ORACLE OPEN WORLD 2014 September 28 - October 2 San Francisco, CA

UYOUG

My Experience Using SQL Plan Baselines

Nelson Calero

September 2014

Pythian love your data

About me

ORACLE ACE

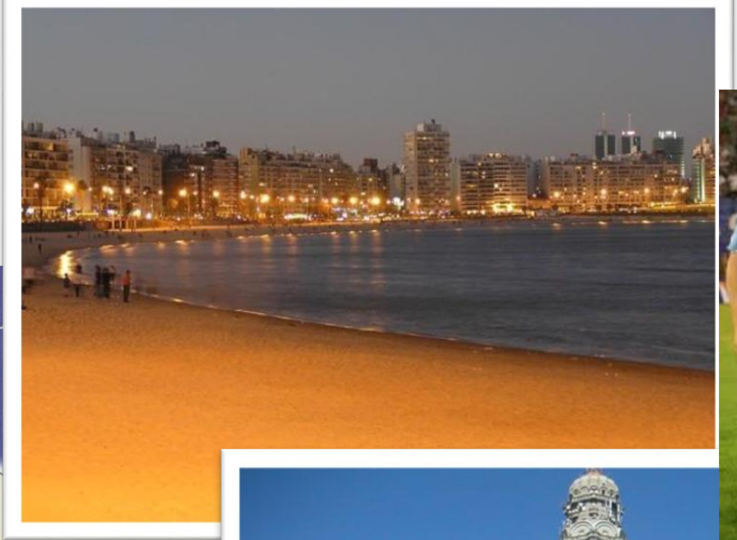
ORACLE
Certified Professional
Oracle Database 11g
Administrator

LAOUC

- Database Consultant at Pythian
- Computer Engineer
- Oracle Certified Professional DBA 10g/11g
- Oracle ACE
- Working with Oracle tools and Linux environments since 1996
- DBA Oracle (since 2001) & MySQL (since 2005)
- Oracle University Instructor
- Co-founder and President of the Oracle user Group of Uruguay
- LAOUC Director of events
- Blogger and frequent speaker: Oracle Open World, Collaborate, OTN Tour, JIAP, MySQL/NoSQL conferences

http://www.linkedin.com/in/ncalero

@ncalerouy

OCÉANO ATL
VENEZUELA
COLOMBIA
GUAYANAS
ECUADOR
BRASIL
PERÚ
BOLIVIA
PARAGUAY
OCÉANO PACÍFICO
CHILE
ARGENTINA
URUGUAY
OCÉANO ATLÁNTICO


Pythian overview

- 17 Years of data infrastructure management consulting
- 170+ Top brands
- 10000+ Systems under management
- Over 200 DBAs in 28 countries
- Top 5% of DBA work force, 9 Oracle ACEs, 4 Microsoft MVPs
- Oracle, Microsoft, MySQL, Netezza, Hadoop, MongoDB, Oracle Apps, Enterprise Infrastructure

ORACLE Platinum Partner

Microsoft Partner
Silver Data Platform
Gold Data Platform

MySQL Enterprise
Authorized Platinum REMOTE DBA SERVICES PARTNER

Why SQL Plan Management?

- Oracle has several functionalities to help troubleshoot performance of SQL
- There is a lot of information online, official and by the community
  - https://blogs.oracle.com/optimizer/ is the Optimizer Team!
  - OTN Virtual Technology summit lab – Create and evolve a Baseline: http://www.oracle.com/technetwork/articles/database/create-sql-plan-baseline-2237506.html

✓ Share my experience using it on production environments

- RAC and single instance
- Closed source applications (Black Box)
- 1000+ users, 40+ concurrent

Today’s topics

- What is SQL Plan Management?
- Simple example
  - SQL matching
  - Plan matching
  - New plan generation
  - Helper scripts
- Evolution on 11g / 12c
- Challenges
- Troubleshooting
- Daily management
  - 11g suggested approach

Introduction to SQL Plan Management

What

Why

How

- SQL Plan Management is a new feature of Oracle 11.1 with no extra cost, available on Enterprise edition, enabled by default.
- The Oracle Optimizer is able to use only well-known execution plans, avoiding the usage of others plans we know inferior performance (regression).

Introduction to SQL Plan Management

What

Why

How

Changes in SQL plan execution can lead to worse execution times, and can impact system performance. It can be caused from a variety of reasons:

- Optimizer version, statistics, and parameters
- Structural objects changes (datatype, indexes, partitions)
- System settings
- System growth (skewed data, concurrency)
- And many more. See additional reason here: http://jonathanlewis.wordpress.com/2013/12/23/plan-changes/

Oracle Database already has several ways to control plan execution that works at different stages in the cost based Optimizer:

- Stored Outlines (deprecated in 11.1)
- SQL Profiles: adds statistics to the plan evaluation phase of the Optimizer (DBMS_SQLTUNE)
- SQL Hints: forces the optimizer to follow a specific path/action

Introduction to SQL Plan Management

What

Why

How

- Baselines are new objects stored on SYSAUX tablespace (SMB area)
- Each SQL statement can have many baselines
- Only **enabled** and **accepted** plans are used
- Fixed plans take precedence, and no new plans are auto added
- Stages: plan load – plan selection – plan evolution
- New evolve advisor autotask in 12c
- SQL needs to run more than once to be considered
- Baselines are global - not per schema

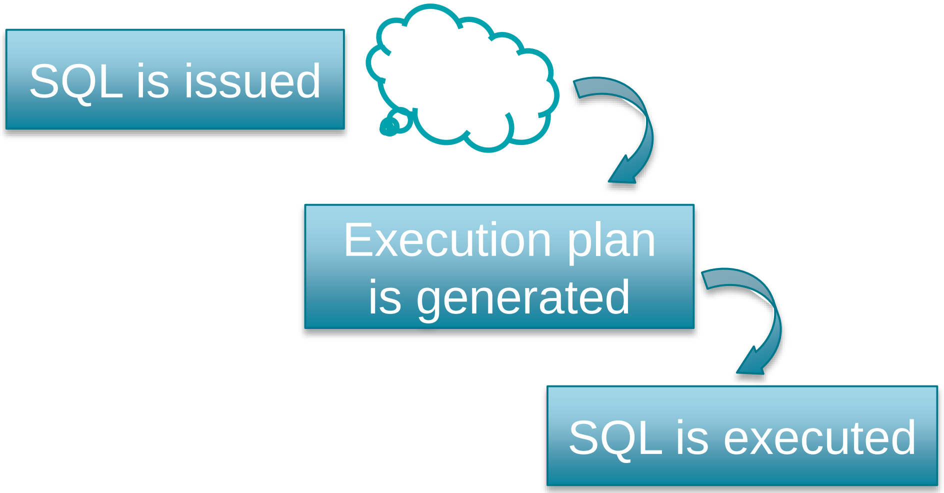
Initialization parameters

- OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES (Default: FALSE)
- OPTIMIZER_USE_SQL_PLAN_BASELINES (Default: TRUE)

Introduction to SQL Plan Management - How

Old history

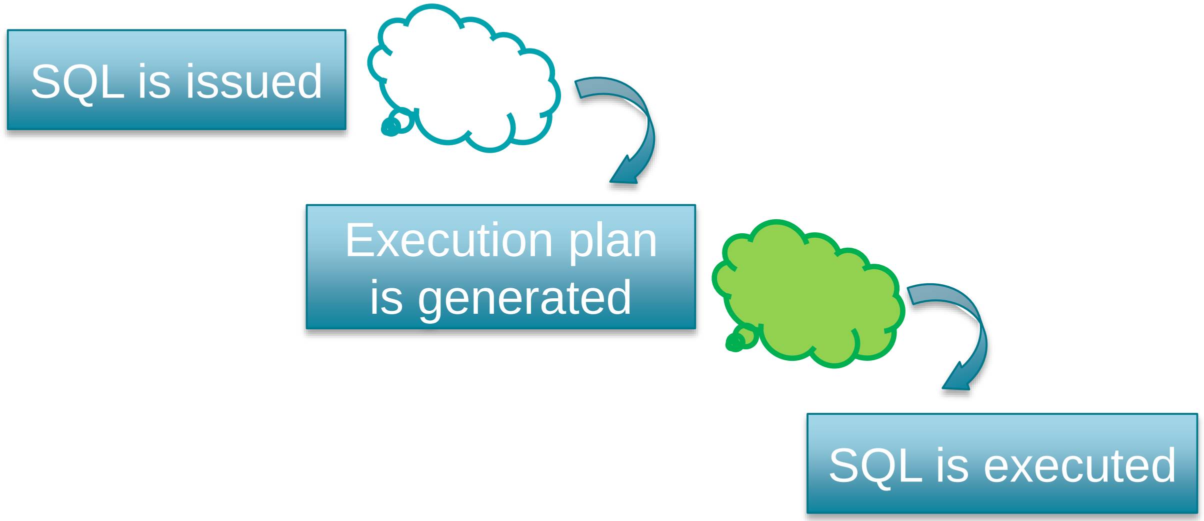
Oracle Cost Based Optimizer (CBO) **Before 11g**



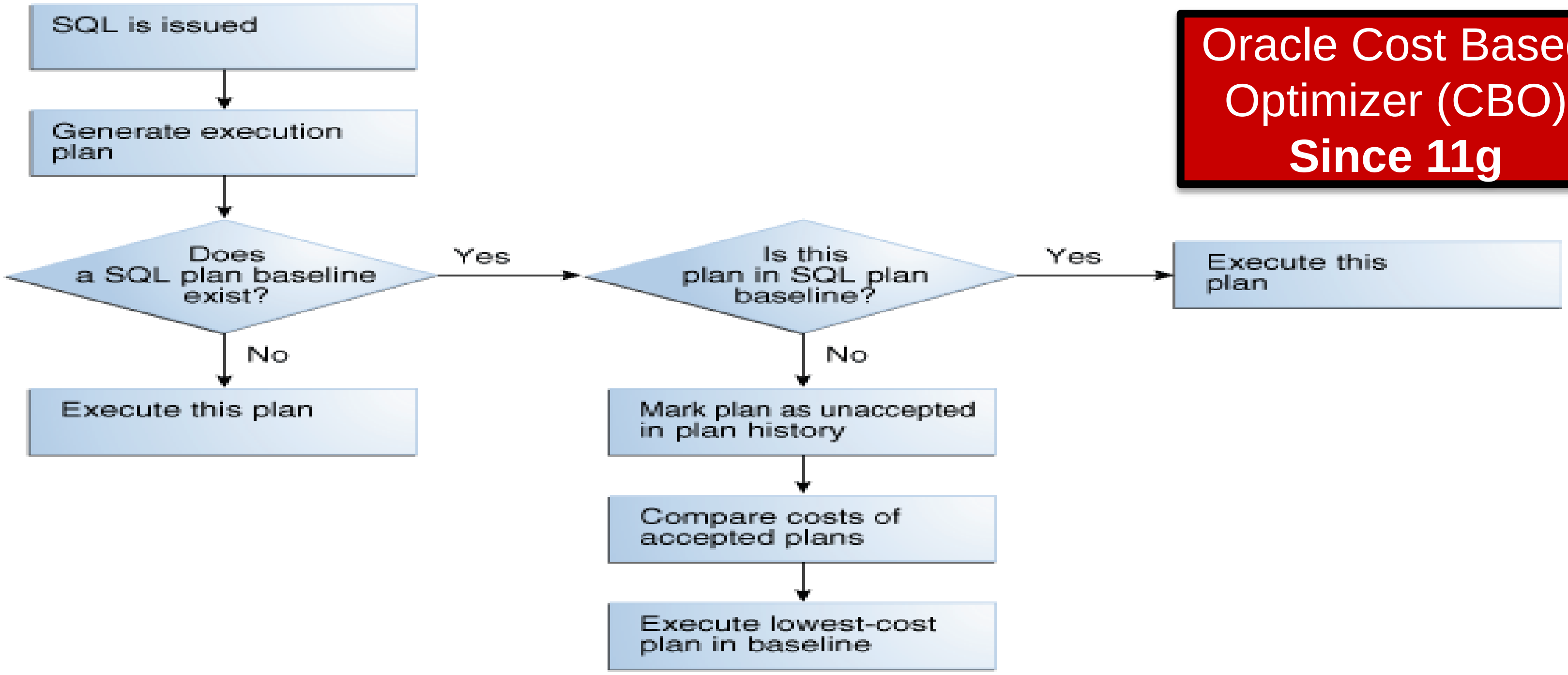
Introduction to SQL Plan Management - How

Baselines are the last step evaluated by the Oracle Optimizer: if an **enabled** and **accepted** plan for the statement exists, it will be used, maybe discarding the already generated plan.

Oracle Cost Based Optimizer (CBO)
Since 11g



Introduction to SQL Plan Management - How



SQL is issued
Generate execution plan
Does a SQL plan baseline exist?
Yes
No
Execute this plan
Is this plan in SQL plan baseline?
Yes
No
Execute this plan
Mark plan as unaccepted in plan history
Compare costs of accepted plans
Execute lowest-cost plan in baseline

Oracle Cost Based Optimizer (CBO)
Since 11g

http://docs.oracle.com/database/121/TGSQL/tgsql_spm.htm#TGSQL626

SPM example – setup

```
SQL*Plus: Release 11.2.0.3.0 Production on Thu Aug 7 23:46:48 2014

Copyright (c) 1982, 2011, Oracle.  All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> create table pp(n number, c varchar2(100));
Table created.

SQL> insert into pp select rownum, object_id from dba_objects where rownum < 100;
99 rows created.

SQL> create index pp_idx on pp(n);
Index created.

SQL> exec dbms_stats.gather_table_stats(user, ‘PP');
PL/SQL procedure successfully completed.
```

SPM example – capture SQL execution

```
SQL> alter session set optimizer_capture_sql_plan_baselines = TRUE;
Session altered.

SQL> var n number;
SQL> exec :n := 1;
PL/SQL procedure successfully completed.

SQL> select * from pp where n=:n;
         N C
---------- ------------------------------------------------------------
         1 20

SQL> /
         N C
---------- ------------------------------------------------------------
         1 20

SQL> alter session set optimizer_capture_sql_plan_baselines = FALSE;
Session altered.
```

Pythian love your data

SPM example – SQL and baseline matching

```
select signature, sql_handle, plan_name, enabled, accepted, fixed, sql_text
from dba_sql_plan_baselines;

            SIGNATURE SQL_HANDLE           PLAN_NAME                      ENA ACC FIX SQL_TEXT
--------------------- -------------------- ------------------------------ --- --- --- ------------------
  1245871306398155660 SQL_114a395a2db6c38c SQL_PLAN_12kjtb8qvdhwc8a71e415 YES YES NO  select * from pp


select sql_id, exact_matching_signature, sql_text
from v$sql
where sql_text like 'select * from pp %';

SQL_ID        EXACT_MATCHING_SIGNATURE SQL_TEXT
------------- ------------------------ ------------------------------
0a14b3yhux040      1245871306398155660 select * from pp where n=:n
```

SPM example – SQL and baseline matching

Our Baseline:

```
select signature, plan_name from dba_sql_plan_baselines;
           SIGNATURE             PLAN_NAME
--------------------- ---------------------
1245871306398155660    SQL_114a395a2db6c38c
```

```
select * from pp where n=:n;
select * from pp where n=2;
select * from pp where n=1;

select sql_id, exact_matching_signature, force_matching_signature, sql_text
from v$sql
where sql_text like 'select * from pp %';

SQL_ID        EXACT_MATCHING_SIGNATURE FORCE_MATCHING_SIGNATURE SQL_TEXT
------------- ------------------------ ------------------------ ------------------------------
D387kpdvh4anb     11466572521187337874     15110712337079575277 select * from pp where n=2
0a14b3yhux040      1245871306398155660      1245871306398155660 select * from pp where n=:n
86svufrd72xqg     14183734311806369169     15110712337079575277 select * from pp where n=1
```

SPM example – plan from last run in SGA

```
SQL> select * from table(dbms_xplan.display_cursor);
SQL_ID 0a14b3yhux040, child number 1
-------------------------------------
select * from pp where n=:n

Plan hash value: 2547524127
------------------------------------------------------------------------------------
| Id  | Operation                   | Name   | Rows  | Bytes | Cost  (%CPU)| Time     |
------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |        |       |       |     2 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| PP     |     1 |     6 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | PP_IDX |     1 |       |     1   (0)| 00:00:01 |
------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("N"=:N)

Note
- SQL plan baseline SQL_PLAN_12kjtb8qvdhwcdeb317bf used for this statement
```

SPM example – hints/plan used by Baseline

```
SQL> select * from table(dbms_xplan.display_sql_plan_baseline(
                              plan_name => 'SQL_PLAN_12kjtb8qvdhwcdeb317bf', format => 'OUTLINE'));
--------------------------------------------------------------------------------
SQL handle: SQL_114a395a2db6c38c
SQL text: select * from pp where n=:n
--------------------------------------------------------------------------------
Plan name: SQL_PLAN_12kjtb8qvdhwcdeb317bf   Plan id: 3736278975
Enabled: YES       Fixed: NO  Accepted: YES       Origin: AUTO-CAPTURE
--------------------------------------------------------------------------------
Outline Data from SMB:
  /*+
      BEGIN_OUTLINE_DATA
      INDEX_RS_ASC(@"SEL$1" "PP"@"SEL$1" ("PP"."N"))
      OUTLINE_LEAF(@"SEL$1")
      ALL_ROWS
      DB_VERSION('11.2.0.3')
      OPTIMIZER_FEATURES_ENABLE('11.2.0.3')
      IGNORE_OPTIM_EMBEDDED_HINTS
      END_OUTLINE_DATA
  */
```

Where is the
Plan hash value?
2547524127

SPM example – view definition

```
select text from dba_views where view_name='DBA_SQL_PLAN_BASELINES';

SELECT /*+ dynamic_sampling(3) */
    so.signature,
    st.sql_handle,
...
    DECODE(BITAND(so.flags, 1), 1, 'YES', 'NO'),            -- enabled
    DECODE(BITAND(so.flags, 2), 2, 'YES', 'NO'),            -- accepted
...
FROM
    sqlobj$         so,
    sqlobj$auxdata ad,
    sql$text        st
WHERE
    so.signature = st.signature AND
    ad.signature = st.signature AND
    so.signature = ad.signature AND
    so.plan_id = ad.plan_id AND
    so.obj_type = 2 AND
    ad.obj_type = 2
```

SPM example – similar now including PHV2

```
SELECT /*+ dynamic_sampling(3) */ so.signature, so.name plan_name,
    DECODE(BITAND(so.flags, 1), 1, 'YES', 'NO') enabled,
    DECODE(BITAND(so.flags, 2), 2, 'YES', 'NO') accepted,
    so.plan_id phv2
FROM
    sqlobj$         so,
    sqlobj$auxdata ad,
    sql$text        st
WHERE
    so.signature = st.signature AND
    ad.signature = st.signature AND
    so.signature = ad.signature AND
    so.plan_id = ad.plan_id AND
    so.obj_type = 2 AND
    ad.obj_type = 2;

            SIGNATURE PLAN_NAME                      ENA ACC       PHV2
--------------------- ------------------------------ --- --- ----------
  1245871306398155660 SQL_PLAN_12kjtb8qvdhwc8a71e415 YES YES 2322719765
```

SPM example – SQL matching plan number

```
col phv2 for 9999999999
SELECT   p.sql_id, p.plan_hash_value, p.child_number, x.phv2
FROM    v$sql_plan p
       ,xmltable('for $i in /other_xml/info where $i/@type eq "plan_hash_2" return $i'
                  passing xmltype(p.other_xml) columns phv2 number path '/') x
WHERE p.sql_id = '0a14b3yhux040'
  and p.other_xml is not null;

SQL_ID        PLAN_HASH_VALUE CHILD_NUMBER PHV2
------------- --------------- ------------ --------------------
0a14b3yhux040      2932947496            0 2322719765
```

Our previous result from the almost similar to DBA_SQL_PLAN_BASELINES:

```
 SIGNATURE PLAN_NAME                      ENA ACC       PHV2
--------------------- ------------------------------ --- --- ----------
  1245871306398155660 SQL_PLAN_12kjtb8qvdhwc8a71e415 YES YES 2322719765
```

Introduction to SQL Plan Management - How

- Important attributes of a baseline (DBA_SQL_PLAN_BASELINES):
  - ENABLED – this can be used if accepted
  - ACCEPTED – this can be used if enabled
  - FIXED – use this as preferred– no evolution
  - PARSING_SCHEMA_NAME
  - ADAPTIVE – new in 12c

- Purging policy – weekly autotask to delete unused plans. Parameters:
  - SPACE_BUDGET_PERCENT - % used - default 10 - alert.log
  - PLAN_RETENTION_WEEKS – to delete non used plans - default 53
  - DBA_SQL_MANAGEMENT_CONFIG view
  - Can be changed using DBMS_SPM.CONFIGURE

SQL Plan Management objects

SPM included in the SQL Management Base (SMB).

SBM also has:

- Statement log – SQL$
- Baselines plan history (12c) - SQLOBJ$PLAN
- SQL plan baselines
    - DBA_SQL_PLAN_BASELINES view
    - DBMS_SPM package
- SQL profiles
    - DBA_SQL_PROFILES view
    - DBMS_SQLTUNE / DBMS_AUTO_SQLTUNE package

When to use SPM

- Just another tool for performance management
- Database upgrade: known plans can be captured/exported/imported
- Third party applications: can include exported baselines to guarantee known behaviour

What is automated by advisors?

- SQL Tuning Advisor (Tuning pack licence!) - SYS_AUTO_SQL_TUNING_TASK
  - Create profiles (and baseline if present) if ACCEPT_SQL_PROFILES parameter is TRUE
  - Manually using DBMS_AUTO_SQLTUNE.ACCEPT_SQL_PROFILE
- Evolve advisor on 12c - SYS_AUTO_SPM_EVOLVE_TASK (more later)
  - parameter ACCEPT_PLANS defaults TRUE

All good things without extra effort?

- Forces us to follow the evolution of SQLs who uses it
  - blindly trust is not a good idea
- Needs a good understanding to explain why an existing baseline is not used

More than just plan selection

Plan load/capture and evolution

Creating new Baselines manually:

- capture plans being in use by the instance
  - OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES=TRUE
- load plans from cursor cache
  - DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE
- load plans from SQL tuning set - Oracle Tuning Pack license
  - DBMS_SPM.LOAD_PLANS_FROM_SQLSET

New baselines are generated automatically:

- for statement that already have Baselines created (when new plans are parsed by the optimizer, as **non accepted**) – it is not capture!
- when creating a SQL Profile on a statement that has Baseline (**as accepted**)

Non accepted plans become **accepted** because evolution or DBMS_SPM attribute change

New plan generation

```
SQL> alter index pp_idx invisible;
Index altered.

SQL> select * from pp where n=:n;
         N  C
---------- ---------------------------------------------
         1 20

SQL> select * from table(dbms_xplan.display_cursor);
SQL_ID  0a14b3yhux040, child number 1
-------------------------------------
select * from pp where n=:n

Plan hash value: 2932947496
--------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |       |       |     2 (100)|          |
|*  1 |  TABLE ACCESS FULL| PP   |     1 |     6 |     2   (0)| 00:00:01 |
--------------------------------------------------------------------------
```

New plan generation

```
select sql_handle, plan_name, enabled, accepted, fixed, reproduced, sql_text
from dba_sql_plan_baselines;

SQL_HANDLE           PLAN_NAME                      ENA ACC FIX REP SQL_TEXT
-------------------- ------------------------------ --- --- --- --- ------------------------------
SQL_114a395a2db6c38c SQL_PLAN_12kjtb8qvdhwc8a71e415 YES NO  NO  YES select * from pp where n=:n
SQL_114a395a2db6c38c SQL_PLAN_12kjtb8qvdhwcdeb317bf YES YES NO  NO  select * from pp where n=:n
```

As the existing baseline could not be reproduced, a new plan was used, and automatically added as non accepted.

Which plan was the original?

```
      select * from table(dbms_xplan.display_sql_plan_baseline(
                                 plan_name => 'SQL_PLAN_12kjtb8qvdhwcdeb317bf'));
```

New plan generation – 11g shows wrong plan

```
--------------------------------------------------------------------------------
SQL handle: SQL_114a395a2db6c38c
SQL text: select * from pp where n=:n
--------------------------------------------------------------------------------
Plan name: SQL_PLAN_12kjtb8qvdhwcdeb317bf    Plan id: 3736278975
Enabled: YES       Fixed: NO  Accepted: YES       Origin: AUTO-CAPTURE
--------------------------------------------------------------------------------

Plan hash value: 2932947496
--------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |     6 |     2   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| PP   |     1 |     6 |     2   (0)| 00:00:01 |
--------------------------------------------------------------------------

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

   1 - filter("N"=TO_NUMBER(:N))
```

New plan generation – 12c shows correct plan

```
--------------------------------------------------------------------------------
SQL handle: SQL_114a395a2db6c38c
SQL text: select * from pp where n=:n
--------------------------------------------------------------------------------
Plan name: SQL_PLAN_12kjtb8qvdhwcdeb317bf    Plan id: 3736278975
Enabled: YES       Fixed: NO  Accepted: YES       Origin: AUTO-CAPTURE
--------------------------------------------------------------------------------
Plan hash value: 452276032

------------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name     | Rows       | Bytes       | Cost (%CPU)| Time      |
------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |          |            |             |     2 (100)|           |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PP       |          1 |           6 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | PP_IDX |          1 |             |     1   (0)| 00:00:01 |
------------------------------------------------------------------------------------------------------
…
```

No misunderstandings

Each SQL Baseline stores the original SQL execution plan?

- 11g: no, it regenerates the plan using captured data
    - SQLOBJ$AUXDATA
    - dbms_xplan.display_sql_plan_baseline can show a plan different from the original captured
- 12c: stores the original execution plan
    - SQLOBJ$PLAN

Scripts for creating new Baselines

- SQLTXPLAIN (SQLT) - MOS note 215187.1
  - Tool that helps to diagnose SQL statements performing poorly
  - coe_load_sql_baseline.sql – allows to add a plan generated by a modified SQL to the original
  - Also useful when loading a plan from AWR (must go through SQL tuning set)

Using SQLT to bind a modified plan

```
SQL_ID        EXACT_MATCHING_SIGNATURE PLAN_HASH_VALUE   CHILD SQL_TEXT
------------- ------------------------ --------------- ------- ----------------------------------------
fxdv1cmv8cksa      6873770436048238943      2932947496       0 select /*+ FULL(pp) */ * from pp where n =:n
0a14b3yhux040      1245871306398155660       452276032       0 select * from pp where n=:n

SQL> select * from dba_sql_plan_baselines;
no rows selected

SQL> sta coe_load_sql_baseline 0a14b3yhux040 fxdv1cmv8cksa 2932947496
       …
      Plans Loaded: 1
      sys_sql_handle: "SQL_114a395a2db6c38c"
      sys_plan_name: "SQL_PLAN_12kjtb8qvdhwc8a71e415"

SQL> select * from pp where n=:n;

SQL_ID        EXACT_MATCHING_SIGNATURE PLAN_HASH_VALUE   CHILD SQL_TEXT
------------- ------------------------ --------------- ------- ----------------------------------------
fxdv1cmv8cksa    6873770436048238943        2932947496       0 select /*+ FULL(pp) */ * from pp where n =:n
0a14b3yhux040    1245871306398155660        452276032        0 select * from pp where n=:n
0a14b3yhux040    1245871306398155660        2932947496       1 select * from pp where n=:n
```

How SQL Plan Management interact with ...?

- Stored Outlines (desupport announced on 11.1)
- SQL Profiles – since 10g
- Adaptive Cursor Sharing (ACS) – new 11g
- Adaptive Query Optimization (12c)
  - Adaptive Plans (joins and parallel distribution methods)
  - Adaptive Statistics (at compile time and at runtime))

How SQL Plan Management interact with ...?

- Stored Outlines (desupport announced on 11.1)
- SQL Profiles – since 10g
- Adaptive Cursor Sharing (ACS) – new 11g
- Adaptive Query Optimization (12c)
  - Adaptive Plans (joins and parallel distribution methods)
  - Adaptive Statistics (at compile time and at runtime))

⇨ Affects plan selection/creation, but SPM is the last step!
Some bugs in the past with profiles+baselines - 12980183

ACS and SPM demo: https://blogs.oracle.com/optimizer/resource/acs-spm-figures/acs-spm-script.sql

Management of new plans (evolution)

Only better performing plans are accepted considering elapsed time, logical IO and CPU time

- Plan reproducibility plays here (ACS/Binds)

11g: manually using DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE

12c: new Evolve Advisor configured as a daily autotask

- SYS_AUTO_SPM_EVOLVE_TASK
- Result report using DBMS_SPM.REPORT_AUTO_EVOLVE_TASK
- Automatically accepts new baselines performing better
  - parameter ACCEPT_PLANS defaults TRUE
- Manually too with DBMS_SPM.CREATE_EVOLVE_TASK
- DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE deprecated

Plan evolution in 11g

```
set long 100000
var e clob;
exec :e := DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE(
                SQL_HANDLE=>'SQL_114a395a2db6c38c',
                COMMIT => 'NO');
print;
```

```
Evolve SQL Plan Baseline Report
-------------------------------------------------------------------------------
Inputs:
-------
  SQL_HANDLE = SQL_114a395a2db6c38c
  PLAN_NAME  =
  TIME_LIMIT = DBMS_SPM.AUTO_LIMIT
  VERIFY     = YES
  COMMIT     = NO

Plan: SQL_PLAN_12kjtb8qvdhwc8a71e415
------------------------------------
  Plan was verified: Time used .06 seconds.
  Plan failed performance criterion: 1.02 times better than baseline plan.

                    Baseline Plan      Test Plan       Stats Ratio
                    -------------      ---------       -----------
  Execution Status:      COMPLETE       COMPLETE
  Rows Processed:               1              1
  Elapsed Time(ms):          .494           .101              4.89
  CPU Time(ms):              .444           .111                 4
  Buffer Gets:                  2              2                 1
  Physical Read Requests:       0              0
  Physical Write Requests:      0              0
  Physical Read Bytes:          0              0
  Physical Write Bytes:         0              0
  Executions:                   1              1
```

```
---------------------------------------------
               Report Summary
---------------------------------------------
Number of plans verified: 1
Number of plans accepted: 0
```

Plan evolution in 11g

Continuing with our example:

- invisible index

```
set long 100000
var e clob;
exec :e :=
DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE(
 SQL_HANDLE=>'SQL_114a395a2db6c38c',
 COMMIT => 'NO');
print;
```

```
-------------------------------------------------------------
                  Evolve SQL Plan Baseline Report
-------------------------------------------------------------

Inputs:
-------
  SQL_HANDLE = SQL_114a395a2db6c38c
  PLAN_NAME  =
  TIME_LIMIT = DBMS_SPM.AUTO_LIMIT
  VERIFY     = YES
  COMMIT     = NO

Plan: SQL_PLAN_12kjtb8qvdhwc8a71e415
------------------------------------
  Plan was not verified.
  Using cost-based plan as could not reproduce any
  accepted and enabled baseline plan.

--------------------------------------------------------------
                        Report Summary
--------------------------------------------------------------
Number of plans verified: 0
Number of plans accepted: 0
```

Plan evolution in 11g – results examples

```
Plan: SQL_PLAN_12kjtb8qvdhwc8a71e415
------------------------------------
  Plan was verified: Time used 7,932 seconds.
  Plan failed performance criterion: performance equal to baseline plan.

           Plan was verified: Time used 152,247 seconds.
           Plan failed performance criterion: 1,20 times worse than baseline plan.

  Plan was verified: Time used 37,411 seconds.
  Plan failed performance criterion: 17,5 times worse than baseline plan.

           Plan was verified: Time used 314,085 seconds.
           Plan verification timed out.

  Plan was verified: Time used 32,618 seconds.
  Error encountered during plan verification (ORA-16960).
    ORA-16960: SQL Analyze no ha podido reproducir el plan deseado.

           Plan was verified: Time used 313,330 seconds.
           Plan passed performance criterion.
```

Plan evolution on 12c

11g evolution deprecated but runs

```
set long 100000
var e clob;
exec :e :=
DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE(
 SQL_HANDLE=>'SQL_114a395a2db6c38c',
 COMMIT => 'NO');
print;
```

```
GENERAL INFORMATION SECTION
-------------------------------------------------------
 Task Information:
-----------------------------------------
 Task Name            : TASK_1604
 Task Owner           : SYS
 Execution Name       : EXEC_1681
 Execution Type       : SPM EVOLVE
 Scope                : COMPREHENSIVE
 Status               : COMPLETED
 Started              : 08/09/2014 21:07:53
 Finished             : 08/09/2014 21:07:53
 Last Updated         : 08/09/2014 21:07:53
 Global Time Limit    : 2147483646
 Per-Plan Time Limit : UNUSED
 Number of Errors     : 0
-------------------------------------------------------

SUMMARY SECTION
-------------------------------------------------------
  Number of plans processed  : 1
  Number of findings         : 1
  Number of recommendations  : 0
  Number of errors           : 0
-------------------------------------------------------
…
```

Pythian love your data®

```
DETAILS SECTION
-------------------------------------------------------
 Object ID          : 2
 Test Plan Name     : SQL_PLAN_12kjtb8qvdhwc8a71e415
 Base Plan Name     : SQL_PLAN_12kjtb8qvdhwca9f022c1
 SQL Handle         : SQL_114a395a2db6c38c
 Parsing Schema     : SYS
 Test Plan Creator  : SYS
 SQL Text           : select * from pp where n=:n

Bind Variables:
-----------------------------
 1  -  (NUMBER):  99

Execution Statistics:
-----------------------------
                    Base Plan        Test Plan
                    ---------------- ----------------
 Elapsed Time (s):  .000003          .000003
 CPU Time (s):      0                0
 Buffer Gets:       0                0
 Optimizer Cost:    2                2
 Disk Reads:        0                0
 Direct Writes:     0                0
 Rows Processed:    0                0
 Executions:        10               10
```

```
FINDINGS SECTION
-------------------------------------------------------
Findings (1):
-----------------------------
 1. The plan was verified in 0.02000 seconds. It failed the benefit
criterion because its verified performance was 0.66667 times worse
than that of the baseline plan.

EXPLAIN PLANS SECTION
-------------------------------------------------------
Baseline Plan
-----------------------------
 Plan Id          : 2201
 Plan Hash Value  : 2851087041
-----------------------------------------------------------------------------------
| Id  | Operation                             | Name   | Rows | Bytes | Cost | Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |        |    1 |     6 |    2 | 00:00:01 |
|   1 |   TABLE ACCESS BY INDEX ROWID BATCHED | PP     |    1 |     6 |    2 | 00:00:01 |
| * 2 |    INDEX RANGE SCAN                   | PP_IDX |    1 |       |    1 | 00:00:01 |
-----------------------------------------------------------------------------------
Predicate Information (identified by operation id):
------------------------------------------
* 2 - access("N"=:N)

Test Plan
-----------------------------
 Plan Id          : 2202
 Plan Hash Value  : 2322719765
---------------------------------------------------------------------
| Id  | Operation           | Name | Rows | Bytes | Cost | Time     |
---------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |    1 | 6     |    2 | 00:00:01 |
| * 1 |   TABLE ACCESS FULL | PP   |    1 | 6     |    2 | 00:00:01 |
---------------------------------------------------------------------
```

Plan evolution on 12c - automatic

```
SELECT owner, description, advisor_name, created, last_modified, status, recommendation_count rc
FROM dba_advisor_tasks
WHERE task_name = 'SYS_AUTO_SPM_EVOLVE_TASK';

OWNER DESCRIPTION                  ADVISOR_NAME         CREATED   LAST_MODI STATUS     RC
----- ---------------------------- -------------------- --------- --------- --------- ---
SYS   Automatic SPM Evolve Task    SPM Evolve Advisor   24-MAY-13 09-AUG-14 COMPLETED   0

SELECT count(*) FROM  DBA_ADVISOR_PARAMETERS
WHERE  TASK_NAME = 'SYS_AUTO_SPM_EVOLVE_TASK';

  COUNT(*)
----------
      29
```

Viewing result report (same output as before):

```
select dbms_spm.report_auto_evolve_task from dual;
```

Plan evolution on 12c - manually

```
var r varchar2(30);
var t varchar2(20);
exec :t := 'EVOTASK';

-- with no parameter will consider all non ACCEPTED baselines
exec :r := dbms_spm.create_evolve_task(task_name=>:t, description=>:t,
                                       plan_name=>'SQL_PLAN_12kjtb8qvdhwc8a71e415');

exec :r := dbms_spm.execute_evolve_task(:t, 'run1', 'first evolve run');

SELECT advisor_name, created, last_modified, status, how_created, recommendation_count rc
FROM dba_advisor_tasks
WHERE task_name = :t;

ADVISOR_NAME          CREATED   LAST_MODI STATUS      HOW_CREATED                RC
-------------------- --------- --------- ----------- ------------------------- ----------
SPM Evolve Advisor    09-AUG-14 09-AUG-14 COMPLETED   CMD                         0

var e clob;
exec :e := dbms_spm.report_evolve_task(:t);
col e for 150
print e                                            ← Same report as before
```

Plan evolution on 12c - manually

Recommendation section: provides command to execute

```
Findings (1):
-----------------------------
 1. None of the accepted plans were reproducible.

Recommendation:
-----------------------------
 Consider accepting the plan. Execute
 dbms_spm.accept_sql_plan_baseline(task_name => 'EVOTASK',
object_id => 2, task_owner => 'SYS');
```

Pythian love your data

Implementing automatic evolution on 11g

```
declare
 cursor c_nuevos_baselines is
   select sql_handle, PLAN_NAME, (select count(1) from DBA_SQL_PLAN_BASELINES b2
                                 where b.sql_handle=b2.sql_handle and b2.accepted='YES') aceptados
   from DBA_SQL_PLAN_BASELINES b
   where enabled='YES' and accepted='NO' and created > trunc(sysdate)-7
   order by sql_handle, PLAN_NAME, created;
begin
  dbms_output.put_line ('var r long;');
  dbms_output.put_line ('set long 100000');
  dbms_output.put_line ('spool evolve-results-'||to_char(sysdate,'yyyymmdd')||'.txt');
  for r_bl in c_nuevos_baselines loop
     dbms_output.put_line ('##################');
     if r_bl.aceptados > 0 then
        dbms_output.put_line ('exec :r:=dbms_spm.evolve_sql_plan_baseline(sql_handle => '''||r_bl.sql_handle||
                                            ''', plan_name=>'''||r_bl.plan_name||''', commit => ''NO'');');
        dbms_output.put_line ('print;');
     end if;
     dbms_output.put_line ('select * from table(dbms_xplan.display_sql_plan_baseline('||
                           'sql_handle=>'''||r_bl.sql_handle||''', plan_name=>'''||r_bl.plan_name||'''));');
  end loop;
  dbms_output.put_line ('spool off');
end;
/
```

Implementing automatic evolution on 11g

Sample run:

```
set serverouput on
spool toevolve.sql
@autoevo11g
spool off
@toevolve

-- to evolve.sql generated:
var r long;
set long 100000
spool evolve-results-20140810.txt
##################
exec :r:=dbms_spm.evolve_sql_plan_baseline(sql_handle => 'SQL_114a395a2db6c38c',
plan_name=>'SQL_PLAN_12kjtb8qvdhwcdeb317bf', commit => 'NO');
print;
select * from table(dbms_xplan.display_sql_plan_baseline(sql_handle=>'SQL_114a395a2db6c38c',
plan_name=>'SQL_PLAN_12kjtb8qvdhwcdeb317bf'));
spool off

PL/SQL procedure successfully completed.
```

Usually the result is a long report. This case is the same as seen today

Challenges?

- SQL without using bind variables
  - Different statements (signature/sql_id), different baselines
  - Change app code. Last resource CURSOR_SHARING=FORCE

- SQL with too many bind variables
  - Maybe cannot reproduce plan – needs a trace to confirm
  - Maybe errors when evolving new ones – (Bug 19326647 on 11.2.0.3)

- SQL already using hints
  - Test original with *alter session set _optimizer_ignore_hints*

- SQL already using SQL Profiles
  - Test original with *alter session set sqltune_category='NNN';*

Challenges

Why my SQL has baselines and none is being used?

1) SPM disabled by parameter
2) Stored outline created – use it and ignore baselines
3) Similar objects on different schemas
4) Baseline cannot be reproduced
5) Bugs

Challenge - similar objects on different schemas

Common on database consolidation scenarios

- Baselines are global for the database (not per schema)
- Same table names, different schemas, same SQL
  - Look at object definition/indexes on each schema

Column PARSING_SCHEMA_NAME in DBA_SQL_PLAN_BASELINES helps

- CREATOR and ORIGIN can lead too

Challenge - baseline cannot be reproduced

- Object changes: if tables used by SQL baseline changes, it can stop being used
  - Remember 11g does not stores captured plans when investigating
- Same PLAN_HASH_VALUE2?
  - minor bugs played here
- Trace to the rescue:
  - It can be reproduced? => session level
    - 10053 / spm_tracing / SQL_Plan_Management
  - Statement level to capture global activity -- new connections only
    - alter system set events 'trace[rdbms.SQL_Optimizer.*][sql: 0a14b3yhux040]’
  - If using connection pools
    - dbms_sqldiag.dump_trace(…p_component=>'Compiler‘)

Troubleshooting SPM – trace alternatives

Global statement trace for already existing and new connections:

```
dbms_sqldiag.dump_trace(p_sql_id=>'0a14b3yhux040', p_child_number=>0,
                        p_component=>'Compiler',  p_file_id=>'trace_0a14b3yhux040');
```

SPM Trace at session level:

```
ALTER SESSION SET EVENTS 'trace[RDBMS.SQL_Plan_Management.*]';
```

Internal SPM trace generation (more detailed):

```
EXEC dbms_spm.configure('spm_tracing',1);
-- validate current settings:
SELECT * FROM sys.smb$config WHERE parameter_name='SPM_TRACING';
```

Classic optimizer trace, which includes SPM standard trace:

```
ALTER SESSION SET EVENTS 'trace[sql_optimizer.*]';
-- same but for in older Oracle versions
ALTER SESSION SET EVENTS='10053 trace name context forever, level 1';
```

Troubleshooting SPM – trace results

When a Plan Baseline is found and used:

```
SPM: statement found in SMB
SPM: cost-based plan found in the plan baseline, planId = 3736278975
SPM: cost-based plan successfully matched, planId = 3736278975
```

When no matching Plan Baseline found, adding new one as non-accepted:

```
SPM: statement found in SMB
SPM: setup to add new plan to existing plan baseline, sig = 1245871306398155660, planId =
2322719765
SPM: planId's of plan baseline are: 3736278975
SPM: using qksan to reproduce, cost and select accepted plan, sig = 1245871306398155660
SPM: failed to reproduce the plan using the following info:
SPM: generated non-matching plan:
SPM: couldn't reproduce any enabled+accepted plan so using the cost-based plan, planId =
2322719765
SPM: add new plan: sig = 1245871306398155660, planId = 2322719765
SPM: new plan added to existing plan baseline, sig = 1245871306398155660, planId = 2322719765
SPM: REPRODUCED status changed to NO: sig = 1245871306398155660, planName =
SQL_PLAN_12kjtb8qvdhwcdeb317bf
```

Usage decision

- optimizer_capture_sql_plan_baselines = TRUE

OR

- created when needed by each SQL?

No silver bullet, choose the better one for your environment

Use case:

- some queries from a specific program would benefit from parameter=value
  - instead of changing the parameter and avoid this functionality to be used on other cases which would be beneficial, SQL baselines are created just for those programs
  - It is more manual work to have a functionality available for the other users

Daily management overhead

Evaluate autotasks recommendations:

- Trusting automatic creation of SQL Profiles/Baselines?
- SQL Tuning Advisor (SQLTA) / SPM Evolve Advisor (12c)
- Several views and reports to explore their results using SQL
  - dba_advisor_tasks / dba_advisor_rationale

- IF SQLTA recommends a SQL profile for a SQL that has no baselines
  - create the new SQL profile
  - move it to a SQL baseline
  - Disable SQL profile

- On 11g: evaluate daily new baselines evolving them
  - How to deal with heavy activity and lots of new plans? AWR helps

Daily management example approach

- Better performing evolved plans should be testing with a variety of binds on test environment before accepting them
- Disable evaluated and confirmed non good plans

```
exec :r :=DBMS_SPM.ALTER_SQL_PLAN_BASELINE (sql_handle => 'SQL_114a395a2db6c38c',
                                             plan_name=>'SQL_PLAN_12kjtb8qvdhwca9f022c1',
                                             attribute_name =>'enabled', attribute_value =>'NO');

select count(*), count(distinct sql_handle) sqls, accepted, enabled
from dba_sql_plan_baselines
group by accepted, enabled;

  COUNT(*)   SQLS ACC ENA
---------- ------ --- ---
     13792  13788 YES YES              ← are in use by optimizer
        20     16 YES NO
      5588   3063 NO  YES
       210     25 NO  NO
```

Daily management example approach

- Better performing evolved plans should be testing with a variety of binds on test environment before accepting them
- Disable evaluated and confirmed non good plans

```
exec :r :=DBMS_SPM.ALTER_SQL_PLAN_BASELINE (sql_handle => 'SQL_114a395a2db6c38c',
                                            plan_name=>'SQL_PLAN_12kjtb8qvdhwca9f022c1',
                                            attribute_name =>'enabled', attribute_value =>'NO');

select count(*), count(distinct sql_handle) sqls, accepted, enabled
from dba_sql_plan_baselines
group by accepted, enabled;

  COUNT(*)   SQLS ACC ENA
---------- ------ --- ---
     13792  13788 YES YES             ← are in use by optimizer
        20     16 YES NO              ← used but manually disabled because of problems found
      5588   3063 NO  YES
       210     25 NO  NO
```

Daily management example approach

- Better performing evolved plans should be testing with a variety of binds on test environment before accepting them
- Disable evaluated and confirmed non good plans

```
exec :r :=DBMS_SPM.ALTER_SQL_PLAN_BASELINE (sql_handle => 'SQL_114a395a2db6c38c',
                                            plan_name=>'SQL_PLAN_12kjtb8qvdhwca9f022c1',
                                            attribute_name =>'enabled', attribute_value =>'NO');

select count(*), count(distinct sql_handle) sqls, accepted, enabled
from dba_sql_plan_baselines
group by accepted, enabled;

  COUNT(*)   SQLS ACC ENA
---------- ------ --- ---
     13792  13788 YES YES            ← are in use by optimizer
        20     16 YES NO             ← used but manually disabled because of problems found
      5588   3063 NO  YES            ← new plans that needs evaluation
       210     25 NO  NO
```

Daily management example approach

- Better performing evolved plans should be testing with a variety of binds on test environment before accepting them
- Disable evaluated and confirmed non good plans

```
exec :r :=DBMS_SPM.ALTER_SQL_PLAN_BASELINE (sql_handle => 'SQL_114a395a2db6c38c',
                                             plan_name=>'SQL_PLAN_12kjtb8qvdhwca9f022c1',
                                             attribute_name =>'enabled', attribute_value =>'NO');

select count(*), count(distinct sql_handle) sqls, accepted, enabled
from dba_sql_plan_baselines
group by accepted, enabled;

  COUNT(*)  SQLS ACC ENA
---------- ----- --- ---
     13792 13788 YES YES              ← are in use by optimizer
        20    16 YES NO               ← used but manually disabled because of problems found
      5588  3063 NO  YES              ← new plans that needs evaluation
       210    25 NO  NO               ← already evaluated new plans nod performing good
```

Daily management example approach

- Better performing evolved plans should be testing with a variety of binds on test environment before accepting them
- Disable evaluated and confirmed non good plans

```
exec :r :=DBMS_SPM.ALTER_SQL_PLAN_BASELINE (sql_handle => 'SQL_114a395a2db6c38c',
                                             plan_name=>'SQL_PLAN_12kjtb8qvdhwca9f022c1',
                                             attribute_name =>'enabled', attribute_value =>'NO');

select count(*), count(distinct sql_handle) sqls, accepted, enabled
from dba_sql_plan_baselines
group by accepted, enabled;

  COUNT(*)  SQLS ACC ENA
---------- ----- --- ---
     13792 13788 YES YES                ← are in use by optimizer
        20    16 YES NO                 ← used but manually disabled because of problems found
      5588  3063 NO  YES                ← new plans that needs evaluation
       210    25 NO  NO                 ← already evaluated new plans nod performing good
```

Will you read a report with 5588 evolutions? Need an improved approach

Choosing what to evolve on 11g using stats

Which statements are now running in more than 5s and have non accepted baselines?

```
select * from (
select s.inst_id, s.sql_id
        ,min(s.first_load_time)                                                  min_load_time
        ,sum(s.executions)                                                       eje
        ,round(sum(s.elapsed_time)/1000000/greatest(sum(s.executions),1),2) avg_sec
        ,round(sum(s.disk_reads)/greatest(sum(s.executions),1),2)            avg_reads
        ,round(sum(s.buffer_gets)/greatest(sum(s.executions),1),2)           avg_gets
        ,dense_rank() over (order by sum(s.elapsed_time)/greatest(sum(s.executions),1) desc) rank
from gv$sql s, DBA_SQL_PLAN_BASELINES b
where s.FORCE_MATCHING_SIGNATURE=b.signature
  and b.enabled='YES' and b.accepted='YES'
  and exists (select 1 from DBA_SQL_PLAN_BASELINES b2
              where b.sql_handle=b2.sql_handle and b2.enabled='YES' and b2.accepted='NO')
group by s.inst_id, s.sql_id
having sum(s.elapsed_time)/greatest(sum(s.executions),1)/1000000 > 5)
where rank < 50
order by rank desc;
```

Choosing what to evolve on 11g using stats

Output example:

```
INST_ID SQL_ID        MIN_LOAD_TIME              EJE    AVG_SEC  AVG_READS   AVG_GETS       RANK
---------- ------------- -------------------- ---------- ---------- ---------- ---------- ----------
         1 4baj8ju7nqpvt 2014-06-22/24:30:20          2       4,34       4872      23787         15
         2 7n8hbzpruj4nu 2014-06-22/07:03:04        327       4,42    2007,47   74787,87         14
         2 t0jxh8b7b4at2 2014-20-22/04:02:48          2       4,44        223       2427         13
         3 7n8hbzpruj4nu 2014-06-22/07:47:28        569       7,28    2342,87   78480,22         12
         1 t22bnbazvj87x 2014-06-22/20:03:42         38       8,23     2348,8     3444,2         11
         3 t2qx4bv4y383b 2014-06-22/23:38:32         28       8,38    4207,32   47883,72         10
         2 arpa78rrkhp80 2014-06-22/08:27:28        240       8,42     2242,8    4747,24          9
         2 t7qntjn4aaj8k 2014-06-22/22:48:27         22       8,47    2787,33    4780,83          8
         2 04h4pq4tpn238 2014-06-22/08:32:08         32       8,78    2207,83   20427,44          7
         3 bqtx4q2jas08u 2014-06-22/02:43:22         34      20,33    2737,28     3444,2          6
         3 t2qx4bv4y383b 2014-06-22/22:24:02         22      20,88     4780,4   47282,78          5
         2 7qbkbt7h00wtw 2014-06-22/07:30:37        404      23,42    2228,87   73488,07          4
         2 0s2b777vta78b 2014-06-22/07:28:02        283       24,8    2377,83  273342,87          3
         1 7qbkbt7h00wtw 2014-06-08/04:04:27        647      27,43    2788,34   80848,38          2
         2 2rnttntn3xb4r 2014-06-22/23:48:28         22      32,77    7273,77   42244,87          1

15 rows selected.
```

Needs another join to show baseline plan_name. That idea next with AWR

Choosing what to evolve on 11g using AWR

Non accepted baselines of statements with executions times max/min variations of 5 times

- AWR has historical data, needs Diagnostic Pack license

Output example:

```
SQL_HANDLE            SQL_ID           MAXE  MINE PLANS #accept PLAN_NAME                      CREATED           ACC FIX
--------------------- --------------- ------ ----- ----- ------- ------------------------------ ----------------- --- ---
SQL_x1c4f31970638583  y283zg7xww7am    4,23   ,31     7       1 SQL_PLAN_s7554kd1hzk11258e3b6f 08/07/14 14:41:23 NO  NO
                                       4,23   ,31     7       1 SQL_PLAN_s7554kd1hzk116e35c426 06/07/14 11:48:34 NO  NO
                                       4,23   ,31     7       1 SQL_PLAN_s7554kd1hzk118d733646 21/07/14 14:06:03 NO  NO
                                       4,23   ,31     7       1 SQL_PLAN_s7554kd1hzk11900895ab 21/07/14 17:04:24 NO  NO

SQL_d8ee9cd8489c1506  d70w2c9n04bky   11,02    ,7     5       2 SQL_PLAN_djvnwv149s50p0de525df 17/06/14 17:50:45 NO  NO

SQL_50563b1a5e7c3515  cdxqhv810sguq  142,12  4,51    12       4 SQL_PLAN_62t9v39g7sdj6e14b0faa 17/06/14 09:37:12 NO  NO

SQL_a69a285b67c3ab33  7jr0hhnrg5qpd    5,12   ,54     4       0 SQL_PLAN_5sw7au49da15rb55d4723 18/06/14 14:09:54 NO  NO
                                       5,12   ,54     4       0 SQL_PLAN_5sw7au49da15r1224e506 06/06/14 13:21:12 NO  NO
                                       5,12   ,54     4       0 SQL_PLAN_5sw7au49da15r92eca281 12/06/14 11:50:16 NO  NO
```

Choosing what to evolve on 11g using AWR

```
with sqlh as (
  select FORCE_MATCHING_SIGNATURE, sql_id, count(1) plans, max(elap_xeje) maxe, min(elap_xeje) mine
  from (select h.FORCE_MATCHING_SIGNATURE, sql_id, plan_hash_value plan_hv
              ,round(sum(h.elapsed_time_delta)/sum(h.executions_delta)/1000000,2) elap_xeje
      from DBA_HIST_SQLSTAT h
      where snap_id >=(select min(snap_id) from dba_hist_snapshot where begin_interval_time>=trunc(sysdate)-30)
        and dbid=(select dbid from v$database) and instance_number in (1,2,3)
        and h.executions_delta > 0
        and h.FORCE_MATCHING_SIGNATURE in (
              select b.signature from DBA_SQL_PLAN_BASELINES b
              where b.enabled='YES' and b.accepted='NO')
      group by FORCE_MATCHING_SIGNATURE, sql_id, plan_hash_value)
  group by FORCE_MATCHING_SIGNATURE, sql_id
  having  max(elap_xeje)/(case min(elap_xeje) when 0 then 1 else min(elap_xeje) end) > 5)
select sql_handle, signature, sqlh.sql_id, sqlh.maxe, sqlh.mine, sqlh.plans,
       (select count(1) from DBA_SQL_PLAN_BASELINES b2
        where b.sql_handle=b2.sql_handle and b2.accepted='YES' and b2.enabled='YES') "#accept"
       ,PLAN_NAME, created, accepted, fixed
from DBA_SQL_PLAN_BASELINES b, sqlh
where b.enabled='YES' and b.accepted='NO' and b.created > trunc(sysdate)-30
  and b.signature = sqlh.FORCE_MATCHING_SIGNATURE
order by sql_handle, sql_id, maxe;
```

Example of bug found – 11.2.0.2

```
SQL_ID  c2408ff99yj8u, child number 0
-------------------------------------
SELECT DATA, COUNT(*) FROM BIG_TABLE WHERE ((((((((DATA LIKE :1 ) OR (DATA LIKE :2 )) OR
(DATA LIKE :3 )) OR (DATA LIKE :4 )) OR (DATA LIKE :5 )) OR (DATA LIKE :6 )) OR (DATA LIKE :7 )) OR
(DATA LIKE :8 )) AND (STATE IN (:9 , :10 , :11 , :12 , :13 , :14 , :15 , :16 , :17 , :18 , :19 , :20 , :21
, :22 , :23 , :24 , :25 , :26 , :27 , :28 , :29 , :30 , :31 , :32 ))) GROUP BY DATA

Plan hash value: 1854823252
-----------------------------------------------------------------------------------------------
| Id  | Operation            | Name                      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                           |       |       |  7504 (100)|          |
|   1 |  HASH GROUP BY       |                           |    16 |  1536 |            |          |
|   2 |   CONCATENATION      |                           |       |       |            |          |
|*  3 |    INDEX RANGE SCAN| BIG_TABLE_CIDN            |  1668 |  156K|   938   (1)| 00:00:12 |
..
|* 10 |    INDEX RANGE SCAN| BIG_TABLE_CIDN            |  1165 |  109K|   938   (1)| 00:00:12 |
-----------------------------------------------------------------------------------------------
...
Note
   - SQL plan baseline SQL_PLAN_8vdc1f1da6d7sa0mtur14 used for this statement
```

Example of bug found – 11.2.0.2

```
-- Plan baselines for this statement:

col created for a25
select PLAN_NAME, ENABLED, accepted, fixed, created, executions, optimizer_cost
from DBA_SQL_PLAN_BASELINES
where sql_handle='SQL_114a395a2db6c38c' and enabled='YES'
order by created;

PLAN_NAME                      ENA ACC FIX CREATED                   EXECUTIONS OPTIMIZER_COST
------------------------------ --- --- --- ------------------------- ---------- --------------
SQL_PLAN_8vdc1f1da6d7sa0mtur14 YES YES NO  05/08/14 16:11:26,000000           1           7504
SQL_PLAN_0sa1muctuur8ve7e49070 YES NO  NO  05/08/14 17:29:45,000000           0           2039

2 rows selected.


-- Want to evaluate the new plan found, not accepted yet
```

Example of bug found – 11.2.0.2

```
-- evolving the new plan:
var r clob;
set long 10000
exec :r := dbms_spm.evolve_sql_plan_baseline(sql_handle => 'SQL_114a395a2db6c38c', \
                                 plan_name=>'SQL_PLAN_0sa1muctuur8ve7e49070', commit => 'NO');
print;
-------------------------------------------------------------------------------
                        Evolve SQL Plan Baseline Report
-------------------------------------------------------------------------------
Inputs:
-------
  SQL_HANDLE = SQL_114a395a2db6c38c
  PLAN_NAME  = SQL_PLAN_0sa1muctuur8ve7e49070
  TIME_LIMIT = DBMS_SPM.AUTO_LIMIT
  VERIFY     = YES
  COMMIT     = NO

Plan: SQL_PLAN_0sa1muctuur8ve7e49070
------------------------------------
  Plan was verified: Time used ,507 seconds.
  Error encountered during plan verification (ORA-1008).
    ORA-01008: no todas las variables han sido enlazadas
```

Bug 14007103 discarded
Bug 19326647 opened

Pythian
love your data

Summary review

- SPM enabled by default, but not capture
- No extra license, available only on Enterprise edition
- Baseline need to be **accepted** AND **enabled to** be used
- SQL needs to run more than once to be considered
- Baselines are global - not per schema
- Evolve advisor (autotask) in 12c, manual evolution in 11g.
- Don’t trust evolution results, test it!
- SQL Trace when in doubt (statement level!)
- Needs a good understanding to troubleshoot

Questions?

calero@pythian.com

@ncalerouy

http://www.linkedin.com/in/ncalero

References

- SPM in Oracle 11g (Nov 2010): http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-sql-plan-management-11gr2-133099.pdf
- SPM in Oracle 12c (Jun 2013): http://www.oracle.com/technetwork/database/bi-datawarehousing/twp-sql-plan-mgmt-12c-1963237.pdf
- Oracle 12c Tuning Guide - Managing SQL Plan Baselines: http://docs.oracle.com/database/121/TGSQL/tgsql_spm.htm#TGSQL94621
- Oracle 12c Tuning Guide - Query Optimizer: http://docs.oracle.com/database/121/TGSQL/tgsql_optcncpt.htm#TGSQL192
- Oracle 11c Tuning Guide - Query Optimizer: http://docs.oracle.com/cd/B28359_01/server.111/b28274/optimops.htm
- Oracle Optimizer development team Blog: https://blogs.oracle.com/optimizer