Execution Plan Stability in Oracle11g

Jože Senegačnik
private researcher
joze.senegacnik@dbprof.com

About the Speaker

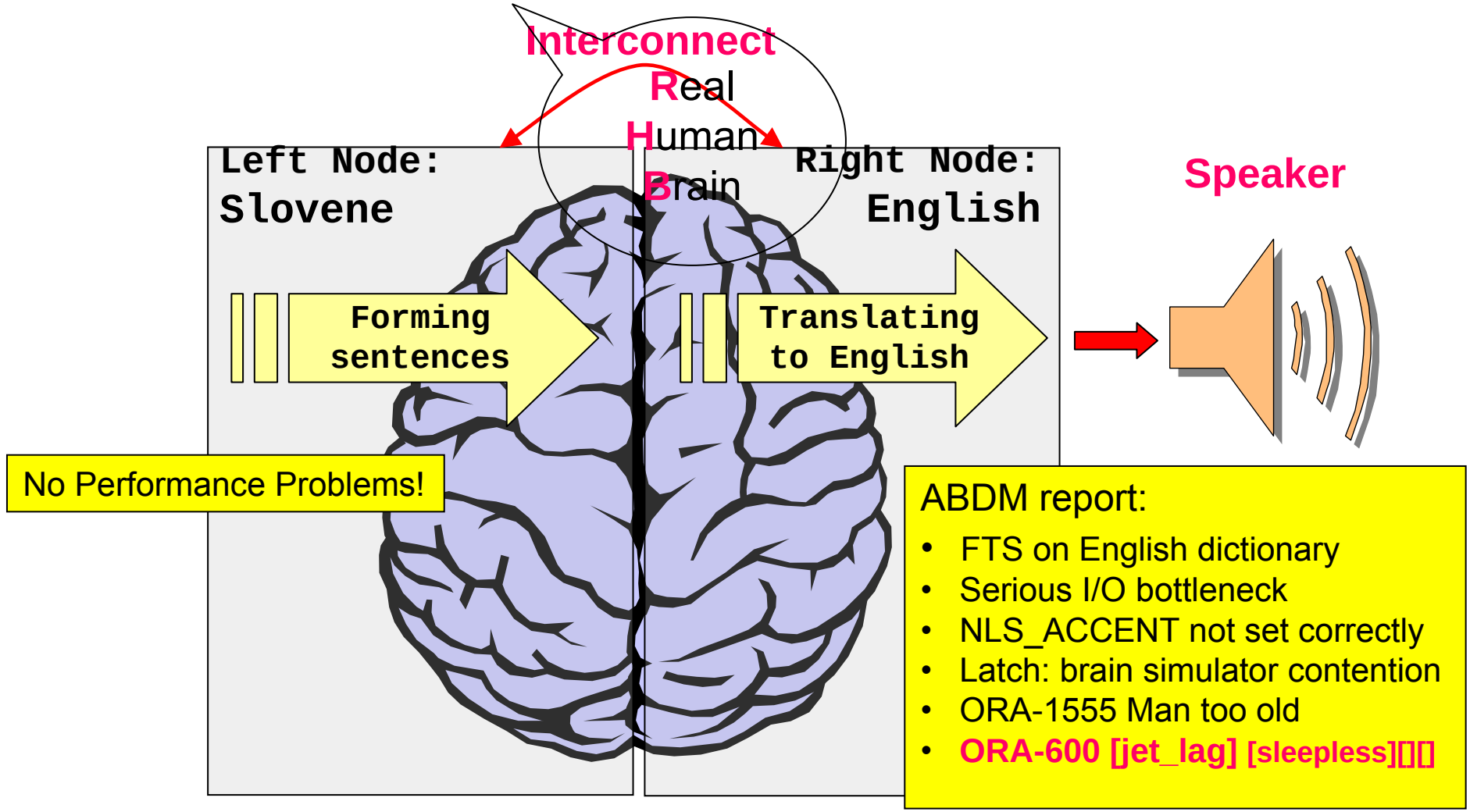
- Jože Senegačnik:
  - Registered private researcher
  - First experience with Oracle Version 4 in 1988
  - 21+ years of experience with Oracle RDBMS.
  - Member of the OakTable Network www.oaktable.net
  - Vice president of the board of SIOUG
  - Owner of www.dbprof.com
  - CISA

Five.

SIOUG SLOVENSKO DRUŠTVO UPORABNIKOV PROGRAMSKE OPREME ORACLE

DbProf
Database Performance Management & Professional Services

Using 2-node RHB Cluster For This Presentation



SQL Statement Optimization Goals

- RULE based optimization – used in past
  - However, still many use it (even Oracle in 10g).
  - Stable execution plans – but still dependent on some changes in data dictionary
- COST based optimization – a.k.a STATISTICAL optimization
  - used in Oracle since version 7
  - getting better in each new version
  - highly dependent on gathered statistical data

Reasons For Changed Execution Plan

- The execution plan of a SQL statement may change due to changed:
  - Optimizer statistics and optimizer parameters
  - Schema and metadata definitions
    - new or dropped index ...
  - System parameter settings
  - New version of the optimizer
    - upgraded Oracle version, patches,...
  - SQL profile used
    - adopted through Automatic SQL Tuning

The Optimization Goal

- The ultimate goal should be:
  - An execution plan should only change when it will result in performance gain.
  - Bad plans should never be used!

- How we can achieve that?

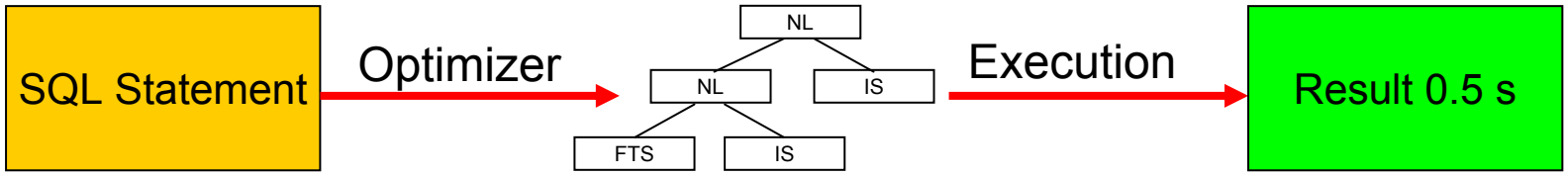
What is SQL Plan Management (SPM)?

- SQL plan management is a preventative mechanism that records and evaluates the execution plans of SQL statements over time, and builds SQL plan baselines composed of a set of existing plans known to be efficient.
- The SQL plan baselines are then used to preserve performance of corresponding SQL statements, regardless of changes occurring in the system.

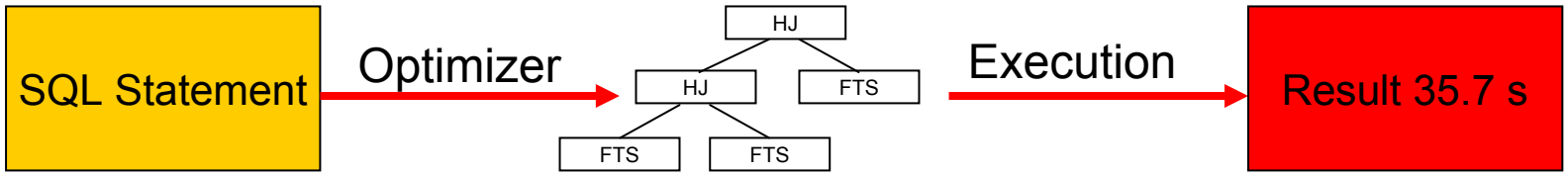
From Oracle® Database Performance Tuning Guide 11g Release 1 – Chapter 15 - Using SQL Plan Management

- Changed Optimizer behavior:
  - Only known and **verified** plans are used. All plan changes are automatically verified and only comparable or better plans are used

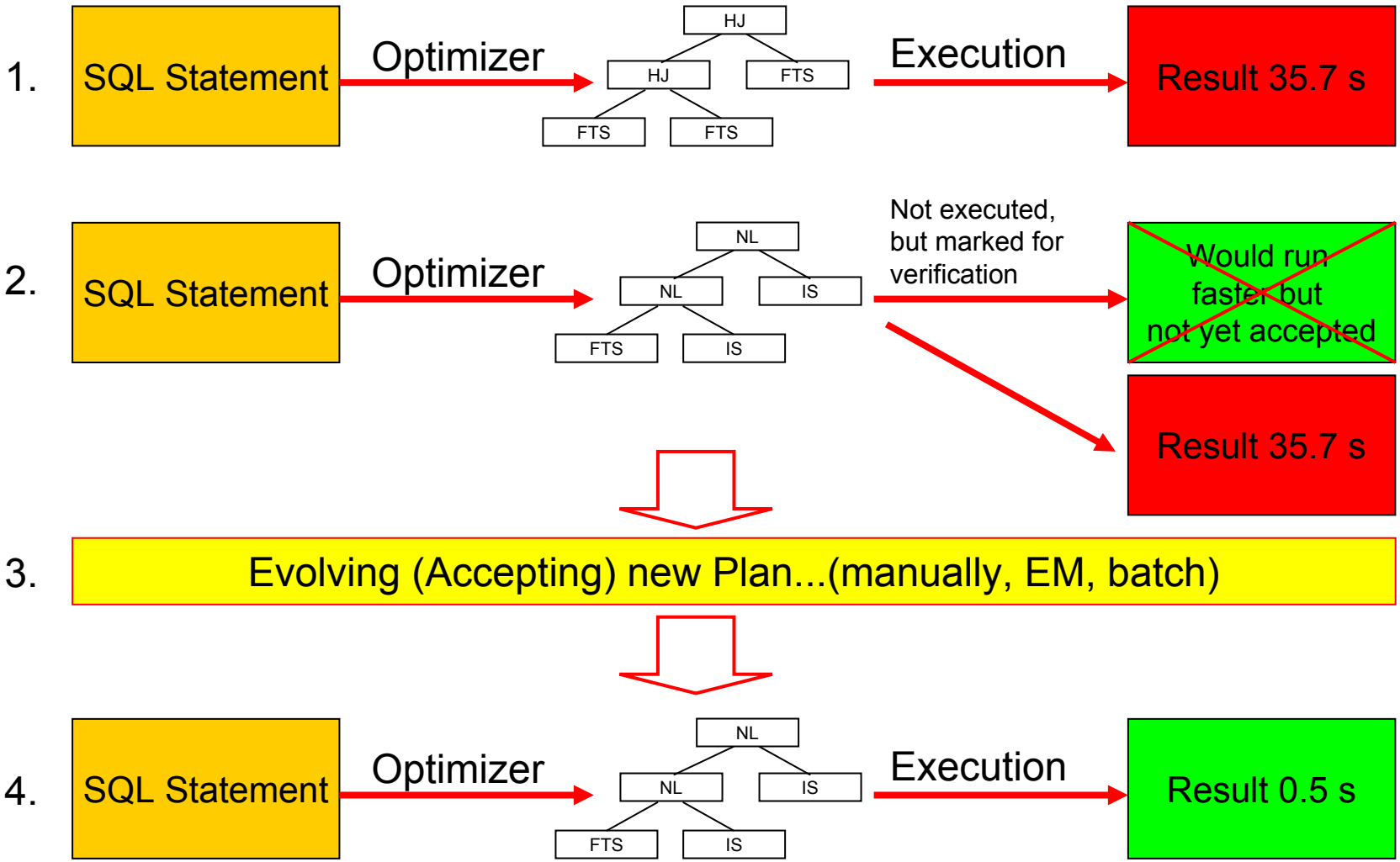
Scenario Without Plan Management



Something changes ....



Scenario With SQL Plan Management



First Execution of SQL Statement

```
SQL> set autotrace on expl
SQL> alter system set optimizer_capture_sql_plan_baselines=true;
System altered.

SQL> select * from t where id=3;
        ID PAD
---------- ---
         3 XXX

Execution Plan
----------------------------------------------------------
-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |   254 |    25   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| T    |     1 |   254 |    25   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("ID"=3)
```

Now We Re-execute The Same Statement

```
SQL> select * from  t  where  id=3;

        ID PAD
---------- ---
         3 XXX

Execution Plan
----------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |   254 |    25   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| T    |     1 |   254 |    25   (0)| 00:00:01 |
----------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
1 - filter("ID"=3)
Note
-----
   - SQL plan baseline "SYS_SQL_PLAN_904f9eda94ecae5c" used for this statement
```

New Index Is Added

```
SQL> create index t_i1 on t (id);
Index created.

SQL> select * from  t  where  id=3;

        ID PAD
---------- ---
         3 XXX

Execution Plan
----------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |   254 |    25   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| T    |     1 |   254 |    25   (0)| 00:00:01 |
----------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("ID"=3)
Note
-----
   - SQL plan baseline "SYS_SQL_PLAN_904f9eda94ecae5c" used for this statement
```

optimizer_use_sql_plan_baselines=false

```
SQL> alter session set optimizer_use_sql_plan_baselines=false;

SQL> select * from  t  where  id=3;

        ID PAD
---------- ---
         3 XXX

Execution Plan
----------------------------------------------------------------------------------------
| Id  | Operation                    | Name | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |     1 |   254 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| T    |     1 |   254 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | T_I1 |     1 |       |     1   (0)| 00:00:01 |
----------------------------------------------------------------------------------------

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

   2 - access("ID"=3)
```

DBA_SQL_PLAN_BASELINES

```
SQL> select sql_handle,plan_name,enabled,accepted
  2  from DBA_SQL_PLAN_BASELINES
  3  where sql_text like '%id=3%';

SQL_HANDLE -> SQL_BASELINE     PLAN_NAME                      ENA ACC
------------------------------ ------------------------------ --- ---
SYS_SQL_8a54f32d904f9eda        SYS_SQL_PLAN_904f9eda10f5b979  YES NO
SYS_SQL_8a54f32d904f9eda        SYS_SQL_PLAN_904f9eda94ecae5c  YES YES
```

- DBA_SQL_PLAN_BASELINE contains information about all baselines in the database
- The SQL handle is a unique identifier for each SQL statement used in SPM for managing plan history.

Evolving New Plan Added

```
SQL> SET SERVEROUTPUT ON
SQL> SET LONG 10000
SQL> DECLARE
  2      report clob;
  3  BEGIN
  4      report := DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE(
                   plan_name=>'SYS_SQL_PLAN_904f9eda94ecae5c');
  5      DBMS_OUTPUT.PUT_LINE(report);
  6  END;
  7  /
```

NOTES:

- PLAN_NAME is obtained from explain plan generated by autotrace or dbms_xplan package
- Another way is to specify SQL_HANDLE which is obtainable from column SQL_PLAN_BASELINE in V$SQL

Evolving New Plan Report

```
                  Evolve SQL Plan Baseline Report
--------------------------------------------------------------------------------
Inputs:
-------
  SQL_HANDLE = SYS_SQL_8a54f32d904f9eda
  PLAN_NAME  =
  TIME_LIMIT = DBMS_SPM.AUTO_LIMIT
  VERIFY     = YES
  COMMIT     = YES

Plan: SYS_SQL_PLAN_904f9eda10f5b979
-----------------------------------
  Plan was verified: Time used ,062 seconds.
  Passed performance criterion: Compound improvement ratio >= 28.
  Plan was changed to an accepted plan.
                       Baseline Plan      Test Plan     Improv. Ratio
                       -------------      ---------     -------------
  Execution Status:        COMPLETE       COMPLETE
  Rows Processed:                 1              1
  Elapsed Time(ms):               0              0
  CPU Time(ms):                   0              0
  Buffer Gets:                   84              3              28
  Disk Reads:                     0              0
  Direct Writes:                  0              0
  Fetches:                        0              0
  Executions:                     1              1
--------------------------------------------------------------------------------
Report Summary
--------------------------------------------------------------------------------
Number of SQL plan baselines verified: 1.
Number of SQL plan baselines evolved: 1.
```

Re-executing Uses Verified Execution Plan

```
SQL> alter session set optimizer_use_sql_plan_baselines=true;

Session altered.

SQL> select * from  t  where  id=3;

Execution Plan
----------------------------------------------------------------------------------------
| Id  | Operation                    | Name | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |      |     1 |   254 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| T    |     1 |   254 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | T_I1 |     1 |       |     1   (0)| 00:00:01 |
----------------------------------------------------------------------------------------

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

   2 - access("ID"=3)

Note
-----
   - SQL plan baseline "SYS_SQL_PLAN_904f9eda10f5b979" used for this statement
```

After Dropping an Index

```
SQL> select * from  t  where  id=3;

Execution Plan
----------------------------------------------------------------------------------------
| Id  | Operation                   | Name | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |      |     1 |   254 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| T    |     1 |   254 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | T_I1 |     1 |       |     1   (0)| 00:00:01 |
----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("ID"=3)

Note
-----
   - SQL plan baseline "SYS_SQL_PLAN_904f9eda10f5b979" used for this statement

SQL> drop index t_i1;
SQL> select * from  t  where  id=3;

Execution Plan
----------------------------------------------------------
Plan hash value: 1601196873

-------------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |   254 |    25   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| T    |     1 |   254 |    25   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("ID"=3)
```

SQL Plan Management Parameters

- SQL Plan Management is controlled by two init.ora parameter
  - **optimizer_capture_sql_plan_baselines**
    - Controls auto-capture of SQL plan baselines for repeatable statements
    - Set to false by default in 11gR1 (system and session modifiable)
  - **optimizer_use_sql_plan_baselines**
    - Controls the use of existing SQL plan baselines by the optimizer
    - Set to true by default in 11gR1 (system and session modifiable)
- SQL Plan Baselines are visible in view **DBA_SQL_PLAN_BASELINE**
- Package DBMS_SPM is used in background for managing SQL Plans

Displaying SQL Plan Baselines

- To view the plans stored in the SQL plan baseline for a given statement, use the DISPLAY_SQL_PLAN_BASELINE function of the DBMS_XPLAN package:

```
select * from table(
    dbms_xplan.display_sql_plan_baseline(
                  sql_handle=>'SYS_SQL_8a54f32d904f9eda',
                  format=>'basic')
 );
```

dbms_xplan.display_sql_plan_baseline Output

```
--------------------------------------------------------------------------------
SQL handle: SYS_SQL_8a54f32d904f9eda
SQL text: select * from  t  where  id=3
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
Plan name: SYS_SQL_PLAN_904f9eda10f5b979
Enabled: YES     Fixed: NO      Accepted: YES     Origin: AUTO-CAPTURE
--------------------------------------------------------------------------------

Plan hash value: 1601196873

-----------------------------------
| Id  | Operation          | Name |
-----------------------------------
|   0 | SELECT STATEMENT   |      |
|   1 |  TABLE ACCESS FULL | T    |
-----------------------------------
--------------------------------------------------------------------------------
Plan name: SYS_SQL_PLAN_904f9eda94ecae5c
Enabled: YES     Fixed: NO      Accepted: NO      Origin: AUTO-CAPTURE
--------------------------------------------------------------------------------

Plan hash value: 1601196873

-----------------------------------
| Id  | Operation          | Name |
-----------------------------------
|   0 | SELECT STATEMENT   |      |
|   1 |  TABLE ACCESS FULL | T    |
-----------------------------------
```

Loading SQL Baselines

- Manually loading plans
  - from the cursor cache or
  - from a SQL tuning set.
  - exporting from one database and importing into another
  - automatically for every statement executed (optimizer_capture_sql_plan_baselines=TRUE)

- When you manually load plans into a SQL plan baseline, these loaded plans are added as accepted plans.

DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE

```
SQL> select sql_id
     from v$sql
     where sql_text=
'select count(distinct owner) from t3 where object_id between :1 and
   :2';
```

```
SQL_ID
-------------
0dbcn62qczucf
```

```
SQL> DECLARE
  2  plan pls_integer;
  3  BEGIN
  4    plan := DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(
  5      sql_id => '0dbcn62qczucf' );
  6  END;
  7  /
```

DBMS_SPM.LOAD_PLANS_FROM_SQLSET

```
DECLARE
  plans pls_integer;
BEGIN plans := DBMS_SPM.LOAD_PLANS_FROM_SQLSET(
                 sqlset_name => ‘joc');
END;
/
```

or

```
DECLARE
  plans pls_integer;
BEGIN plans :=
   DBMS_SPM.LOAD_PLANS_FROM_SQLSET(
    sqlset_name => ‘joc‘
    basic_filter => 'sql_text like ''select%data_n%''' );
END;
/
```

Plan Capturing for SQLs with Stored Outlines

- If an outline is created and use_stored_outlines=TRUE automatic plan capture will not work for that SQL statement.

- Workaround:
  1. Prepare tuning set through capture_cursor_cache_sqlset() in DBMS_SQLTUNE package.
  2. Create baseline from tuning set.
  3. Disable using stored outlines (use_stored_outlines=FALSE)
  4. SQL Plan management will now handle these statements as well.

Influence of Stored Outlines on SPM

- We have a SQL statement with:
  - Stored Outline
  - SPM Baseline
- If stored outline is activated then SPM baseline is not used.
- In future releases stored outlines will be desupported.
- SPM should be better:
  - Stored outlines will always generate same plane, hence no potential progress in performance
  - Only one stored outline can be used for the statement
  - If index is dropped still a partial stored outline can be used for generating execution plan but the consequences can be catastrophic.

SPM Baselines and SQL Profiles

- SQL statement can have:
  - SPM Baseline
  - SQL Profile
    - SQL Profile provides additional information to the CBO for scaling up/down the statistical data (base cardinality, join cardinality,...)
  - When SQL profile is present CBO may choose different accepted plan.

SPM Baselines and Adaptive Cursor Sharing

- Adaptive Cursor Sharing (ACS)
  - Multiple cursors are present for a given bind sensitive SQL statement with different plans – several plans may be accepted
- SPM and ACS
  - Each cursor is generated by forcing a hard parse of the statement (peeking at the values of bind variables).
  - Each of the accepted plans is optimal for a different bind set.
  - Optimizer will select the best plan for the current bind set.

Conclusions

1. Do not store baselines for all SQL Statements – this will overkill you SYSAUX tablespace. Store only those, which are performance critical and might change due to changed statistics.
2. Dropping Indexes, which were used in verified execution plans **will change** the plans!
3. Apparently doesn't work with RULE based optimizer or even RULE hint!

References:

- Oracle® Database Performance Tuning Guide 11g Release 1 – Chapter 15 - Using SQL Plan Management
- Jože Senegačnik, SQL Plan Management, SIOUG 2008
- Blog “Inside the Oracle Optimizer - Removing the black magic” – SQL Plan Management - http://optimizermagic.blogspot.com/

Thank you for your interest!

Q&A