Oracle SQL Plan Management (SPM) your imperative performance tool

Gary Gordhamer

New York Oracle Users Group

May 2022

Gary Gordhamer

Managing Principal Consultant
Viscosity North America

31 years of IT experience
30 years with Oracle (6.x up to 21c)
Worked in many different industries including healthcare, manufacturing, utilities, banking, and marketing

@ggordham
linkedin.com/in/ggordhamer/
oraontap.blogspot.com

Quest IOUG Database & Technology Community

ORACLE ACE

Note: The views in this presentation are my own and do represent the views of the company I work for.

VISCOSITY NORTH AMERICA ORACLE Platinum Partner Microsoft Gold Partner

Developer Resource Center /
Oracle ACE Program

Viscosity's Oracle ACES

Oracle ACE Program

The Oracle ACE Program recognizes and rewards community members for their technical contributions in the Oracle community.

Charles Kim,
CEO & Co-Founder

Twitter: @racdba

ORACLE ACE Director

Rich Niemiec,
Chief Innovation Officer

Twitter: @richniemiec

ORACLE ACE Director

Craig Shallahamer
Applied AI Scientist

Twitter: @orapub

ORACLE ACE Director

Sean Scott,
Consultant

Twitter:
@oraclesean

ORACLE ACE

Gary Gordhamer,
Consultant

Twitter: @ggordham

ORACLE ACE

Julio Ayapan,
Consultant

ORACLE ACE Associate

VISCOSITY NORTH AMERICA ORACLE Platinum Partner

Microsoft Gold Partner

VISCOSITY
NORTH AMERICA

Oracle License Management
Get the most out of your Oracle investment

ZERO DOWNTIME
Migrations

Apps
SaaS/PaaS,
Mobility,
Application
Development

Professional Services
Where you need it most

Performance Health Checks
How's it running?

Staff Aug
Workforce Capacity
on Demand

DBA Services
Remote and
On-site

On-Call Support
Managed Services

VISCOSITY NORTH AMERICA ORACLE Platinum Partner Microsoft Gold Partner

All Viscosity hosted **webinar recordings** and **conference session slide decks** will be posted to www.OraPub.com for **free** and paid members!

VISCOSITY NORTH AMERICA

Upgrading to Oracle
Database 19c Performance Tuning

Pocket Solutions Guide for Oracle Database 19c

Book by: **Gary Gordhamer**
in collaboration with **Ronald Mehrer**

Will be realeased in May 2022

Want to try this out?

You can download all the scripts from GitHub
https://github.com/ggordham/ora-presentations/tree/main/spm-NYOUG

Or: https://bit.ly/3yEFjBL `bit.ly / 3yEFjBL`

All scripts are "as is" and meant for teaching purposes only.
DO NOT run these in any production system or system relied upon by your development teams.

All SPM steps have been set to run as the created PERFLAB user

So, you have a bad SQL statement?

Options to change the performance of a bad SQL:

- Look at environment setup (like statistics / missing index)
- Change the SQL / add a HINT
- Apply a SQL PROFILE
- Use a SQL PLAN BASELINE
- Create and apply a SQL PATCH

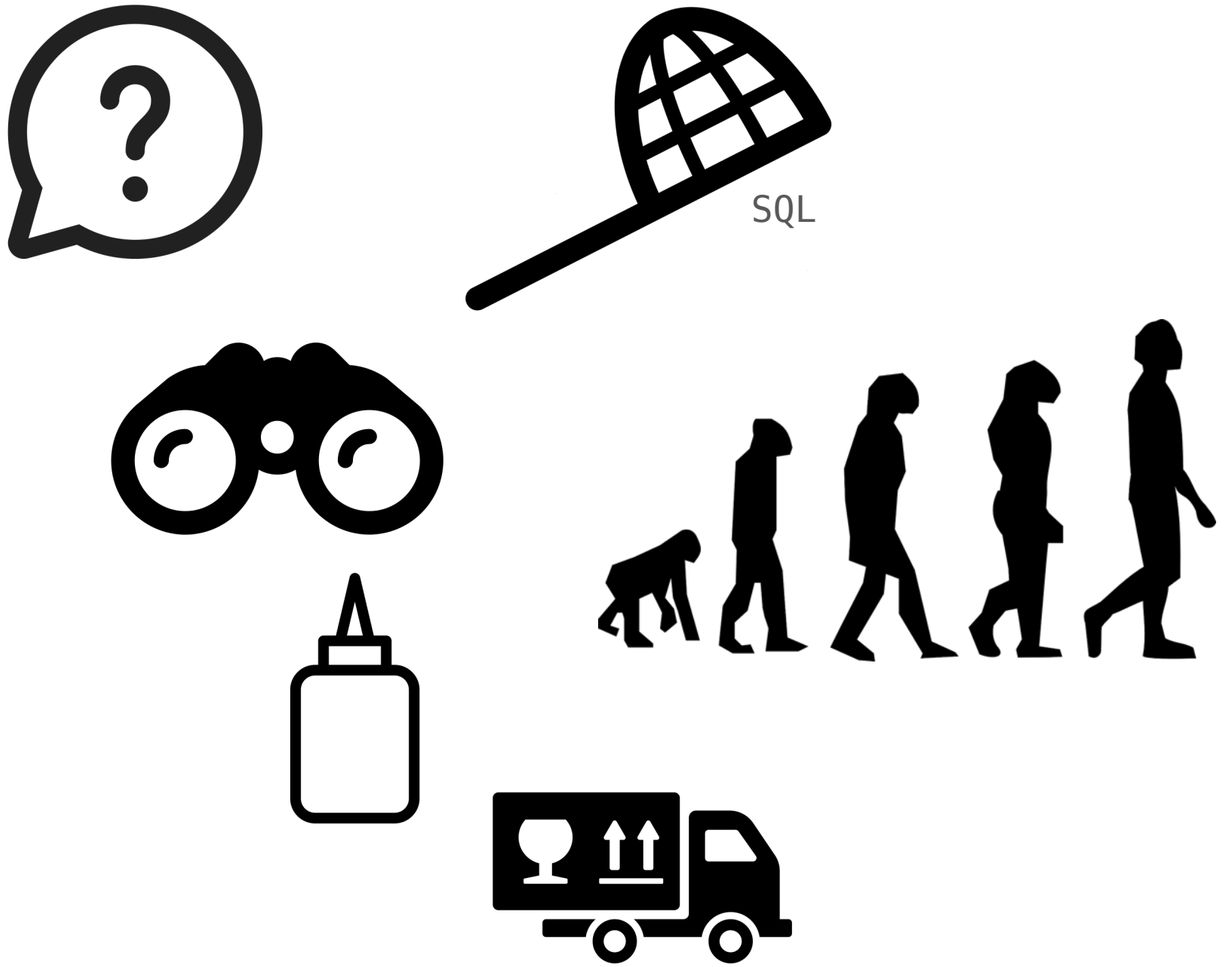
Step 1 in this process is to identify how a SQL statement is performing

Step 2 what is a better performance option for that SQL statement

Step 3 apply a fix from the list above

Agenda

- What is SPM
- Capturing plans
- Viewing baseline information
- Evolving baselines
- Moving baselines
- Fixing baselines



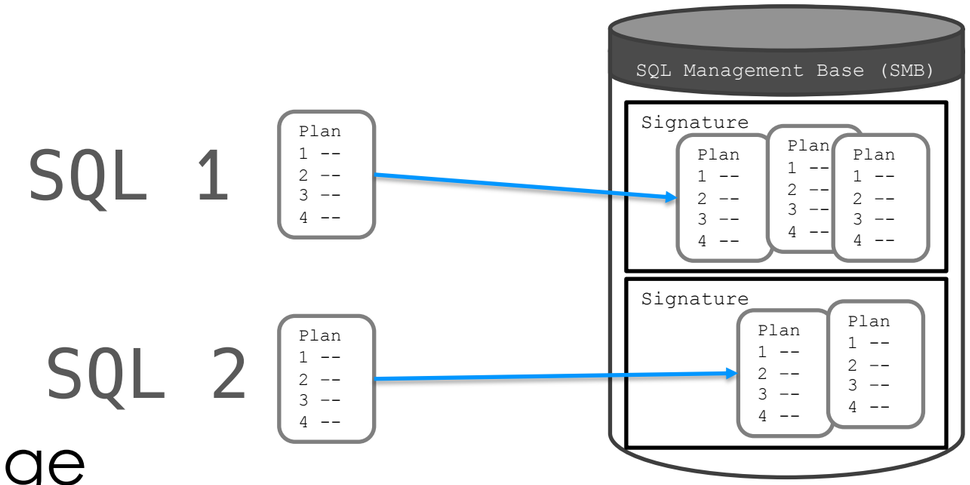
What is SPM?

SQL Plan Management (SPM)

First appeared in Oracle 11.1

- Replaced OUTLINES
- Allows for more control on how the optimizer chooses execution plans

SQL Execution plans are captured and stored in SQL Management Base (SMB) part of the SYSAUX tablespace



Plans are "Accepted" to prefer their usage

Creates "PLAN STABILITY"

SQL Plan Management (SPM) - Basics

Baselines contain information about:

- SQL Text
- Bind Variables
- Environment when SQL ran

Available in Standard and Enterprise Edition

- In SE you can have only one plan / baseline per SQL
- In EE you can have more than one plan / baseline per SQL
- Also, in EE baselines can evolve

Exadata includes Automatic SQL Plan Management in 19c +

Basic SPM Process

1. Capture baselines (plans) for SQL statement(s)
2. Accept (allow or prefer usage of) baseline plans
3. Evolve (add or accept additional plans)
4. ”Fix” or anchor a baseline

You can also:

- Migrate baselines from one database to another
  - Good for testing / code migration
- Migrate stored outlines to baselines (recommended)

SQL Plan Management

Plan Capture

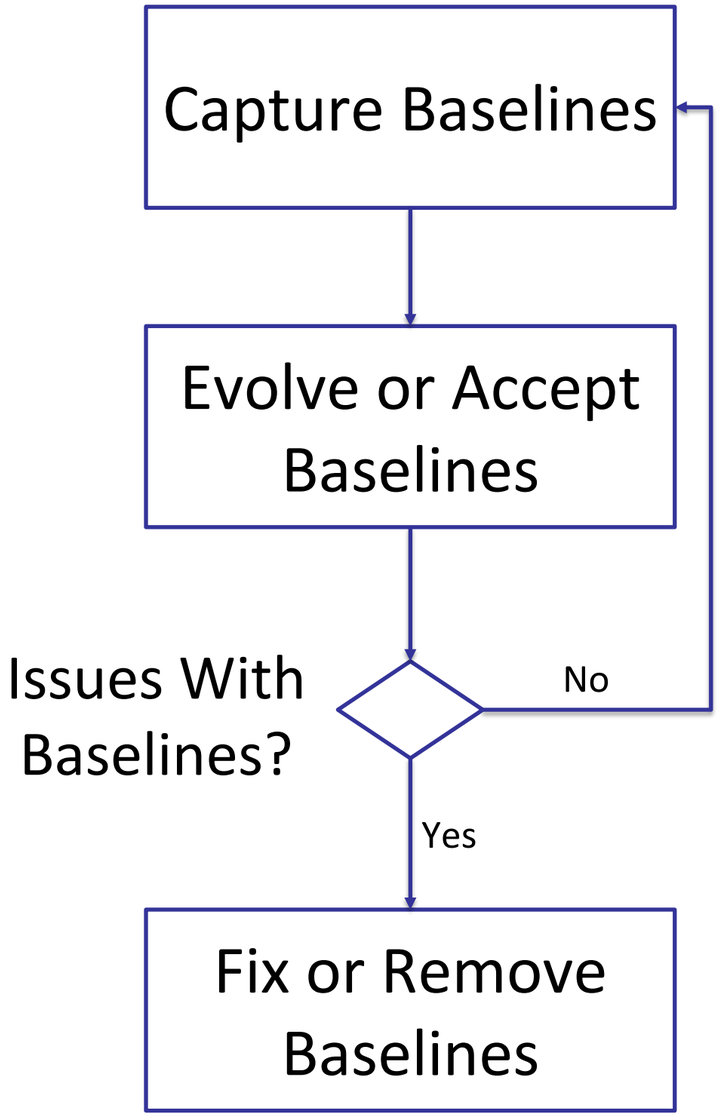
- Automatic
  `optimizer_capture_sql_plan_baselines`
- Manually
  `DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE()`

Plan Selection

- The Optimizer uses Accepted Baselines

Plan Evolution

- Accept or Reject new Baselines



Profiles vs. Baselines

The Optimizer uses object and system stats + bind variables to generate execution plans

- SQL Profile influences the optimizer plan creation
- SQL Baseline forces the optimizer to choose only accepted plans

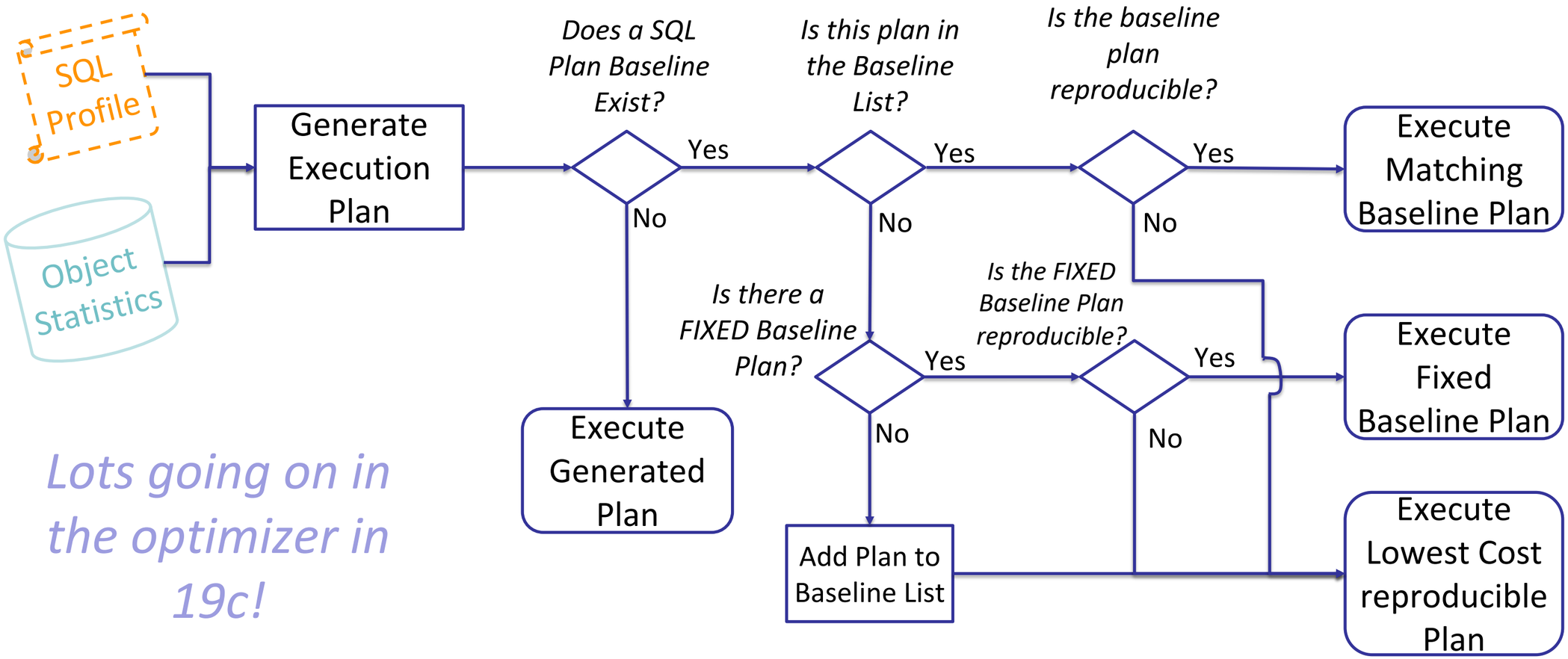
Profiles and Baselines can be used together

Profiles can influence multiple SQL statements

Baselines force a specific plan and only affect single statements

Baselines can contain multiple plans, and plans can evolve

Optimizer Decision Tree



Lots going on in the optimizer in 19c!

Notes:

Most SPM features are available in OEM
- Showing commands in slide deck
- Good for scripting / automating
- Can manually do most things in OEM

Cost of a SQL Plan is not absolute
- Different bind variable values create different costs
- Same execution plan / different costs

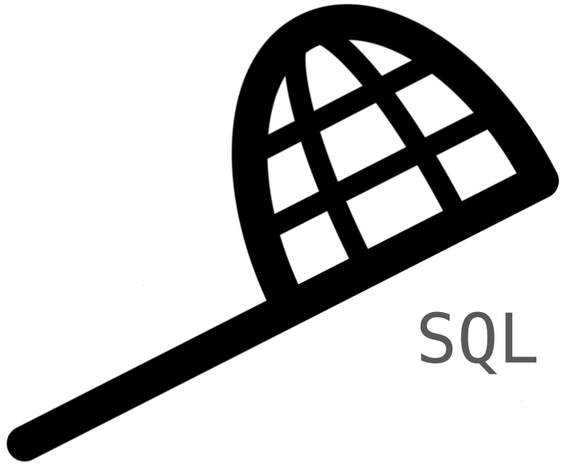
The Optimizer is not always right!

Example – Optimizer Estimate vs Reality

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1			553 (100)	1	00:00:00.04	1978	1976
1	SORT AGGREGATE		1	1	22		1	00:00:00.04	1978	1976
2	NESTED LOOPS		1	1	22	553 (1)	1	00:00:00.04	1978	1976
3	NESTED LOOPS		1	1	22	553 (1)	1	00:00:00.04	1977	1975
* 4	TABLE ACCESS FULL	T1	1	1	13	550 (1)	1	00:00:00.04	1974	1972
* 5	INDEX RANGE SCAN	T2I	1	1		2 (0)	1	00:00:00.01	3	3
6	TABLE ACCESS BY INDEX ROWID	T2	1	1	9	3 (0)	1	00:00:00.01	1	1

Id	Operation	Name	Starts	E-Rows	E-Bytes	Cost (%CPU)	A-Rows	A-Time	Buffers	Reads
0	SELECT STATEMENT		1			553 (100)	1	00:00:01.00	11089	4536
1	SORT AGGREGATE		1	1	22		1	00:00:01.00	11089	4536
2	NESTED LOOPS		1	1	22	553 (1)	250K	00:00:00.93	11089	4536
3	NESTED LOOPS		1	1	22	553 (1)	250K	00:00:00.36	9118	2566
* 4	TABLE ACCESS FULL	T1	1	1	13	550 (1)	250K	00:00:00.05	1976	1971
* 5	INDEX RANGE SCAN	T2I	250K	1		2 (0)	250K	00:00:00.34	7142	595
6	TABLE ACCESS BY INDEX ROWID	T2	250K	1	9	3 (0)	250K	00:00:00.45	1971	1970

Capturing Baselines



Capturing Baselines

You can Automatically or Manually capture baselines

- Automatically
  - Database parameter
  - Filter by Schema or SQL Text
  - Set at Session level
- Manually
  - Load from Cursor Cache
  - Load from AWR
  - Load from SQL Set

Automatic Capture

Set database parameter

```
OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES = TRUE
```

Can be set at database level, or session level

Configure capture using DBMS_SPM package:

```
EXEC DBMS_SPM.CONFIGURE(
            parameter_name => 'AUTO_CAPTURE_SQL_TEXT',
            parameter_value => '%TEST_ONLY%',
            allow => false);
```

Excludes all SQL that contains text "TEST_ONLY" in upper case

Note: FALSE in this syntax is to exclude,
to include only SQL containing the text "TEST_ONLY" then set TRUE.

Automatic Capture

Configure capture using DBMS_SPM package:

Includes all SQL where parsing schema is SH

Note: TRUE in this syntax is to include

```
EXEC DBMS_SPM.CONFIGURE(
              parameter_name => 'AUTO_CAPTURE_PARSING_SCHEMA_NAME',
              parameter_value => 'sh', allow => TRUE);
```

To view the current capture filter:

```
SELECT PARAMETER_NAME, PARAMETER_VALUE
   FROM DBA_SQL_MANAGEMENT_CONFIG
   WHERE PARAMETER_NAME LIKE '%AUTO%';
```

Automatic Capture Why no Baselines?

Parameters than can be adjusted:

```
PARAMETER_NAME                          PARAMETER_VALUE
------------------------------ ------------------------------
AUTO_CAPTURE_ACTION
AUTO_CAPTURE_MODULE
AUTO_CAPTURE_PARSING_SCHEMA_NAME
AUTO_CAPTURE_SQL_TEXT                  (sql_text LIKE %TEST_ONLY%)
AUTO_SPM_EVOLVE_TASK                   OFF
AUTO_SPM_EVOLVE_TASK_INTERVAL           3600
AUTO_SPM_EVOLVE_TASK_MAX_RUNTIME        1800
```

I'm not seeing plans being captured or the ones I want?

- Needs to be executed more than twice (repeatable)
- Data dictionary SQL statement are not captured

Manual Capture

Multiple options in DBMS_SPM package to capture plans

- LOAD_PLANS_FROM_CURSOR_CACHE
- LOAD_PLANS_FROM_AWR
- LOAD_PLANS_FROM_SQLSET

Each has a slightly different syntax

Note: All captured plans are accepted by default!

Plans added after capture are not accepted by default!

```
SQL_HANDLE           PLAN_NAME                      ORIGIN         ENA ACC FIX OPTIMIZER_COST
-------------------- ------------------------------ -------------- --- --- --- --------------
SQL_37d2d1608ca0c6f5 SQL_PLAN_3gnqjc26a1jrpceb2ac1e AUTO-CAPTURE   YES YES NO             553
SQL_37d2d1608ca0c6f5 SQL_PLAN_3gnqjc26a1jrpc6a45b88 AUTO-CAPTURE   YES NO  NO            1891
```

Manual capture example – Cursor Cache

Capturing a single plan from cursor cache

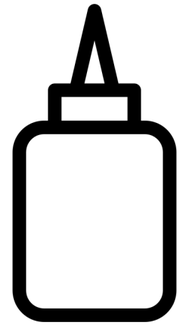
- Specify a SQL_ID and a PLAN_HASH_VALUE (PHV)
- If you exclude PHV, then all plans for the given SQL are loaded

```
VARIABLE v_plan_cnt NUMBER

EXECUTE :v_plan_cnt :=
   DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(
     sql_id => '27m0sdw9snw59',
     plan_hash_value => 3534348942);
```

```
Returns the number of plans loaded
```

Why fix (marked as preferred)

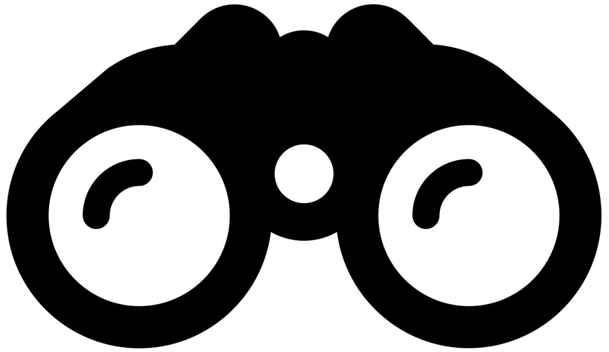


New plans will be collected once a baseline is in place

- Accepted and Enabled baseline plans will try to be used
- Fixed plans will be preferred over non-fixed
- If a SQL has a fixed plan, new baselines will NOT be collected
- Evolution of baselines with fixed plans will not happen
- More than one plan can be fixed

```
DBMS_SPM.ALTER_SQL_PLAN_BASELINE (
      SQL_HANDLE => 'SQL_28d363dc39b4d314',
      PLAN_NAME =>  'SQL_PLAN_2jnv3vhwv9nsnc6a45b88',
      ATTRIBUTE_NAME => 'fixed',
      ATTRIBUTE_VALUE => 'YES');
  DBMS_OUTPUT.PUT_LINE( 'dropped ' || v_dplans || ' plans');
```

Viewing Baseline Information



Viewing Baseline Information

Lots of information available:

```
SELECT b.plan_name, b.sql_handle, b.creator, b.origin,
       TO_CHAR(b.created, 'YYYYMMDD HH24:MI:SS') created,
       TO_CHAR(b.last_executed, 'YYYYMMDD HH24:MI:SS') baseline_executed,
       TO_CHAR(b.last_verified, 'YYYYMMDD HH24:MI:SS') baseline_verified,
       b.enabled, b.accepted, b.fixed, b.optimizer_cost,
       b.elapsed_time, b.cpu_time, b.buffer_gets, b.disk_reads,
       b.direct_writes, b.rows_processed, b.fetches
  FROM dba_sql_plan_baselines b;
```

PLAN_NAME	SQL_HANDLE	CREATOR	ORIGIN	CREATED	BASELINE_EXECUTED
SQL_PLAN_ct5006ahu8hpbc6a45b88	SQL_cc940032a1a442ab	SYS	EVOLVE-LOAD-FROM-CURSOR-CACHE	20201211 22:01:59	(null)
SQL_PLAN_ct5006ahu8hpbceb2ac1e	SQL_cc940032a1a442ab	SYS	MANUAL-LOAD-FROM-CURSOR-CACHE	20201211 15:50:39	20201211 16:05:14

BASELINE_VERIFIED	ENABLED	ACCEPTED	FIXED	OPTIMIZER_COST	ELAPSED_TIME	CPU_TIME	BUFFER_GETS	DISK_READS	DIRECT_WRITES	ROWS_PROCESSED	FETCHES
20201211 22:02:01	YES	YES	NO	1893	22109547	373104	3952	0	0	1	1
(null)	YES	YES	YES	553	1137796	615734	13070	1	0	2	2

What does all this mean?

- Enabled – Can this baseline plan be used?
- Accepted – Is the baseline plan accepted for use?
- Fixed – Is the baseline anchored?
- Performance data: CPU, Buffer Gets, Disk Reads, etc.
  - From V$SQL area at the time the baseline was created
- Timestamps: Modified, Executed, Verified
  - Based on the Baseline usage / creation

There are many more columns to look at!

Is a Baseline Being Used?

The v$sql view also has a column to show if a baseline is in use.

```
SELECT sql_id, plan_hash_value, sql_plan_baseline, exact_matching_signature
 FROM v$sql
 WHERE sql_id = '0ptw8zskuh9r4';
```

SQL_ID	PLAN_HASH_VALUE	SQL_PLAN_BASELINE	EXACT_MATCHING_SIGNATURE
0ptw8zskuh9r4	906334482	SQL_PLAN_2jnv3vhwv9nsnc6a45b88	2941804779115172628

You can also rename the baseline, this example uses SQL ID

PLAN_NAME	SQL_HANDLE
SQL_ID_0ptw8zskuh9r4_NAME_2jnv3vhwv9nsnc6a45b88	SQL_28d363dc39b4d314

Note: statements will match by SIGNITURE not SQL ID
Signatures are based on bind variable usage!

Example Baseline Plan Rename:

Use the ALTER_SQL_PLAN_BASELINE function:

```
SET SERVEROUTPUT ON
DECLARE
  v_dplans number;
BEGIN
  v_dplans :=
    DBMS_SPM.ALTER_SQL_PLAN_BASELINE(
       sql_handle => 'SQL_28d363dc39b4d314',
       plan_name => 'SQL_PLAN_2jnv3vhwv9nsnc6a45b88',
       attribute_name => 'plan_name',
       attribute_value => 'SQL_ID_0ptw8zskuh9r4_NAME_2jnv3vhwv9nsnc6a45b88');
  DBMS_OUTPUT.PUT_LINE( 'updated ' || v_dplans || ' plans');
END;
/
```

Note: returns number of plans updated

How do I map a baseline to a SQL ID?

Baselines have a “Plan Name” and ”SQL Handle”

- SQL_PLAN_ct5006ahu8hpbceb2ac1e
- SQL_cc940032a1a442ab

How do I map that to what is happening in real time (`v$sql`)?

- SQL_TEXT – this is the full text of the SQL. Can be hard to match, and watch out for bind variables syntax
- Signature – depending on your `CURSOR_SHARING` DB setting, maps to `v$sql` column:
  - exact_matching_signature
  - force_matching_signature

`V$SQL` also has a `SQL_PLAN_BASELINE` column, if baseline used

Viewing a Plan from a Baseline

There's a function for that!

We join to the v$sql table if we are looking up by sql_id

Note: only works if the SQL is in the cursor cache!

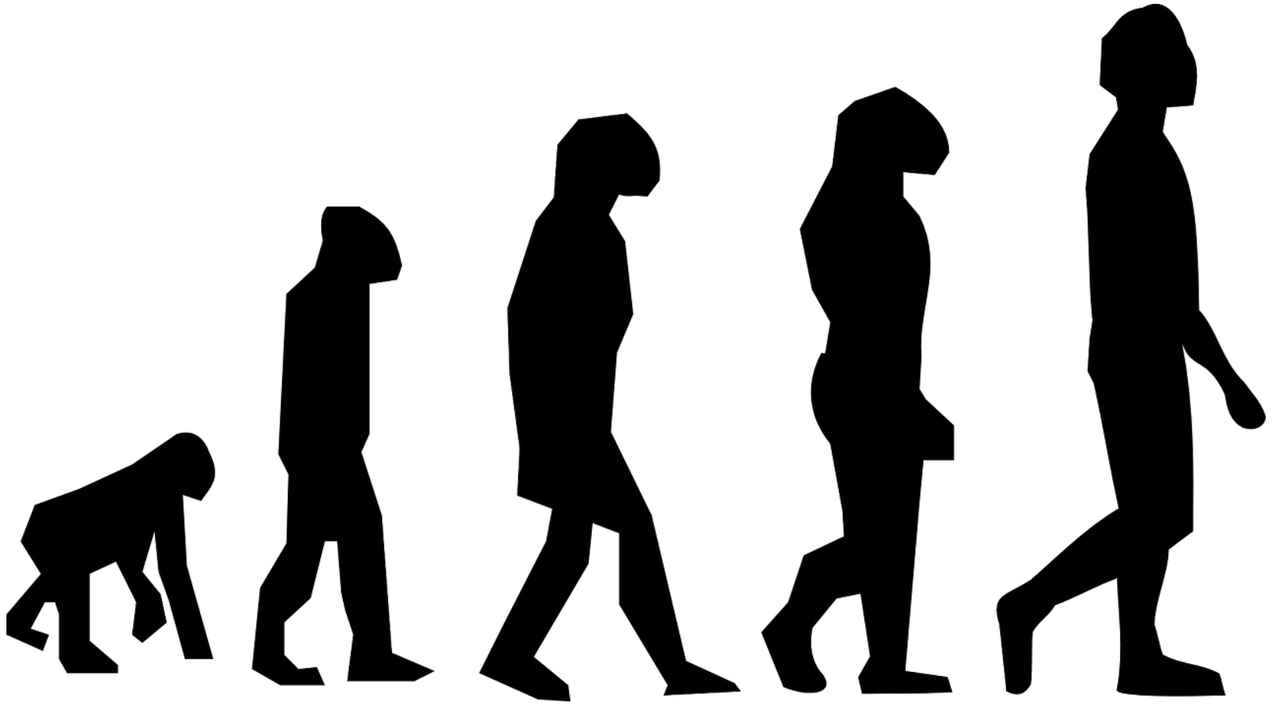
```
SELECT PLAN_TABLE_OUTPUT
FROM   V$SQL s, DBA_SQL_PLAN_BASELINES b,
       TABLE(
     DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE(b.sql_handle,b.plan_name,'typical')
       ) t
WHERE  s.EXACT_MATCHING_SIGNATURE=b.SIGNATURE
AND    b.PLAN_NAME=s.SQL_PLAN_BASELINE
AND    s.SQL_ID='0ptw8zskuh9r4';
```

Note this example uses exact_matching_signiture

Viewing a Plan from a Baseline

```
--------------------------------------------------------------------------------
SQL handle: SQL_28d363dc39b4d314
SQL text: select /*+ NO_ADAPTIVE_PLAN */ sum(t1.c), sum(t2.c) from   t1, t2
          where  t1.a = t2.a and    t1.d = 10
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Plan name: SQL_PLAN_2jnv3vhwv9nsnc6a45b88         Plan id: 3332660104
Enabled: YES     Fixed: NO      Accepted: YES     Origin: EVOLVE-LOAD-FROM-
CURSOR-CACHE
Plan rows: From dictionary
--------------------------------------------------------------------------------
Plan hash value: 3332660104
-----------------------------------------------------------------------------
| Id  | Operation            | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      |     1 |    22 |  1101   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE      |      |     1 |    22 |            |          |
|*  2 |   HASH JOIN          |      |     2 |    44 |  1101   (1)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| T1   |     2 |    26 |   550   (1)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| T2   |   500K|  4394K|   549   (1)| 00:00:01 |
-----------------------------------------------------------------------------
```

Evolving Baselines



Evolving Baselines

Baseline plans need to be accepted to be used

- Manually accept
- Automated task can accept “verified” plans

Optimizer will capture new plans in the SMB

- For baselines that are not fixed
- For new SQL if auto capture is turned on

Running Evolve Manually

You can run the Evolve process manually

```
BEGIN
  :v_rep := DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE(
    sql_handle => 'SQL_37d2d1608ca0c6f5',
    time_limit => DBMS_SPM.AUTO_LIMIT,
    verify => 'YES',
    commit => 'YES');
END;
/
```

This statement returns a CLOB with the report from the evolve task

Note we asked for new plans to be accepted (commit set to YES)

Partial Evolve Report

```
GENERAL INFORMATION SECTION
-------------------------------------------
 Task Information:
 -------------------------------------------
 Task Name            : TASK_19359
 Execution Type       : SPM EVOLVE
 Scope                : COMPREHENSIVE
 Status               : COMPLETED
 Number of Errors     : 0
-------------------------------------------
SUMMARY SECTION
-------------------------------------------
  Number of plans processed  : 1
  Number of findings         : 2
  Number of recommendations  : 1
  Number of errors           : 0
-------------------------------------------
Findings (2):
-----------------------------
 1. The plan was verified in 2.52400 seconds. It passed the benefit criterion
    because its verified performance was 2.80913 times better than that of the
    baseline plan.
 2. The plan was automatically accepted.

Recommendation:
-----------------------------
 Consider accepting the plan.
```

Running Evolve Manually

Baseline Status changes

```
SQL_HANDLE            PLAN_NAME                          ORIGIN         ENA ACC FIX OPTIMIZER_COST
--------------------- ---------------------------------- -------------- --- --- --- --------------
SQL_37d2d1608ca0c6f5  SQL_PLAN_3gnqjc26a1jrpceb2ac1e     AUTO-CAPTURE   YES YES NO              553
SQL_37d2d1608ca0c6f5  SQL_PLAN_3gnqjc26a1jrpc6a45b88     AUTO-CAPTURE   YES NO  NO             1891
```

After the evolve task accepts the new plan

```
SQL_HANDLE            PLAN_NAME                          ORIGIN         ENA ACC FIX OPTIMIZER_COST
--------------------- ---------------------------------- -------------- --- --- --- --------------
SQL_37d2d1608ca0c6f5  SQL_PLAN_3gnqjc26a1jrpceb2ac1e     AUTO-CAPTURE   YES YES NO              553
SQL_37d2d1608ca0c6f5  SQL_PLAN_3gnqjc26a1jrpc6a45b88     AUTO-CAPTURE   YES YES NO             1891
```

Creating an Evolve Task Looking at AWR

```
VAR rep CLOB
SET LONG 10000000
COLUMN report FORMAT A200

DECLARE
   tname varchar2(1000);
   ename varchar2(1000);
   n number;
BEGIN
 tname := DBMS_SPM.CREATE_EVOLVE_TASK(sql_handle =>'SQL_28d363dc39b4d314');
 DBMS_SPM.SET_EVOLVE_TASK_PARAMETER( task_name => tname,
                                     parameter => 'ALTERNATE_PLAN_BASELINE',
                                     value     => 'EXISTING');
 DBMS_SPM.SET_EVOLVE_TASK_PARAMETER( task_name => tname,
                                     parameter => 'ALTERNATE_PLAN_SOURCE',
                                     value     => 'CURSOR_CACHE+AUTOMATIC_WORKLOAD_REPOSITORY');
 DBMS_SPM.SET_EVOLVE_TASK_PARAMETER( task_name => tname,
                                     parameter => 'ALTERNATE_PLAN_LIMIT',
                                     value     => 'UNLIMITED');
   ename := DBMS_SPM.EXECUTE_EVOLVE_TASK(tname);

  -- returns number of plans accepted
  n := DBMS_SPM.IMPLEMENT_EVOLVE_TASK(tname);

   select DBMS_SPM.REPORT_EVOLVE_TASK(task_name => tname) into :rep from dual;
END;
/
SELECT :rep AS report FROM DUAL;
```

Evolve Task Options

ALTERNATE_PLAN_SOURCE – list of locations to pull new plans by separated by “+” sign

- CURSOR_CACHE
- AUTOMATIC_WORKLOAD_REPOSITORY
- SQL_TUNING_SETS

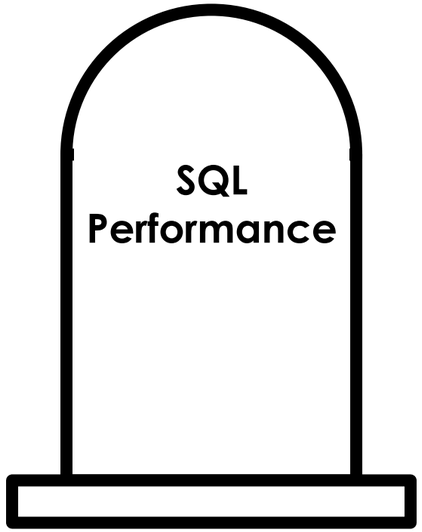
ALTERNATE_PLAN_BASELINE -

- EXISTING – only load plans for statements that have baselines
- EXISTING+NEW – Also create new baselines
- AUTO – Let Oracle decide

TIME_LIMIT – Time in seconds to spend for the task

ALTERNATE_PLAN_LIMIT – Maximum number of plans to load (not per SQL statement)

Final Thoughts



Baseline Flowchart – After Parsing / Plan Generation

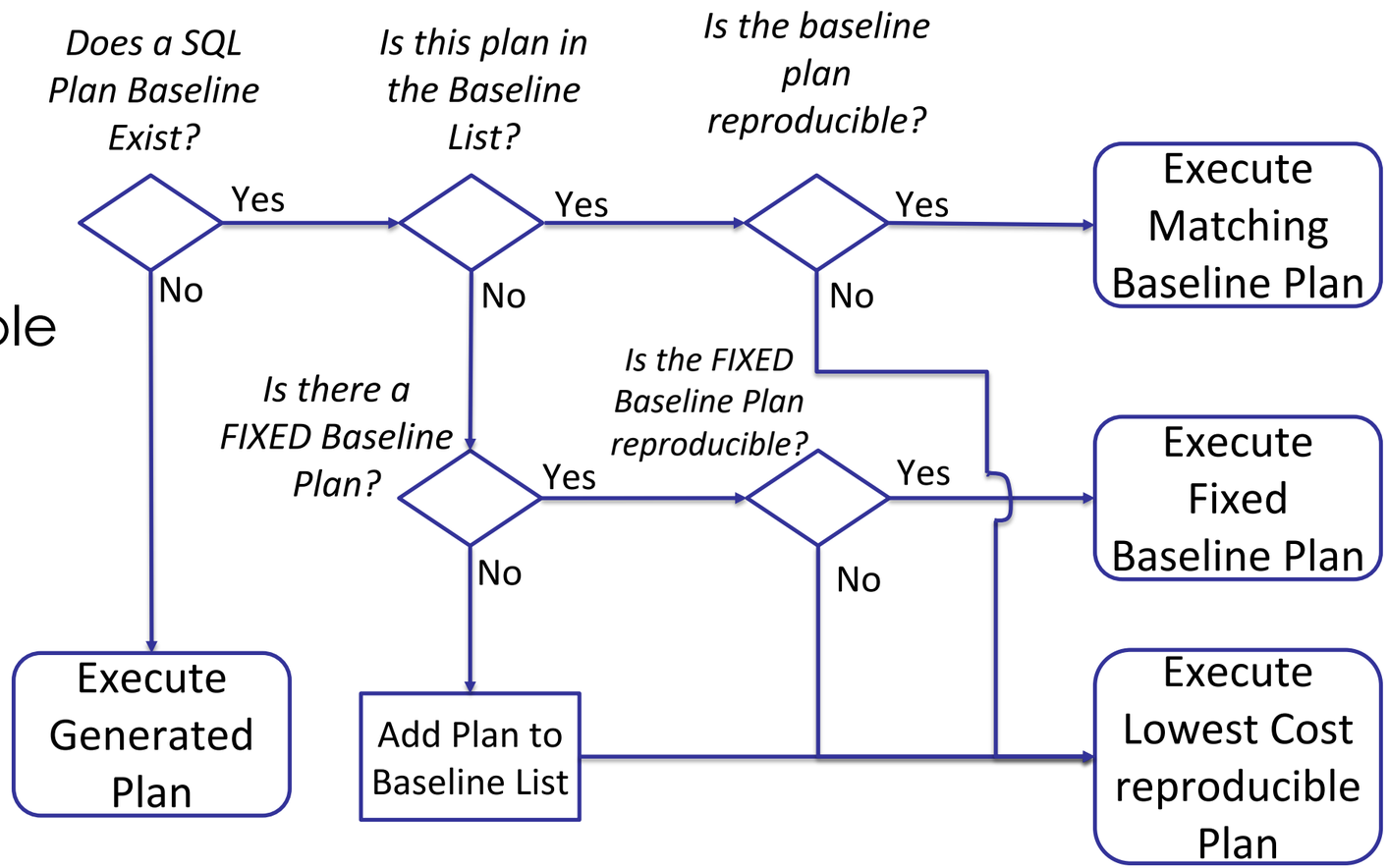
No Baseline – Optimizer chooses

Baseline – not fixed

- Capture new plans
- Try to use lowest cost repeatable and accepted baseline

Baseline – Fixed

- Try to use fixed baseline



Baselines – additional Notes

Baselines do not “guarantee” a specific plan will always be used

Once a baseline exists, new plans will continue to be captured
If a SQL has a fixed baseline, then no new plans will be captured

The plan must be reproducible to be used

- E.G. an index has been dropped or corrupted
- The plan can not be reproduced so it can not be used

SQL Management Base - Notes

By default, SMB uses10% of the SYSAUX tablespace

You can configure this:

```
DBMS_SPM.CONFIGURE('space_budget_percent',30);
```

A weekly purging task run in the maintenance window:

Plans not used for more than 53 weeks are purged

You can also configure this:

```
DBMS_SPM.CONFIGURE('plan_retention_weeks',105);
```

More thing to think about

Baselines are stored Execution Plans

- Specific set of steps to retrieve data
- Tied to object names / structures

Plans need to change:

- New partitions added to a table / new local index partitions
- Data distribution changes

Oracle database optimizer is designed to adjust to how your database changes.

An execution plan is one extremely specific way to do something.

Couple of Ways to Use Baselines

1. Manually capture a specific set of Baselines and only accept plans you want to use. Fix the plans if helpful.
2. Auto capture Baselines for a specific user / schema (reporting?). Curate the baseline set (drop some, accept some, etc.). Let the optimizer discover new plans and evolve them.
3. Hybrid – manually capture some baselines and fix them. Capture (auto or manual) other baselines and allow them to evolve.

Your Millage May Vary!

References

- How to Use SQL Plan Management (SPM) - Plan Stability Worked Example (Doc ID 456518.1)
- SQL Plan Baseline Not always created (Doc ID 788853.1)
- How To Use DBMS_XPLAN.COMPARE_PLANS & Hint Usage Report To Troubleshoot Plan Reproducibility Issues (SPM Baselines & SQL Profiles) From 19c (Doc ID 2736319.1)
- FAQ: SQL Plan Management (SPM) Frequently Asked Questions (Doc ID 1524658.1)
- Transporting SQL PLAN Baselines from One Database to Another. (Doc ID 880485.1)
- Loading Hinted Execution Plans into SQL Plan Baseline. (Doc ID 787692.1)
- Master Note: Plan Stability Features (Including SQL Plan Management (SPM)) (Doc ID 1359841.1)
- How to Drop Plans from the SQL Plan Management (SPM) Repository (Doc ID 790039.1)
- How to Load SQL Plans into SQL Plan Management (SPM) from the Automatic Workload Repository (AWR) (Doc ID 789888.1)
- SRDC - How to Collect Standard Information for an Issue Where a SQL Plan Management (SPM) Baseline is Not Used (Doc ID 1644732.1)

Appendix

Manual capture example – AWR Report

Capturing from AWR or SQL Tuning Set (STS) use a filter syntax

- For AWR specific snap ID's
- For STS specify the STS name
- Filter syntax is similar for both

```
VARIABLE v_plan_cnt NUMBER

EXEC :v_plan_cnt :=
   DBMS_SPM.LOAD_PLANS_FROM_AWR(
      begin_snap => 3188,
      end_snap => 3189,
      basic_filter => 'parsing_schema_name = ''HR''');
```

Note the double quotes on the filter!

Manual capture example – STS

Here we capture from a STS using a different filter syntax

- Filter syntax detailed in the manual

```
VARIABLE v_plan_cnt NUMBER

DBMS_SPM.LOAD_PLANS_FROM _SQLSET( -
     sqlset_name  => 'SPM_STS', -
     basic_filter => 'sql_text like ''SELECT /* MY_SQL */%''');
```

Note the double quotes on the filter!

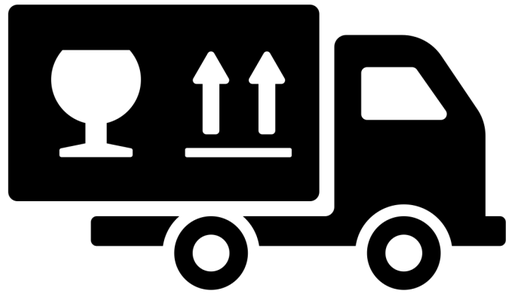
Fixing a Baselines

Use the ALTER_SQL_PLAN_BASELINE function

```
SET SERVEROUTPUT ON
DECLARE
  v_dplans number;
BEGIN
  v_dplans :=
    DBMS_SPM.ALTER_SQL_PLAN_BASELINE (
      SQL_HANDLE => 'SQL_28d363dc39b4d314',
      PLAN_NAME =>  'SQL_PLAN_2jnv3vhwv9nsnc6a45b88',
      ATTRIBUTE_NAME => 'fixed',
      ATTRIBUTE_VALUE => 'YES');
  DBMS_OUTPUT.PUT_LINE( 'dropped ' || v_dplans || ' plans');
END;
```

Note: this fixes a single execution plan in the baseline

Moving Baselines



Why Move a Baseline?

- Copy from production to non-production
- Or from non-production to production

- Backup your baselines

- Maybe you are a vendor, and you want to ship baselines with your application product

- See more details about the baseline

Moving Baselines

1. Create staging table

```
exec DBMS_SPM.CREATE_STGTAB_BASELINE('STGTAB', user);
```

2. Pack baselines into the staging table

```
DBMS_SPM.PACK_STGTAB_BASELINE('STGTAB', user)
```

3. Export the staging table (data pump)
4. Import staging table (data pump)
5. Unpack staging table

```
DBMS_SPM.UNPACK_STGTAB_BASELINE('STGTAB', user)
```

There are options to filter baselines by SQL_ID and PLAN_NAME that are packed or unpacked

Removing a Baselines

Use the DROP_SQL_PLAN_BASELINE function

```
SET SERVEROUTPUT ON
DECLARE
  v_dplans number;
BEGIN
  v_dplans :=
    DBMS_SPM.DROP_SQL_PLAN_BASELINE (
      SQL_HANDLE => 'SQL_28d363dc39b4d314',
      PLAN_NAME =>  'SQL_PLAN_2jnv3vhwv9nsnc6a45b88');
  DBMS_OUTPUT.PUT_LINE( 'dropped ' || v_dplans || ' plans');
END;
/
```

Note: this drops a single execution plan in the baseline

Checking SMB Retention Length

Default retention is 53 weeks

```
SELECT parameter_name, parameter_value
  FROM  dba_sql_management_config
  WHERE parameter_name = 'PLAN_RETENTION_WEEKS';
```

Weekly scheduled task in the maintenance window will purge plans from the SMB that have a `LAST_EXECUTED` timestamp greater than the `PLAN_RETENTION_WEEKS`

You can set the retention from 5 to 523 weeks.

```
DBMS_SPM.CONFIGURE('plan_retention_weeks',105);
```

Follow Us Online!

Facebook.com/ViscosityNA

Linkedin.com/company/Viscosity-North-America

@ViscosityNA

Viscosity North America

@Viscosity_NA

VISCOSITY

NORTH AMERICA