HOWTO: Create SQL Profile and SPM baseline

By: Karl Arao

**(best viewed using View -> Web Layout)**

1. Create a SQL profile

* The goal is for the SQL\_ID to use the good or best plan (PHV)

Here are the different ways to create a SQL Profile on a SQL\_ID

|  |  |  |  |
| --- | --- | --- | --- |
| **copy\_plan\_hash\_value.sql** | **create\_hint\_sqlprofile.sql** | **create\_sql\_profile-goodbad.sql** | **COE\_\* scripts** |
| On the same SQL\_ID, create a profile using one of its good PHVs  <https://github.com/karlarao/scripts/blob/master/performance/copy_plan_hash_value.sql>  If the PHV is not found in cursor cache use this  <https://github.com/karlarao/scripts/blob/master/performance/create_sql_profile_awr.sql> | Creates 1 hint on the existing SQL\_ID  <https://github.com/karlarao/scripts/blob/master/performance/create_hint_sqlprofile.sql> | Attach the PHV of good SQL\_ID to the bad SQL\_ID  <https://github.com/karlarao/scripts/blob/master/performance/create_sql_profile-goodbad.sql> | Create SQL Profile using COE\_\* scripts (I use the 12c version these days).   * Ways of using this   + You can use this to create a SQL Profile     - Pull COE from the good plan then create SQL Profile on the same SQL\_ID   + Create a backup of the SQL Profile to implement on another environment     - Scenario: good plan on test env and bad plan on prod     - Generate the coe\_\* file on test env     - Execute the coe\_\* on prod   + Rewrite SQL and edit OTHER\_XML to achieve the good plan and copy to the original SQL\_ID     - You have a modified test SQL (added hints) and want to also use the OTHER\_XML hints of the original SQL\_ID, this results to new SQL\_ID     - You have to replace the SQL section of COE w/ the new SQL TEXT     - The new SQL\_ID (test SQL) will have the good plan which you can attach to the original (bad performing) SQL\_ID using **create\_sql\_profile-goodbad.sql**   <https://github.com/karlarao/scripts/blob/master/performance/coe_xfr_sql_profile.sql>  <https://github.com/karlarao/scripts/blob/master/performance/coe_xfr_sql_profile_112.sql>  <https://github.com/karlarao/scripts/blob/master/performance/coe_xfr_sql_profile_12c.sql> |
| Examples:  Create a profile by copying the plan\_hash\_value from the same SQL (let’s say the previous good plan\_hash\_value exist, and you want the SQL\_ID to use that) or from a testcase SQL w/ a good plan and you want that used by the production SQL  HCMPRD1> @copy\_plan\_hash\_value.sql  Enter value for plan\_hash\_value to generate profile from (X0X0X0X0): 3609883731 <-- this is the good plan  Enter value for sql\_id to attach profile to (X0X0X0X0): c7tadymffd34z  Enter value for child\_no to attach profile to (0):  Enter value for category (DEFAULT):  Enter value for force\_matching (false):    PL/SQL procedure successfully completed. | Examples:  We have the following SQL that ran long in DWTST that we fixed through SQL profile (from 50mins to 3secs)  SELECT MIN("LOAD\_DATE") FROM "DIM"."ENS\_CSM\_SUMMARY\_DT\_GLT"  @create\_hint\_sqlprofile  Enter value for sql\_id: dg7zj0q9qa2gf  Enter value for profile\_name (PROFILE\_sqlid\_MANUAL): <just hit ENTER here>  Enter value for category (DEFAULT): <just hit ENTER here>  Enter value for force\_matching (false): <just hit ENTER here>  Enter value for hint\_text: NO\_PARALLEL  Profile PROFILE\_dg7zj0q9qa2gf\_MANUAL created.  This will make the query run in serial through a profile hint which is the fix for this issue. | Examples:  Create a profile by copying the plan\_hash\_value from a different SQL\_ID (let’s say you rewrote the SQL and you want to inject that new plan to the old SQL\_ID)  dwbs001s1(sys): @create\_sql\_profile-goodbad.sql  Enter value for goodsql\_id: 22s34g2djar10  Enter value for goodchild\_no (0): <HIT ENTER>  Enter value for badsql\_id: 00fnpu38hz98x  Enter value for badchild\_no (0): <HIT ENTER>  Enter value for profile\_name (PROF\_sqlid\_planhash): <HIT ENTER>  Enter value for category (DEFAULT): <HIT ENTER>  Enter value for force\_matching (FALSE): <HIT ENTER>  Enter value for plan\_hash\_value: <HIT ENTER>  SQL Profile PROF\_00fnpu38hz98x\_ created. | Examples:  HOWTO: create a manual SQL Profile <https://www.evernote.com/shard/s48/sh/f1bda7e9-2ced-4794-8c5e-32b1beac567b/96cd95cebb8f3cad0329833d7aa4a328>  There’s a tool inside SQLTXPLAIN (search this in MOS) it’s called coe\_xfr\_sql\_profile <https://raw.githubusercontent.com/karlarao/scripts/master/performance/coe_xfr_sql_profile_12c.sql> What this does is you run it in a SQL\_ID and PLAN\_HASH\_VALUE and it will create a sql file. And when you run this on another environment it will create a sql profile on that SQL\_ID and PLAN\_HASH\_VALUE combination. So it becomes a backup of that SQL performance or another way of migrating or backing up profiles across environments.  In summary you have full control over the code. Rewriting or putting hints (but not too much) to behave optimally.  You can also push the generate COE file on your git repo for tracking and version control and propagate across environments. You can also baseline on top of the rewrite or hints but make sure this is maintained across environments. |

Be aware that it’s possible to have two SQL profiles for the same SQL statement: one with force\_match set to FALSE,

and the other with force\_match set to TRUE. If two SQL profiles exist, the one with force\_match set to FALSE takes

precedence over the one with force\_match set to TRUE. This is sound because the one with force\_match set to FALSE

is more specific than the other. That means that you might have one SQL profile to cover most of the literals, and

another for literals requiring particular handling (for example, when a restriction based on a literal is applied

to a column containing skewed data).

1. Create the SQL Plan Baseline

* Mental model
  + Hierarchy: one **EXACT\_MATCHING\_SIGNATURE** -> will have one SQL Handle -> and multiple PLAN\_NAME
  + Use **LOAD\_PLANS\_FROM\_CURSOR\_CACHE** to load SQL\_ID and PHV
    - If PHV is not in cursor cache need to use **LOAD\_PLANS\_FROM\_SQLSET**
  + LOAD\_PLANS\_\* will allow you to create a baseline with just the good plans in it. This will ignore the execution of bad plans
  + You can also enable/disable the PLAN\_NAMEs -> by choosing the SQL Handle + PLAN\_NAME

-- create the baseline

DECLARE

my\_plans pls\_integer;

BEGIN

my\_plans := DBMS\_SPM.LOAD\_PLANS\_FROM\_CURSOR\_CACHE(sql\_id => 'a5jq5khm9w64n',plan\_hash\_value=>'246648590', fixed =>'YES', enabled=>'YES');

END;

/

<https://github.com/karlarao/scripts/blob/master/performance/spm_baselines.sql>

11:46:34 KARLARAO@cdb1> @spm\_baselines

Enter value for sql\_text:

Enter value for exact\_matching\_signature:

PARSING\_ CREATED PLAN\_NAME SQL\_HANDLE SQL\_TEXT OPTIMIZER\_COST ENA ACC FIX REP ORIGIN

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

KARLARAO 03/23/20 11:18:18 SQL\_PLAN\_fahs3brrwbxcm950a48a8 SQL\_e543035defc5f593 select \* from skew where skew=3 8 YES YES YES YES MANUAL-L

OAD

KARLARAO 03/23/20 11:41:32 SQL\_PLAN\_fahs3brrwbxcm08e93fe4 SQL\_e543035defc5f593 select \* from skew where skew=3 2 YES YES YES YES MANUAL-L

OAD

-- add the other plan

**-- you can even use a different SQL\_ID, what matters is the text matches the EXACT\_MATCHING\_SIGNATURE to be tied to SQL\_HANDLE as a new SQL PLAN\_NAME**

DECLARE

my\_plans pls\_integer;

BEGIN

my\_plans := DBMS\_SPM.LOAD\_PLANS\_FROM\_CURSOR\_CACHE(sql\_id => 'a5jq5khm9w64n',plan\_hash\_value=>'1949605896', fixed =>'YES', enabled=>'YES');

END;

/

Show plans of SQL\_HANDLE

<https://github.com/karlarao/scripts/blob/master/performance/spm_plans.sql>

11:46:38 KARLARAO@cdb1>

11:46:38 KARLARAO@cdb1> @spm\_plans

Enter value for sql\_handle: SQL\_e543035defc5f593

Enable SQL\_HANDLE

<https://github.com/karlarao/scripts/blob/master/performance/spm_enable.sql>

set verify off

declare

myplan pls\_integer;

begin

myplan:=DBMS\_SPM.ALTER\_SQL\_PLAN\_BASELINE (sql\_handle => '&sql\_handle',plan\_name => '&plan\_name',attribute\_name => 'ENABLED', attribute\_value => '&YES\_OR\_NO');

end;

/11:49:35 KARLARAO@cdb1> 11:49:35 KARLARAO@cdb1> 11:49:35 2 11:49:35 3 11:49:35 4 11:49:35 5 11:49:35 6

Enter value for sql\_handle: SQL\_e543035defc5f593

Enter value for plan\_name: SQL\_PLAN\_fahs3brrwbxcm08e93fe4

Enter value for yes\_or\_no: no

PL/SQL procedure successfully completed.

@spm\_baselines

Enter value for sql\_text:

Enter value for exact\_matching\_signature:

PARSING\_ CREATED PLAN\_NAME SQL\_HANDLE SQL\_TEXT OPTIMIZER\_COST ENA ACC FIX REP ORIGIN

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

KARLARAO 03/23/20 11:18:18 SQL\_PLAN\_fahs3brrwbxcm950a48a8 SQL\_e543035defc5f593 select \* from skew where skew=3 8 YES YES YES YES MANUAL-L

OAD

KARLARAO 03/23/20 11:41:32 SQL\_PLAN\_fahs3brrwbxcm08e93fe4 SQL\_e543035defc5f593 select \* from skew where skew=3 2 NO YES YES YES MANUAL-L

OAD

Drop Baseline

<https://github.com/karlarao/scripts/blob/master/performance/spm_drop_baseline.sql>

drop the individual baselines

set verify off

DECLARE

plans\_dropped PLS\_INTEGER;

BEGIN

plans\_dropped := DBMS\_SPM.drop\_sql\_plan\_baseline (

sql\_handle => '&sql\_handle',

plan\_name => '&plan\_name');

DBMS\_OUTPUT.put\_line(plans\_dropped);

END;

/

**LOAD\_PLANS\_FROM\_SQLSET (example)**

--######################################################################

if PHV is not on cursor cache then the DBMS\_SPM.LOAD\_PLANS\_FROM\_SQLSET has to be used

--########################################################################

exec dbms\_sqltune.create\_sqlset(sqlset\_name => '93c0q2r788x6c\_sqlset\_test',description => 'sqlset descriptions');

declare

baseline\_ref\_cur DBMS\_SQLTUNE.SQLSET\_CURSOR;

begin

open baseline\_ref\_cur for

select VALUE(p) from table(

DBMS\_SQLTUNE.SELECT\_WORKLOAD\_REPOSITORY(&begin\_snap\_id, &end\_snap\_id,'sql\_id='||CHR(39)||'&sql\_id'||CHR(39)||' and plan\_hash\_value=2849155601',NULL,NULL,NULL,NULL,NULL,NULL,'ALL')) p;

DBMS\_SQLTUNE.LOAD\_SQLSET('93c0q2r788x6c\_sqlset\_test', baseline\_ref\_cur);

end;

/

SELECT NAME,OWNER,CREATED,STATEMENT\_COUNT FROM DBA\_SQLSET where name='93c0q2r788x6c\_sqlset\_test';

select \* from table(dbms\_xplan.display\_sqlset('93c0q2r788x6c\_sqlset\_test','&sql\_id'));

select sql\_handle, plan\_name, origin, enabled, accepted, fixed, module from dba\_sql\_plan\_baselines;

set serveroutput on

declare

my\_int pls\_integer;

begin

my\_int := dbms\_spm.load\_plans\_from\_sqlset (

sqlset\_name => '93c0q2r788x6c\_sqlset\_test',

basic\_filter => 'sql\_id="93c0q2r788x6c",

sqlset\_owner => 'SYS',

fixed => 'YES',

enabled => 'YES');

DBMS\_OUTPUT.PUT\_line(my\_int);

end;

/

select sql\_handle, plan\_name, origin, enabled, accepted, fixed, module from dba\_sql\_plan\_baselines;

-- make sure the additional PHV is ACCEPTED and FIXED

SET SERVEROUTPUT ON

DECLARE

l\_plans\_altered PLS\_INTEGER;

BEGIN

l\_plans\_altered := DBMS\_SPM.alter\_sql\_plan\_baseline(

sql\_handle => 'SQL\_c244ec33ef56024a',

plan\_name => 'SQL\_PLAN\_c4j7c6grpc0kaf8003e90',

attribute\_name => 'ACCEPTED',

attribute\_value => 'YES');

DBMS\_OUTPUT.put\_line('Plans Altered: ' || l\_plans\_altered);

END;

/

set serveroutput on

DECLARE

l\_plans\_altered PLS\_INTEGER;

BEGIN

l\_plans\_altered := DBMS\_SPM.alter\_sql\_plan\_baseline(

sql\_handle => 'SQL\_c244ec33ef56024a',

plan\_name => 'SQL\_PLAN\_c4j7c6grpc0kaf8003e90',

attribute\_name => 'FIXED',

attribute\_value => 'YES');

DBMS\_OUTPUT.put\_line('Plans Altered: ' || l\_plans\_altered);

END;

/