Hash Values and captured in different Snaphots.
Plan 2 is the “older” plan and took less time. This was the Plan with the Index Range Scan.

Plan 1(PHV: 1231079358)

Plan Statistics DB/Inst: ORCL/orcl Snaps: 1152-1156
-> % Total DB Time is the Elapsed Time of the SQL statement divided
into the Total Database Time multiplied by 100

Stat Name	Statement	Per Execution	% Snap
Elapsed Time (ms)	400	399.5	0.4
CPU Time (ms)	248	248.0	0.3
Executions	1	N/A	N/A
Buffer Gets	1,281	1,281.0	0.2
Disk Reads	883	883.0	4.1
Parse Calls	1	1.0	0.0
Rows	3,000	3,000.0	N/A
User I/O Wait Time (ms)	42	N/A	N/A
Cluster Wait Time (ms)	0	N/A	N/A
Application Wait Time (ms)	0	N/A	N/A
Concurrency Wait Time (ms)	0	N/A	N/A
Invalidations	0	N/A	N/A
Version Count	1	N/A	N/A
Sharable Mem(KB)	14	N/A	N/A

Execution Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				244 (100)	
1	FILTER					
2	TABLE ACCESS FULL	SALES	3000	166K	244 (2)	00:00:03

Plan 2(PHV: 909439854)

Plan Statistics   DB/Inst: ORCL/orcl Snaps: 1152-1156
-> % Total DB Time is the Elapsed Time of the SQL statement divided
   into the Total Database Time multiplied by 100

Stat Name	Statement	Per Execution	% Snap
Elapsed Time (ms)	118	118.0	0.1
CPU Time (ms)	45	45.0	0.1
Executions	1	N/A	N/A
Buffer Gets	666	666.0	0.1
Disk Reads	59	59.0	0.3
Parse Calls	1	1.0	0.0
Rows	3,000	3,000.0	N/A
User I/O Wait Time (ms)	84	N/A	N/A
Cluster Wait Time (ms)	0	N/A	N/A
Application Wait Time (ms)	0	N/A	N/A
Concurrency Wait Time (ms)	0	N/A	N/A
Invalidations	0	N/A	N/A
Version Count	1	N/A	N/A
Sharable Mem(KB)	14	N/A	N/A

Execution Plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				35 (100)	
1	FILTER					
2	TABLE ACCESS BY INDEX ROWID	SALES	3000	166K	35 (0)	00:00:01
3	INDEX RANGE SCAN	SALES_DATES_NDX	3000		9 (0)	00:00:01

```
Full SQL Text

SQL ID       SQL Text
------------ ----------------------------------------------------------------
3pat1z2qx4gy select sale_id, cust_id, remarks from sales where sale_date betwe
             een to_date('01-NOV-10', 'DD-MON-RR') and to_date('04-NOV-10', 'D
             D-MON-RR')

Report written to awrsqlrpt_1_1152_1156.txt
SQL>
```