**Pinning an old execution plan**

We have number of ways to pin an old plan, but in this documents I have used **SQLT** (it's a DB Level installation, we can use it from any instance, if the database is rac – but extracted **SQLT scripts should be used for interpreting sql, execution plan, sql profile creation**

Yellow marked ones are required to give inputs.

**Interpreting SQL ID Details:**

Select distinct SQL\_ID,PLAN\_HASH\_VALUE,TIMESTAMP from dba\_hist\_sql\_plan where SQL\_ID='0zwrsg1nh990q' order by TIMESTAMP;

**More details**

Select id, operation, options, object\_name, cost from dba\_hist\_sql\_plan where sql\_id = '8j44jcmd9fjmv' and plan\_hash\_value = 3827406737;

**Detailed query**

This will help for finding the last successful Plan hash value

=================================================================================================

SET PAUSE ON

SET PAGESIZE 60

SET LINESIZE 300

SELECT DISTINCT sql\_id, plan\_hash\_value, OPTIMIZER\_ENV\_HASH\_VALUE, s.begin\_interval\_time

FROM dba\_hist\_sqlstat q,

    (

    SELECT /\*+ NO\_MERGE \*/ MIN(snap\_id) min\_snap, MAX(snap\_id) max\_snap , to\_char(ss.begin\_interval\_time,'DD-MON-YY') begin\_interval\_time

    FROM dba\_hist\_snapshot ss

    WHERE ss.begin\_interval\_time BETWEEN (SYSDATE - &No\_Days) AND SYSDATE

    group by to\_char(ss.begin\_interval\_time,'DD-MON-YY')) s

WHERE q.snap\_id BETWEEN s.min\_snap AND s.max\_snap

  AND q.sql\_id IN ( '&SQLID')

order by 4

/

=================================================================================================

**OUTPUT Examples**

**SQL\_ID HASH VALUE TIMESTAMP**

0zwrsg1nh990q 3180314405 22-FEB-13

0zwrsg1nh990q 1404930042 05-APR-13

0zwrsg1nh990q 794980781 02-MAY-13

0zwrsg1nh990q 1669346102 01-AUG-13

0zwrsg1nh990q 1233168671 02-AUG-13

**Interpreting and Analyzing Execution plan Details:**

We can see the history of execution plan along with hash value form awr history tables,

SELECT tf.\* FROM DBA\_HIST\_SQLTEXT ht, table

(DBMS\_XPLAN.DISPLAY\_AWR(ht.sql\_id,null, null, 'ALL' )) tf

WHERE ht.sql\_id='9hnyz5f8g2mgs';

Or

SET LINESIZE 130

SET LONG 10000;

SET PAGESIZE 0

Select \* from TABLE(dbms\_xplan.display\_awr('50tk94ucg18s6'));

Example below,

Plan hash value: **218161689**

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time | Pstart| Pstop | TQ |IN-OUT| PQDistrib

---------------------------------

| 13 | PX SEND| :TQ10000| 27 | 1350 | 234 (0)| 00:00:03 | | | Q1,00 | P->P | BROADCAST |

| 14 | TABLE ACCESS BY LOCAL INDEX ROWID| PRICE\_HIST | 27 | 1350 | 234 (0)| 00:00:03 | | | Q1,00 | PCWC | |

| 15 | NESTED LOOPS | | 27 | 1809 | 199 (4)| 00:00:03 | | | Q1,00 | PCWP | |

| 16 | PX BLOCK ITERATOR | | | | | | | | Q1,00 | PCWC | |

| 17 | TABLE ACCESS FULL | SPC\_CHAIN\_PRODUCT | 1 | 17 | 166 (4)| 00:00:02 | | | Q1,00 | PCWP | |

Like from above we can see what are the different stages of the query, what is the operation, time consuming, cost, cpu, execution time of each stage, objects accessed, which path(full table scan or index path etc….

After analysis decide about which hash value to pin.

Depending up on the above analysis, we can get a clear picture where it’s taking much time.

Please Navigate to the /utl directory from sqlt installation directory where we can find the Below two scripts

**cd orabackup/sqlt/sqlt/sqlt/utl**

sqltprofile.sql

coe\_xfr\_sql\_profile.sql

Any of the above scripts in above location, we can use it for pinning an old plan,

**Example:**

SQL> start coe\_xfr\_sql\_profile.sql 50tk94ucg18s6

Available Hash values will be listed below during the above execution of the script,

3180314405

1404930042

794980781

1669346102

3620676433

**Enter hash value:** 3620676433

Depending up on above hash value it would generate a sql script in which sql profile is created and once we execute the script it would be enabled.

**coe\_xfr\_sql\_profile\_<sql\_id>\_<hash\_value>.sql**

coe\_xfr\_sql\_profile\_**50tk94ucg18s6\_3620676433**.sql;

Part of the output,

------------------------Script contents --------------------------------No need to run below , script does it, pasted below for reference.

REM coe\_xfr\_sql\_profile\_50tk94ucg18s6\_3620676433.sql

REM

REM DESCRIPTION

REM This script was generated by coe\_xfr\_sql\_profile.sql and it

REM contains the commands to create a manual custom SQL Profile

REM based on SQL\_ID 50tk94ucg18s6 AND PLAN\_HASH\_VALUE 3620676433.

……………………………………………………………….

…………………………………………………………………………………….

We have to execute the above script to link the plan to the sql id,In-turn its calling dbms package to create sql profile and to import sql\_profile.

**Plan 3620676433 has been linked to SQL 50tk94ucg18s6.**

**Review log coe\_xfr\_sql\_profile\_50tk94ucg18s6\_3620676433.log.**

Plan has been linked to the appropriate hash value.

**Testing whether its using the sql profile created:**

SQL> explain plan for

* Sql\_query
* Explained. 🡪 output.
* Set autotrace on
* Use dbms\_plan table to display the execution plan,

SET LINESIZE 130

SET PAGESIZE 0

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);

End of the plan you would see the sql profile has been enabled.

**This implemented profile will be effective from the next run only, not if it’s running currently.**