
ORACLE®

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Plan Stability

How to Get Creative with SQL Plan Management and SQL Profiles (and SQL Patch)
DOAG 2017 Datenbank - Düsseldorf

Nigel Bayliss
nigel.bayliss@oracle.com
@vldbb
http://blogs.oracle.com/optimizer

Optimizer Product Manager

ORACLE

Why Control Execution Plans?

- Get better plans without changing application code
- Reduce MTTR
- Discover better SQL execution plans
  - Proactively with SQL tuning advisor and SQL profiles
  - Strategically with SQL plan management *evolution*
- Avoid query performance regression altogether

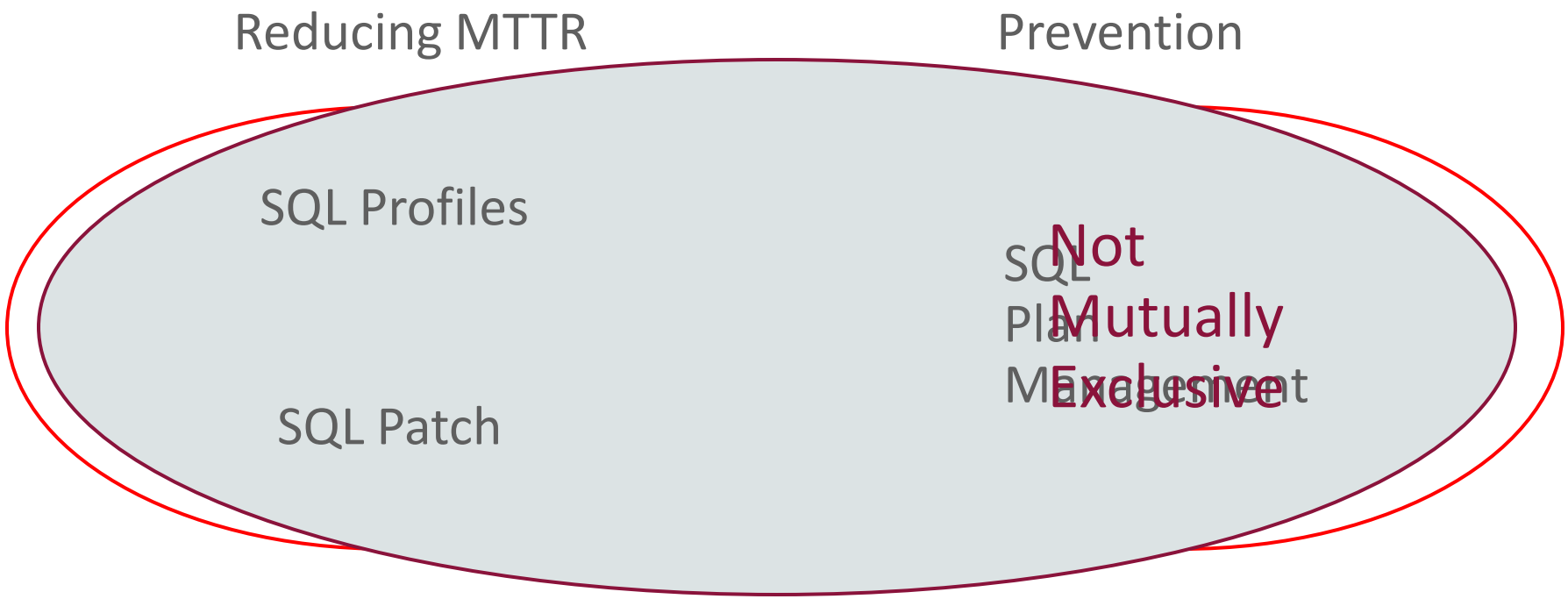
Common Questions

- Why do SQL execution plans change?
- How do I prevent execution plans from changing?
- What's the difference between SQL profiles and SQL plan baselines?
- How can I keep my SQL profiles relevant over time?
- What if I want to use SQL profiles *and* SQL plan baselines?
- Is SQL plan management practical in my environment?

Features for Controlling Execution Plans

- SQL Profiles
  - DBMS_SQLTUNE
- SQL Plan Management
  - DBMS_SPM
- SQL Patch
  - DBMS_SQLDIAG_INTERNAL
  - DBMS_SQLDIAG (Oracle Database 12c Release 2)
- Stored Outlines
  - Deprecated in Oracle Database 11g Release 1

Resolving and Preventing Issues



SQL Patch

Use SQL Patch to Hint SQL Statements

- https://blogs.oracle.com/optimizer/entry/how_can_i_hint_a

```
select num
from   tab1
where  id = 10;
```

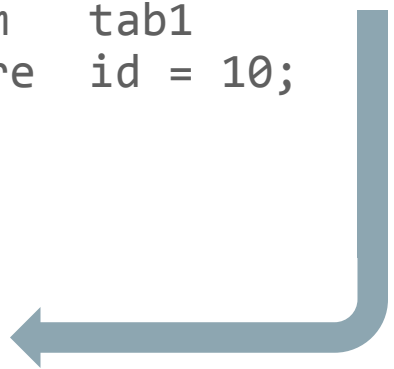
```
select /*+ FULL(tab1) */ num
from   tab1
where  id = 10;
```

Apply hints to a SQL statement using SQL Patch

sys.dbms_sqldiag_internal.i_create_patch

```
declare
   v_sql CLOB;
begin
   select sql_fulltext
   into   v_sql
   from   v$sqlarea
   where  sql_id='37090abayamah';

   sys.dbms_sqldiag_internal.i_create_patch(
      sql_text  =>v_sql,
      hint_text =>'FULL(@"SEL$1" "TAB1"@"SEL$1")',
      name      =>'q0_patch');
end;
/
```

```
select /*+ FULL(tab1) */ num
from   tab1
where  id = 10;
```



Apply hints to a SQL statement using SQL Patch

```
select /*+ FULL(tab1) */ num
from   tab1
where  id = 10;

SELECT * FROM table(DBMS_XPLAN.DISPLAY_CURSOR(FORMAT=>'OUTLINE'))

  /*+
      BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      ...
      FULL(@"SEL$1" "TAB1"@"SEL$1")
      ...
      END_OUTLINE_DATA
  */
```

What if we want to disable hints for a SQL statement?

```
select /*+ USE_HASH(tab1 tab2) */
       sum(num)
from   tab1
where  id in
(select id
 from   tab2
 where  ty = 'T10');
```

(c) Real World Performance Group ☺

A Hint that Disables Hints

```
select /*+ USE_HASH(tab1 tab2) */
       sum(num)
from   tab1
where  id in
(select id
 from   tab2
 where  ty = 'T10');
```

```
declare
   v_sql CLOB;
begin
   select sql_fulltext
   into   v_sql
   from   v$sqlarea
   where  sql_id='bdrzvmxvcju4y';

   sys.dbms_sqldiag_internal.i_create_patch(
      sql_text  =>v_sql,
      hint_text =>'IGNORE_OPTIM_EMBEDDED_HINTS',
      name      =>'q1_patch');
end;
/
```

(c) Real World Performance Group ☺

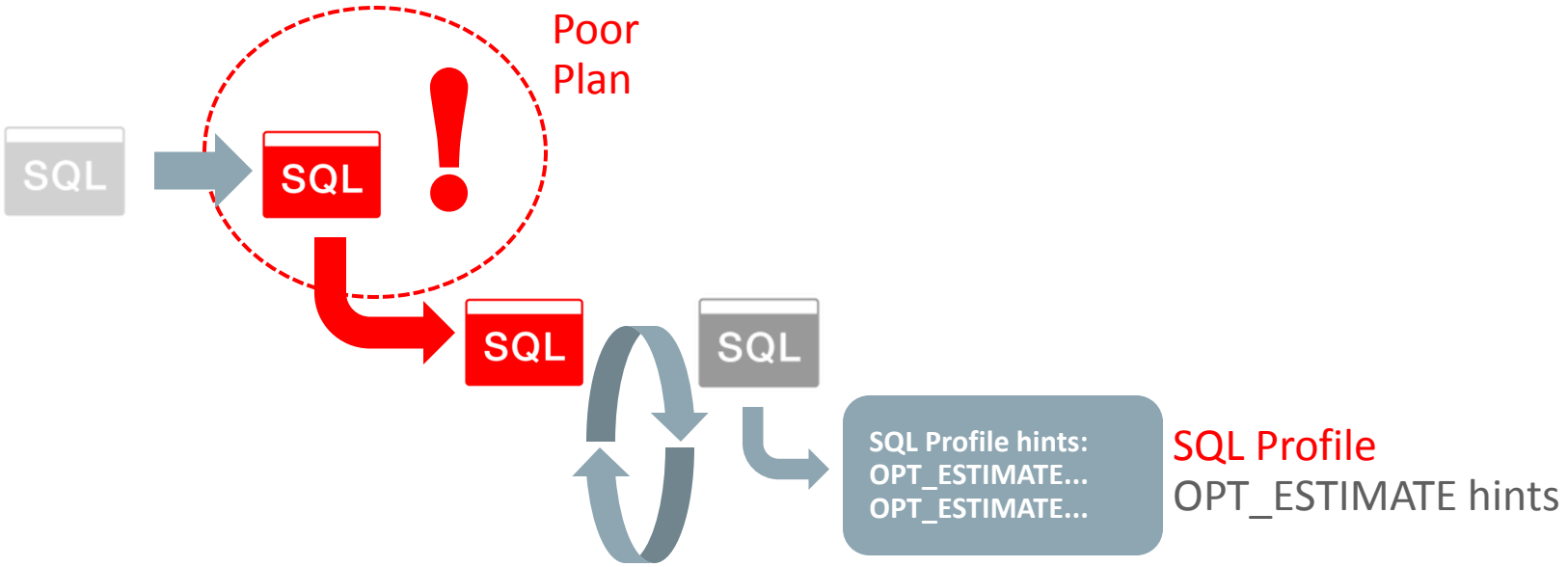
SQL Patch – Public Interface in Oracle Database 12c Release 2

- Prior to Oracle Database 12c Release 2
  - SQL patch API has limited utility because HINT_TEXT is VARCHAR2
- Beginning with Oracle Database 12c Release 2
  - Procedure becomes a function
  - HINT_TEXT is CLOB
  - **CREATE_SQL_PATCH** is new to public API: DBMS_SQLDIAG
  - New SQL_ID parameter as well as SQL_TEXT
- Patch #17203284 available for Oracle Database 12.1.0.2

```
patch_name:= dbms_sqldiag.create_sql_patch(
             sql_id    => 'wn8qqwkgarr65',
             hint_text => 'IGNORE_OPTIM_EMBEDDED_HINTS',
             name      => 'ignore_hint_patch');
```

SQL Profiles

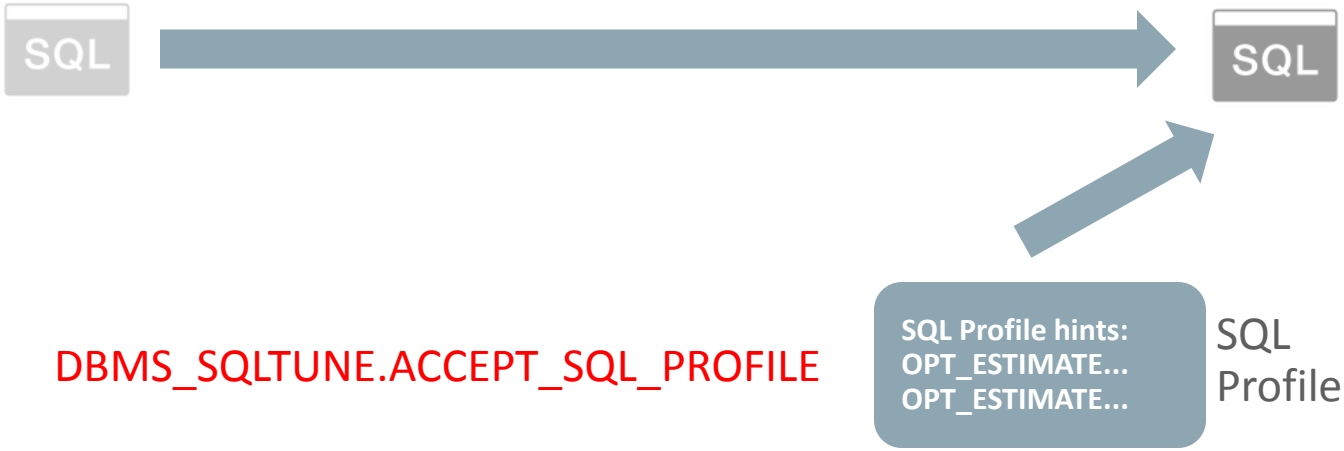
SQL Profile Example Flow



DBMS_SQLTUNE.CREATE_TUNING_TASK

SQL Tuning Advisor rest executes the query and finds a better plan

SQL Profile Example Flow



What Does a SQL Profile Look Like?

```
SELECT  extractValue(value(h),'.') AS hint
FROM    DBMSHSXP_SQL_PROFILE_ATTR dspa,
        TABLE(xmlsequence(
          extract(xmltype(dspa.comp_data),'/outline_data/hint'))) h
where   dspa.profile_name = 'SYS_SQLPROF_015a039fbe670000';


HINT
-----------------------------------------------
OPT_ESTIMATE(@"SEL$1", TABLE, "ORDERS"@"SEL$1", SCALE_ROWS=9.426849516e-06)
OPT_ESTIMATE(@"SEL$1", INDEX_SCAN, "ORDERS"@"SEL$1", "ORDERS_SHIPPED_IDX", SCALE_ROWS=6.042852254e-06)
OPTIMIZER_FEATURES_ENABLE(default)
```

SQL Profiles

What about a longer-term solution?

- The **OPT_ESTIMATE** adjustments may not be appropriate "forever"
- SQL tuning advisor is a good way to find better SQL execution plans, but the cardinality adjustments don't guarantee a plan in the long term
  - Data volume and data distribution might change
  - The physical database schema might change (e.g. adding and removing indexes)

SQL Profiles

- It is possible to FORCE match queries with literals to use a SQL profile
  - DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (… force_match=>TRUE …)
- There are export/import tricks to apply arbitrary hints to SQL statements. See, for example:
  - Kerry Osborne
  - Randolf Geist

SQL Plan Management

SQL Plan Management - Terminology

"Repeatable" SQL Statement

```
SELECT a.data,
       b.data
FROM   tab1 a
JOIN tab2 b ON
        b.tab1_id = a.id
WHERE  a.code = :1
```

CURSOR_SHARING=FORCE works too

A signature is a unique SQL identifier generated from the normalized SQL text (uncased and with whitespaces removed).

Eg: **3578512732271849948**

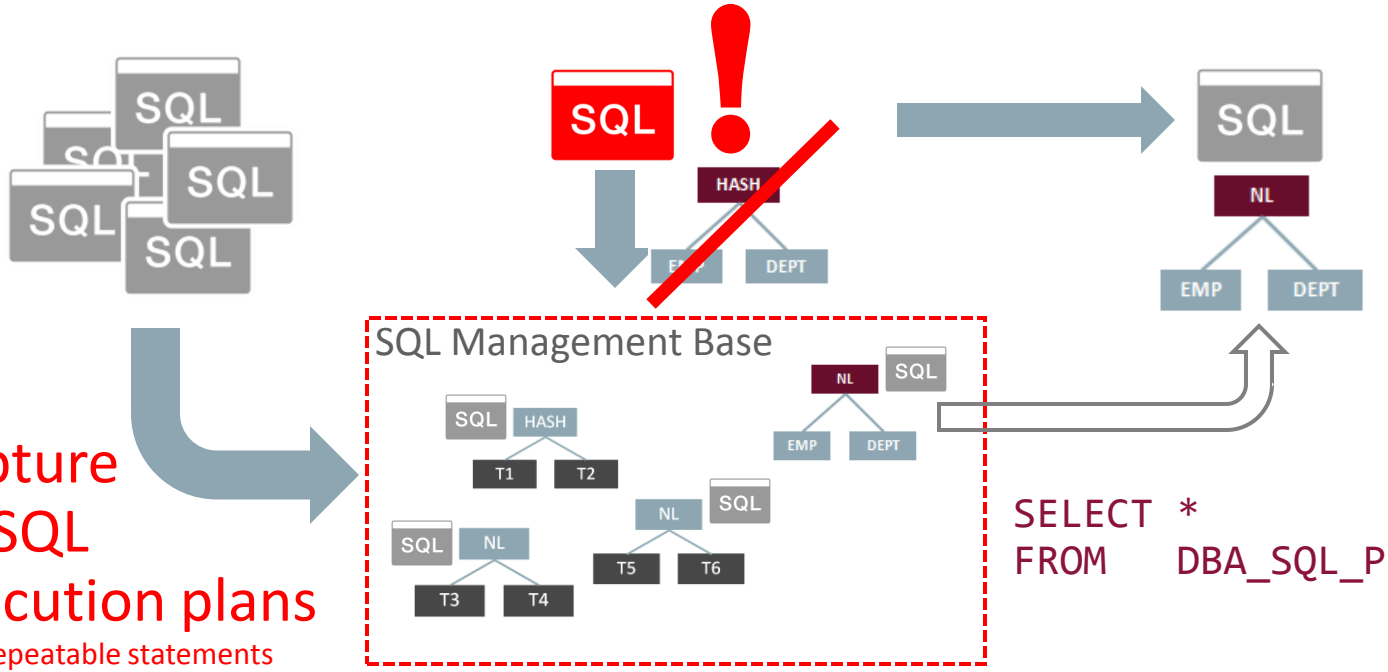
Used by SQL profiles too

```
SELECT a.data,
       b.data
FROM   TAB1 a
JOIN tab2 b
  ON b.tab1_id = a.id
WHERE  a.code = :1
```

```
select a.data,
       b.data
from   tab1 a
join   tab2 b
 on    b.tab1_id = a.id
where  a.code = :1
```

Prevent Query Regression

An automated, strategic approach



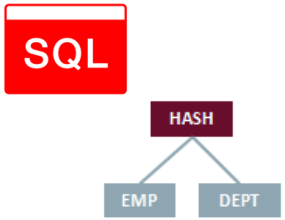
Capture *all* SQL execution plans

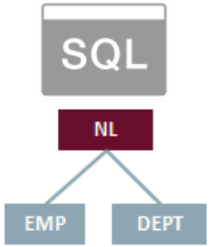
...for repeatable statements

```
SELECT *
FROM   DBA_SQL_PLAN_BASELINES;
```

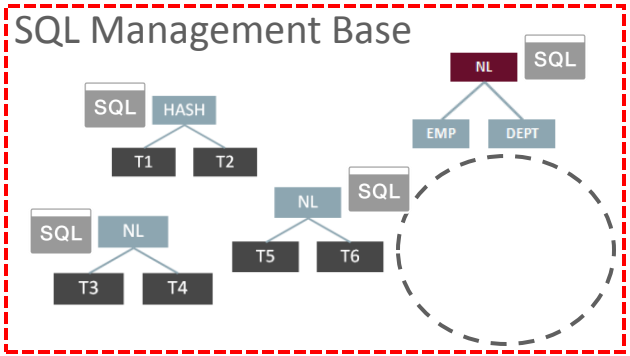
Prevent Query Regression

Validating new SQL execution plans automatically





Capture Everything!

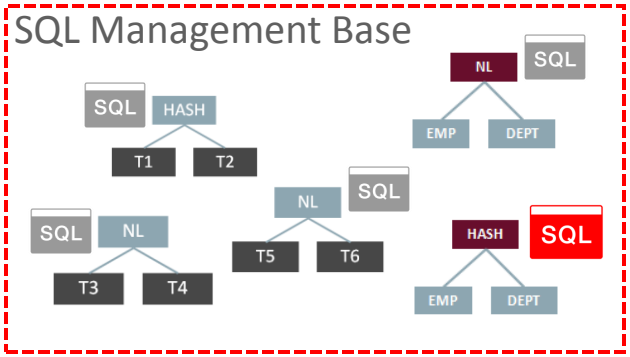


Is this plan better?

Evolve

Prevent Query Regression

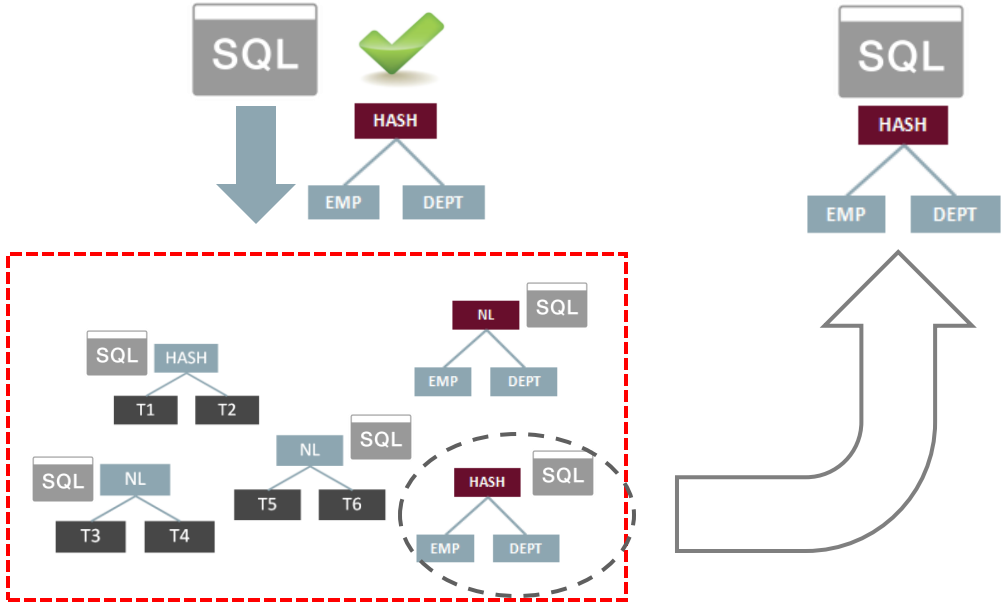
Validating new SQL execution plans automatically




```
DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE();

DBMS_SPM.EXECUTE_EVOLVE_TASK();
```

Prevent Query Regression

Use the improved SQL execution plan



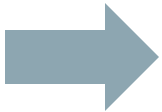
What Does a SQL Plan Baseline Look Like?

```
select plan_table_output
from   table(dbms_xplan.display_sql_plan_baseline('SQL_6c0757fcfdf7ec80','SQL_PLAN_6s1urzmyzgv402db27268'));

Plan hash value: 358968005

-----------------------------------------------------------------------------------------------
| Id  | Operation                             | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |       |       |       |     2 (100)|          |
|   1 |  SORT AGGREGATE                       |       |     1 |     9 |     0   (0)|          |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| SALES |     1 |     9 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                   | SI    |     1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

What Does a SQL Plan Baseline *Really* Look Like?

```
SELECT  extractValue(value(h),'.') AS hint
FROM    sys.sqlobj$plan od,
        TABLE(xmlsequence(
          extract(xmltype(od.other_xml),'/*/outline_data/hint'))) h
WHERE   od.other_xml is not null
AND     (signature,category,obj_type,plan_id) = (select signature,
                                                        category,
                                                        obj_type,
                                                        plan_id
                                                from    sys.sqlobj$ so
                                                where   so.name = 'SQL_PLAN_6s1urzmyzgv402db27268');
```

```
IGNORE_OPTIM_EMBEDDED_HINTS
OPTIMIZER_FEATURES_ENABLE('12.1.0.2')
DB_VERSION('12.1.0.2')
ALL_ROWS
OUTLINE_LEAF(@"SEL$1")
INDEX_RS_ASC(@"SEL$1" "SALES"@"SEL$1" ("SALES"."ID"))
BATCH_TABLE_ACCESS_BY_ROWID(@"SEL$1" "SALES"@"SEL$1")
```



```
-----------------------------------------------------
| Id  | Operation                          | Name  |
-----------------------------------------------------
|   0 | SELECT STATEMENT                   |       |
|   1 |  SORT AGGREGATE                    |       |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| SALES |
|*  3 |    INDEX RANGE SCAN                | SI    |
-----------------------------------------------------
```

Capture "Everything"

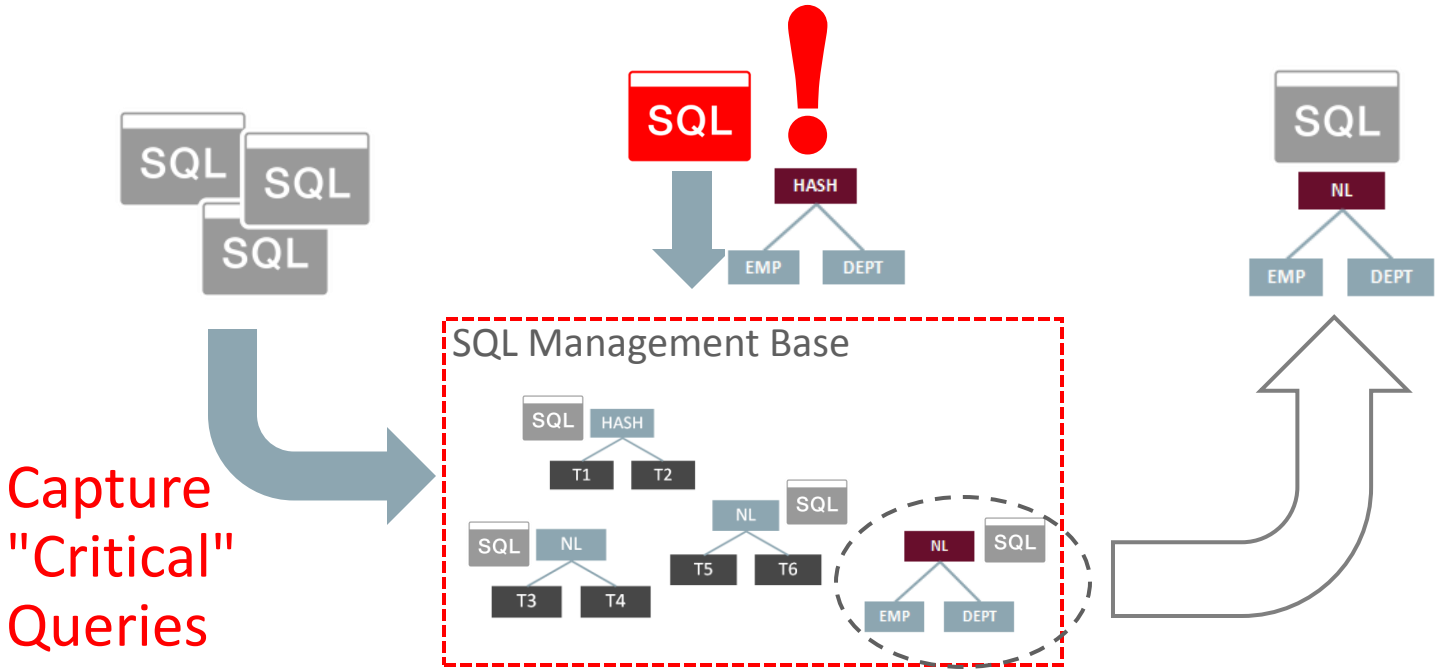
- Capture repeatable SQL statements:
  - `optimizer_capture_sql_plan_baselines=TRUE`
  - Filter is available in Oracle Database 12c Release 2
    - SQL Text, Parsing Schema, Module, Action
- Periodically capture everything from the cursor cache
  - Repeat:
    - `dbms_spm.load_plans_from_cursor_cache(enabled=>'YES');`
  - Filter has been available since Oracle Database 11g
  - Duplicates are not stored

Other Capture Sources

- From SQL tuning sets
  - Filtering is very easy
- From AWR repository in Oracle Database 12c Release 2
  - Between start and end snapshots

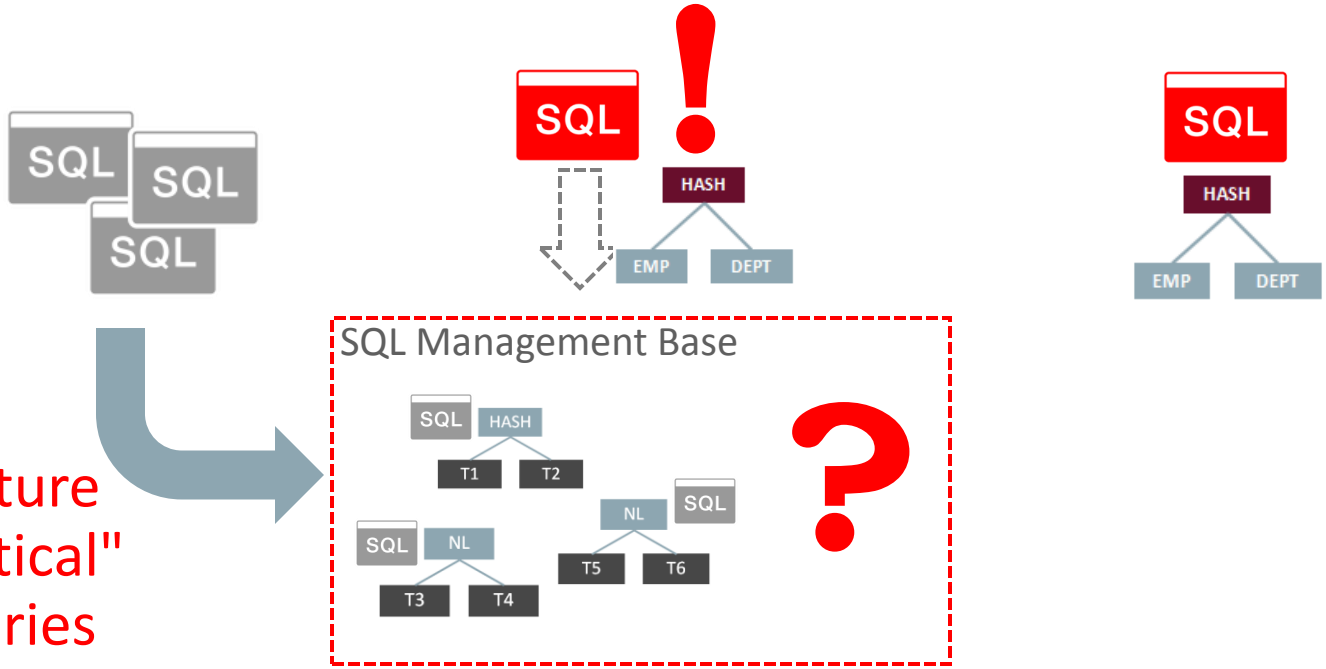
A Tactical Approach

Capture a subset of queries



Tactical Approach

Regression is still possible



Creative Capture

For example: tactical capture

```
for rec in (select sql_id
            from   v$sqlarea
            where  executions > 10000
            and    parsing_schema_name not in ('SYS','SYSTEM'))
loop
   ret := dbms_spm.load_plans_from_cursor_cache(
               sql_id   => rec.sql_id,
               enabled  => 'YES');
end loop;
```

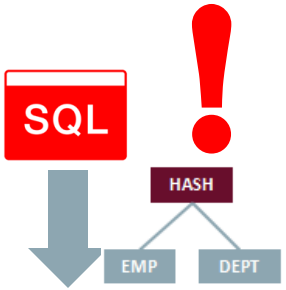
Creative Capture

You can choose to enable SQL plan baselines later

```
for rec in (select sql_id
            from   v$sqlarea
            where  executions > 10000
            and    parsing_schema_name not in ('SYS','SYSTEM'))
loop
   ret := dbms_spm.load_plans_from_cursor_cache(
               sql_id=>rec.sql_id,
               enabled=>'NO');
end loop;
```

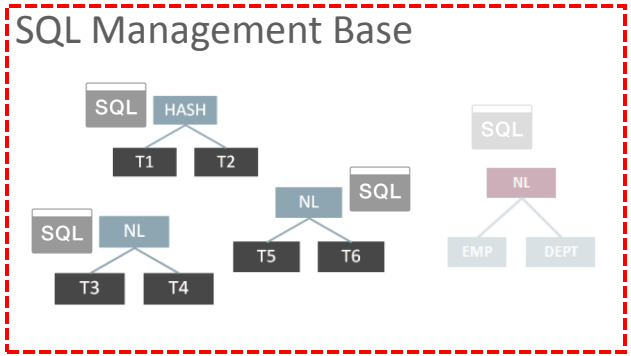
Tactical Approach

Maybe a better plan is already available but not enabled



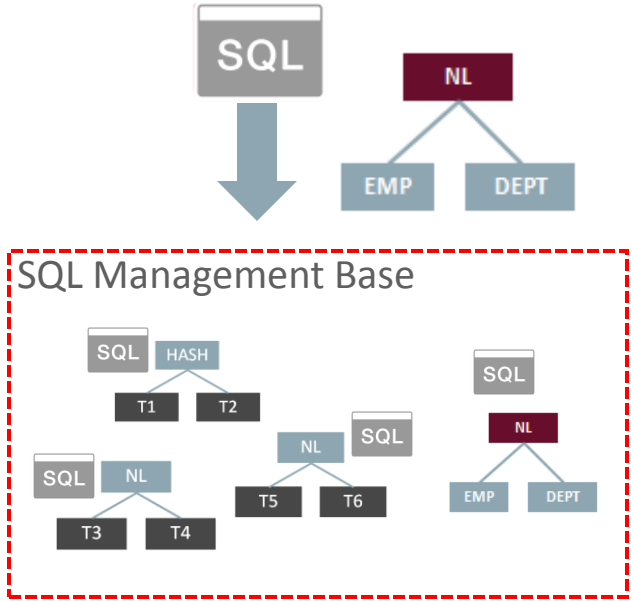
SQL plan baselines don't *have to* be enabled immediately:

```
dbms_spm.load_plans_from_cursor_cache(...
                                enabled=>'NO');
```



Tactical Approach

Enable it – low MTTR




```
dbms_spm.alter_sql_plan_baseline(
        ...
        attribute_name  =>'enabled',
        attribute_value =>'YES');
```

Automation in Oracle Database 12c

Controlling SPM evolution

```
SELECT PARAMETER_NAME, PARAMETER_VALUE AS VALUE
FROM   DBA_ADVISOR_PARAMETERS WHERE
( (TASK_NAME = 'SYS_AUTO_SPM_EVOLVE_TASK') AND
( (PARAMETER_NAME = 'ACCEPT_PLANS') OR
  (PARAMETER_NAME = 'TIME_LIMIT') ) );


PARAMETER_NAME            VALUE
------------------------- ----------
ACCEPT_PLANS              TRUE
TIME_LIMIT                3600

BEGIN
   DBMS_AUTO_TASK_ADMIN.ENABLE (
   client_name => 'sql tuning advisor‘
   ,   operation   => NULL
   ,   window_name => NULL);
END;
```

```
BEGIN
   DBMS_SPM.SET_EVOLVE_TASK_PARAMETER(
      task_name => 'SYS_AUTO_SPM_EVOLVE_TASK' ,
      parameter => 'ACCEPT_PLANS',
      value => 'FALSE');
END;
/
```

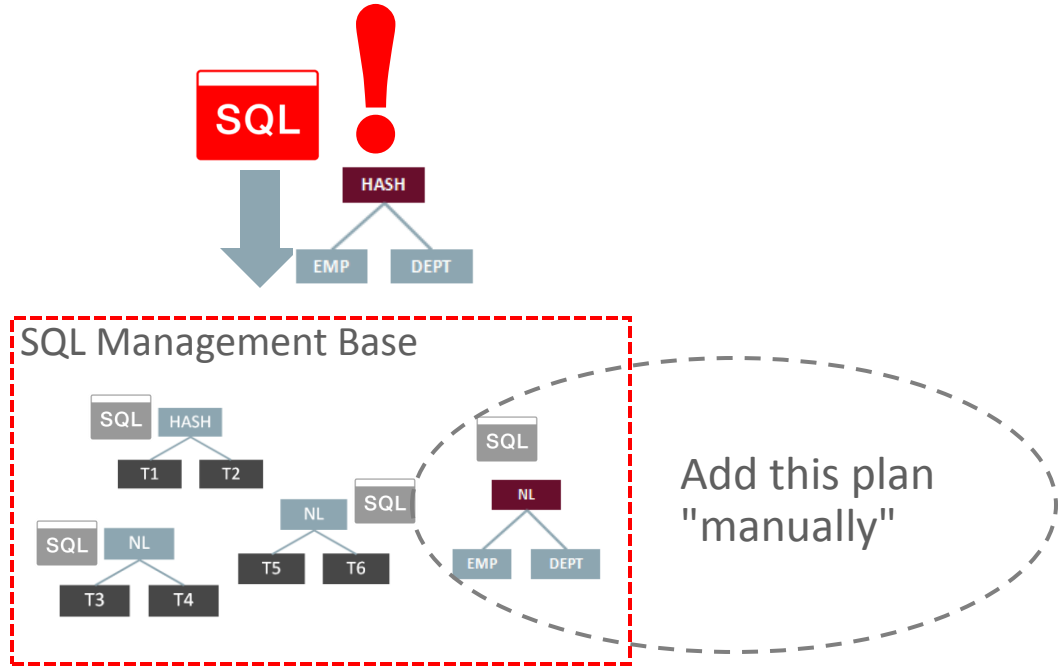
Execution plans are validate but they are *not* accepted automatically.

You Can "Capture Only"

- Capture repeatable SQL statements:
  - `optimizer_capture_sql_plan_baselines=TRUE`
  - `optimizer_use_sql_plan_baselines=FALSE`
- Periodically capture everything from the cursor cache
  - Repeat:
    - `dbms_spm.load_plans_from_cursor_cache(enabled=>'NO');`

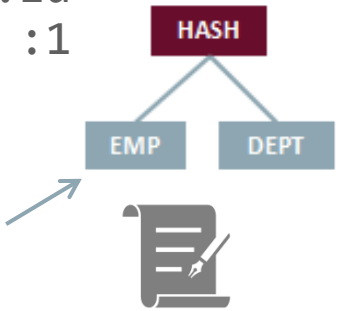
Tactical Approach

Supply a better plan

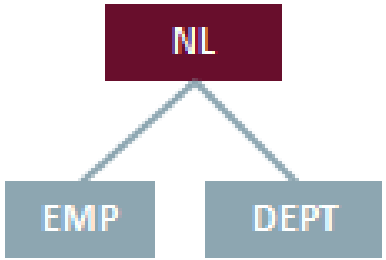


Correcting Poor SQL Execution Plans

```
SELECT a.data,
       b.data
FROM   emp  a
JOIN   dept b ON
       b.id = a.id
WHERE  a.code = :1
```



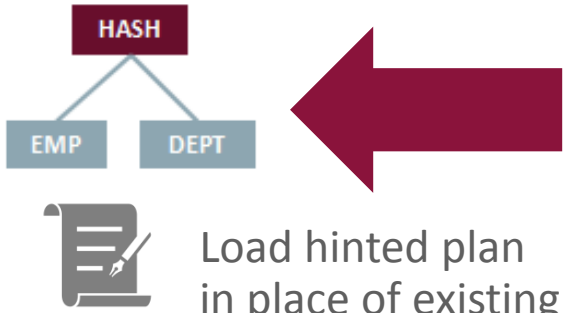
Create SQL plan baseline



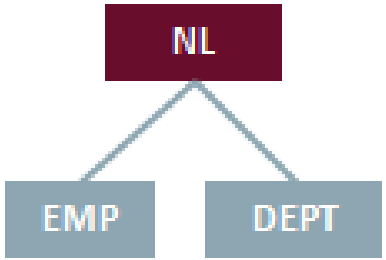

```
SELECT /*+ USE_NL(a b) */
       a.data,
       b.data
FROM   emp  a
JOIN   dept b ON
       b.id = a.id
WHERE  a.code = :1
```

Correcting Poor SQL Execution Plans

```
SELECT a.data,
       b.data
FROM   emp  a
JOIN   dept b ON
       b.id = a.id
WHERE  a.code = :1
```



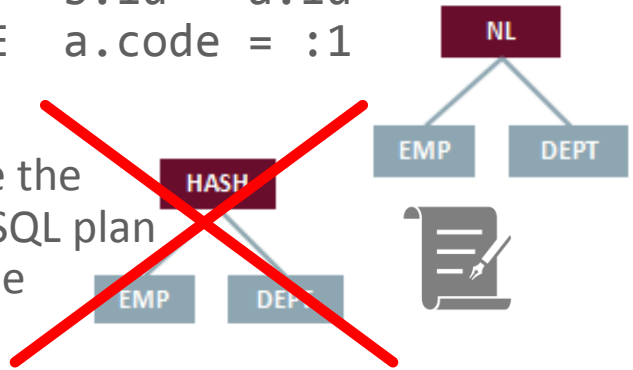
Load hinted plan
in place of existing
SQL plan baseline



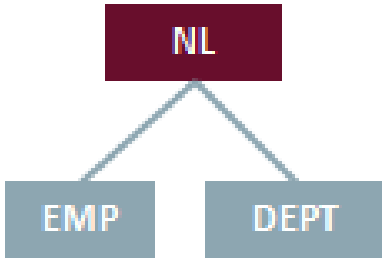

```
SELECT /*+ USE_NL(a b) */
       a.data,
       b.data
FROM   emp  a
JOIN   dept b ON
       b.id = a.id
WHERE  a.code = :1
```

Correcting Poor SQL Execution Plans

```
SELECT a.data,
       b.data
FROM   emp  a
JOIN   dept b ON
       b.id = a.id
WHERE  a.code = :1
```



Disable the "bad" SQL plan baseline

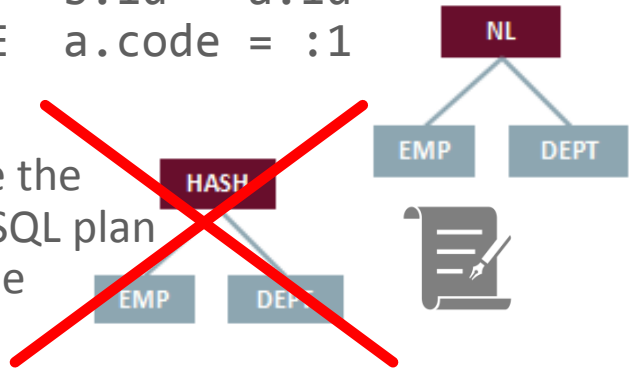



```
SELECT /*+ USE_NL(a b) */
       a.data,
       b.data
FROM   emp  a
JOIN   dept b ON
       b.id = a.id
WHERE  a.code = :1
```

Correcting Poor SQL Execution Plans

```
SELECT a.data,
       b.data
FROM   emp  a
JOIN   dept b ON
       b.id = a.id
WHERE  a.code = :1
```

Disable the "bad" SQL plan baseline



See white paper:

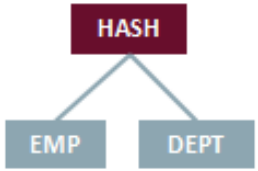
SQL Plan Management with Oracle Database 12c Release 2

Using SPM to Correct Regressed SQL Statements

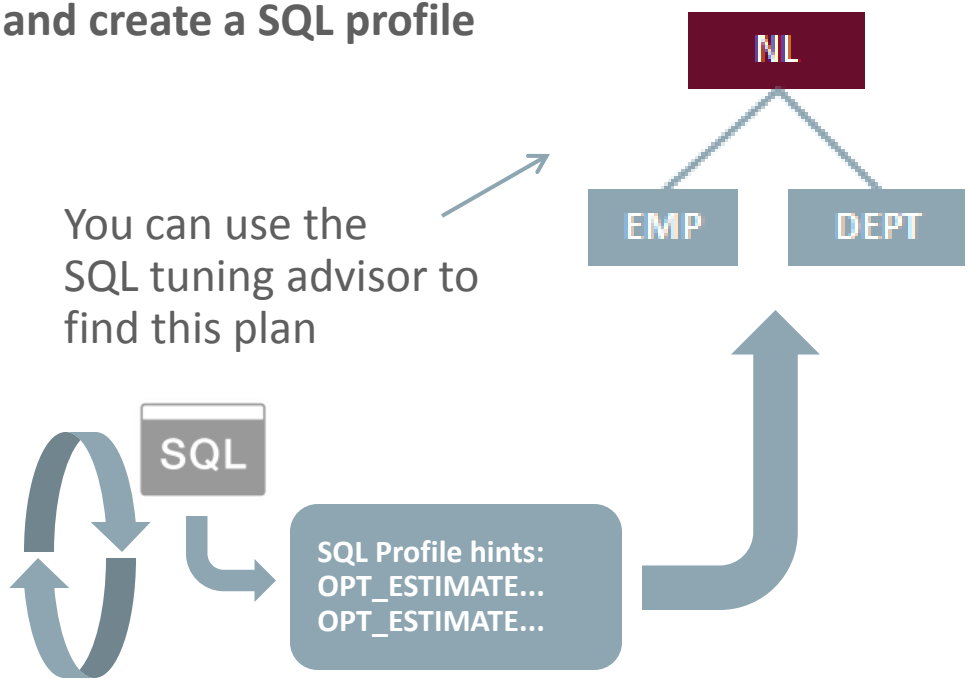
Use SQL Tuning Advisor and SQL Plan Management

Let STA find a better plan and create a SQL profile

```
SELECT a.data,
       b.data
FROM   emp  a
JOIN   dept b ON
       b.id = a.id
WHERE  a.code = :1
```



You can use the SQL tuning advisor to find this plan



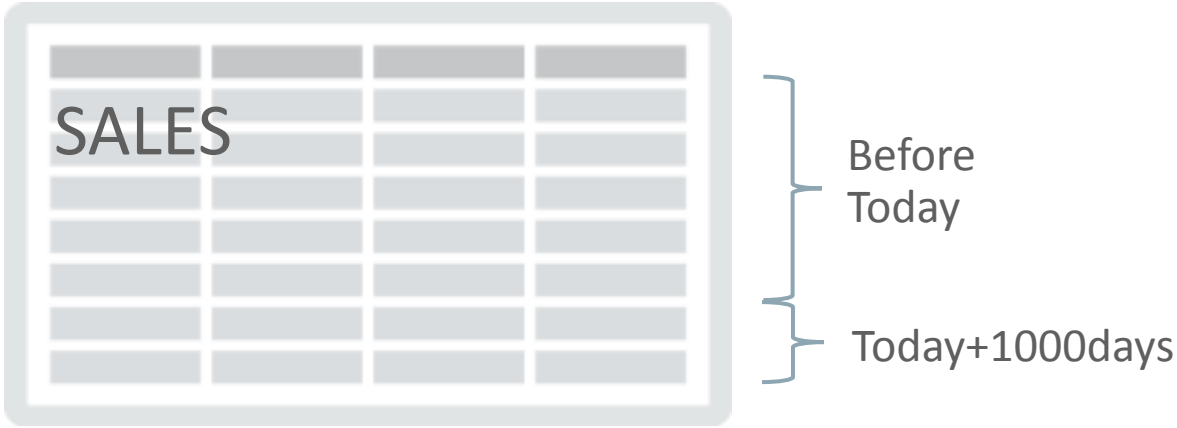
DBMS_SQLTUNE.CREATE_TUNING_TASK

Demo

SQL Profile workflow, followed by SQL plan baseline creation and evolution

A Challenging Data Distribution

Table with index and data column with range skew



SQL Profiles and SQL Plan Management

Poorly performing query

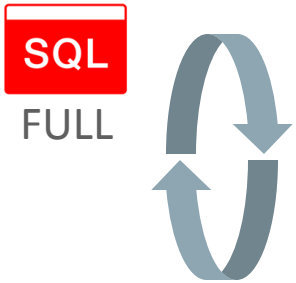
FULL

Actually returns a
few hundred rows

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |       |       |   351 (100)|          |
|*  1 |  TABLE ACCESS FULL| SALES |   333K|  4233K|   351   (3)| 00:00:01 |
-------------------------------------------------------------------------------
```

SQL Profiles and SQL Plan Management

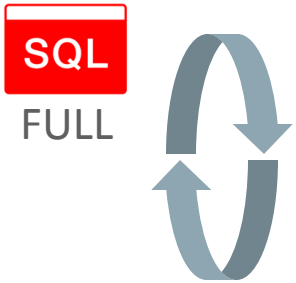
Use SQL tuning advisor to find a better plan – e.g. Index access method




```
dbms_sqltune.create_tuning_task(sql_id=>...

dbms_sqltune.execute_tuning_task
```

SQL Profiles and SQL Plan Management

Use SQL tuning advisor to find a better plan – e.g. Index access method

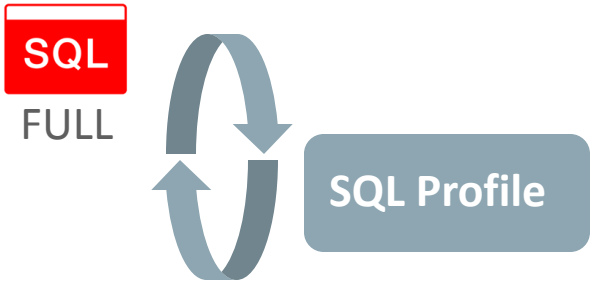


dbms_sqltune.
report_tuning_task

```
Recommendation (estimated benefit: 99.52%)
 ----------------------------------------
 - Consider accepting the recommended SQL profile.
   execute dbms_sqltune.accept_sql_profile(task_name => 'TASK_632',
           task_owner => 'ADHOC', replace => TRUE);

 Validation results
 ------------------
 The SQL profile was tested by executing both its plan and the original plan
 and measuring their respective execution statistics. A plan may have been
 only partially executed if the other could be run to completion in less time.

                              Original Plan  With SQL Profile  % Improved
                              -------------  ----------------  ----------
 Completion Status:                COMPLETE          COMPLETE
 Elapsed Time (s):                 .015779          .000079      99.49 %
 CPU Time (s):                     .014097                0        100 %
 User I/O Time (s):                      0                0
```

SQL Profiles and SQL Plan Management

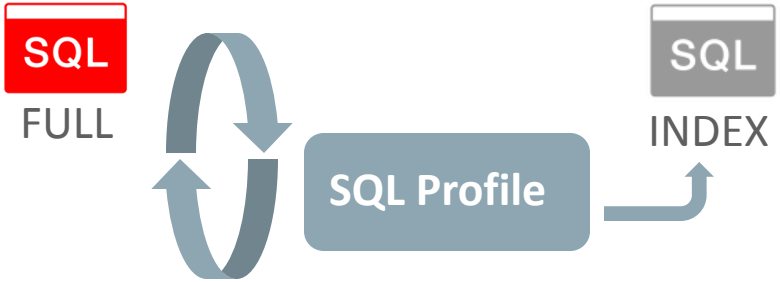
Use SQL tuning advisor to find a better plan – e.g. Index access method




```
dbms_sqltune.accept_sql_profile
```

SQL Profiles and SQL Plan Management

Use SQL tuning advisor to find a better plan – e.g. Index access method

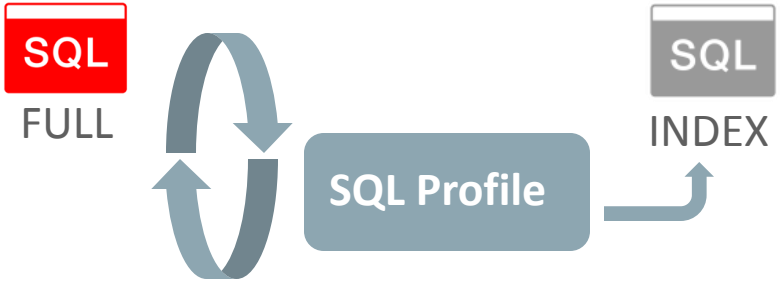



```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |        |       |       |     6 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES  |   454 |  5902 |     6   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | SALESI |   724 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

SQL profile SYS_SQLPROF_015c1c35866c0017 used for this statement

SQL Profiles and SQL Plan Management

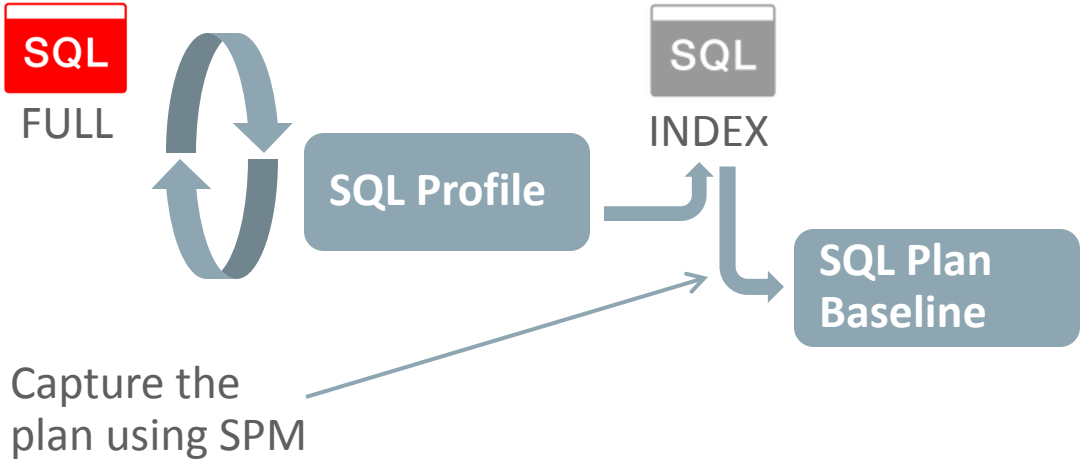
Use SQL tuning advisor to find a better plan – e.g. Index access method




```
select sql_id,exact_matching_signature,sql_profile
from   v$sqlarea
where  sql_text like '%PROFTEST%'
and    sql_profile is not null
order by sql_profile desc;

SQL_ID         EXACT_MATCHING_SIGNATURE  SQL_PROFILE
-------------- ------------------------- ---------------------------------------
G6y6gpnzww95b  10496852328441737191      SYS_SQLPROF_015c1c35866c0017
```

SQL Profiles and SQL Plan Management

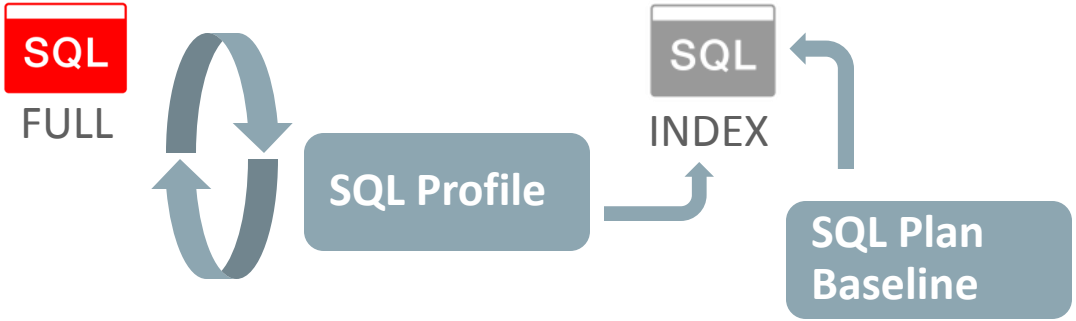
Capture the plan using SQL plan management




```
dbms_spm.load_plans_from_cursor_cache(
                    sql_id  =>'328yfewhu8732',
                    enabled =>'YES');
```

SQL Profiles and SQL Plan Management

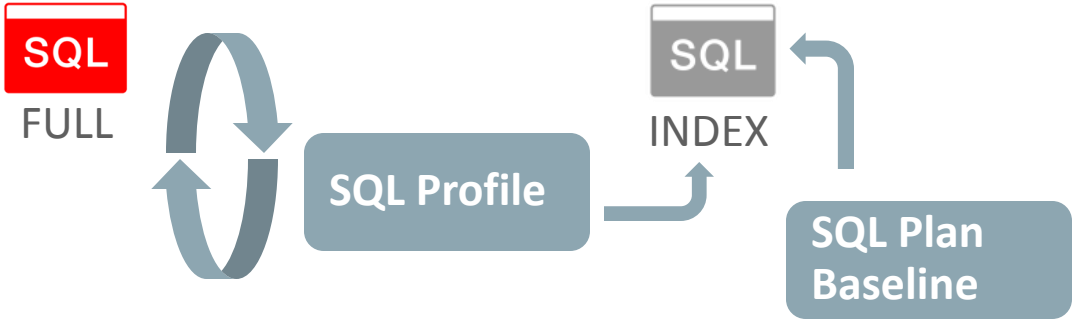
The SQL plan baseline enforces the plan even if SQL profile exists




```
-------------------------------------------------------------------------------------
| Id  | Operation                    | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |        |       |       |     6 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES  |   404 |  5252 |     6   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | SALESI |   691 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

Note
-----
   - SQL profile SYS_SQLPROF_015c1c35866c0017 used for this statement
   - SQL plan baseline SQL_PLAN_93b2gkgw80vz7f244f78f used for this statement
```

SQL Profiles and SQL Plan Management

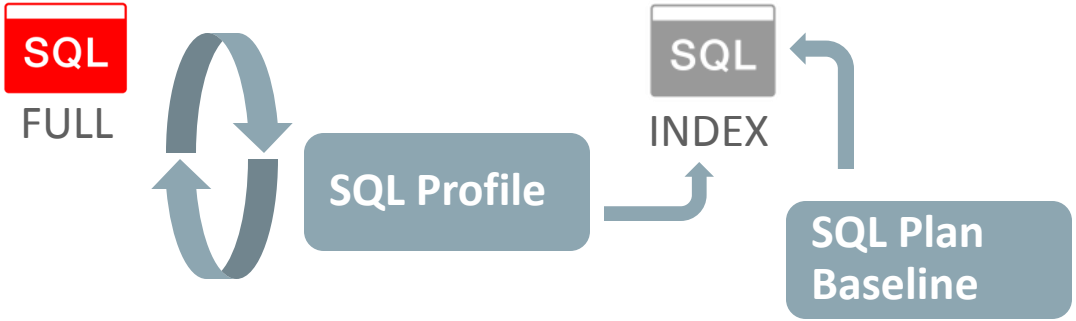
The SQL plan baseline enforces the plan even if SQL profile exists




```
select sql_id,sql_plan_baseline,sql_profile
from   v$sqlarea
where  sql_text like '%PROFTEST%'
and    sql_profile is not null
order by sql_profile desc;

SQL_ID         SQL_PLAN_BASELINE              SQL_PROFILE
-------------- ------------------------------ ----------------------------------------
G6y6gpnzww95b SQL_PLAN_93b2gkgw80vz7f244f78f SYS_SQLPROF_015c1c35866c0017
```

SQL Profiles and SQL Plan Management

The SQL plan baseline enforces the plan even if SQL profile exists

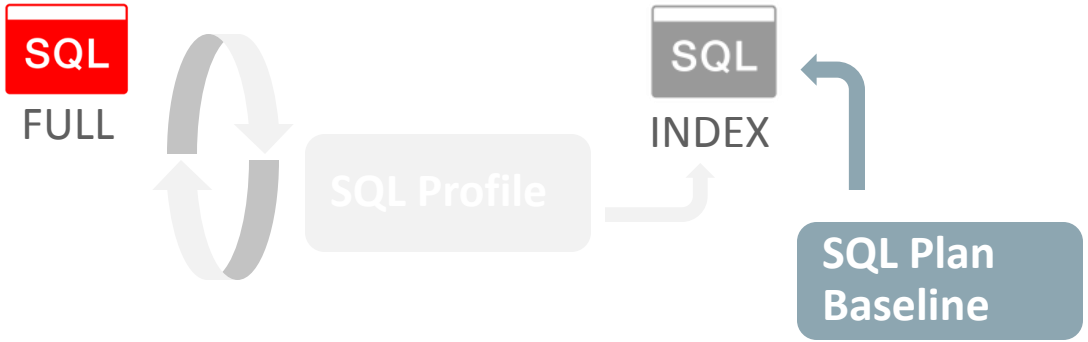



```
SELECT signature,sql_handle,plan_name,accepted
FROM   dba_sql_plan_baselines
WHERE  sql_text LIKE '%PROFTEST%';
```

```
SIGNATURE            SQL_HANDLE                     PLAN_NAME                      ACCEPTED
---------            ------------------------       -------------                  ----------------
10496852328441737191 SQL_91ac4f93f8806fe7           SQL_PLAN_93b2gkgw80vz7f244f78f YES
```

SQL Profiles and SQL Plan Management

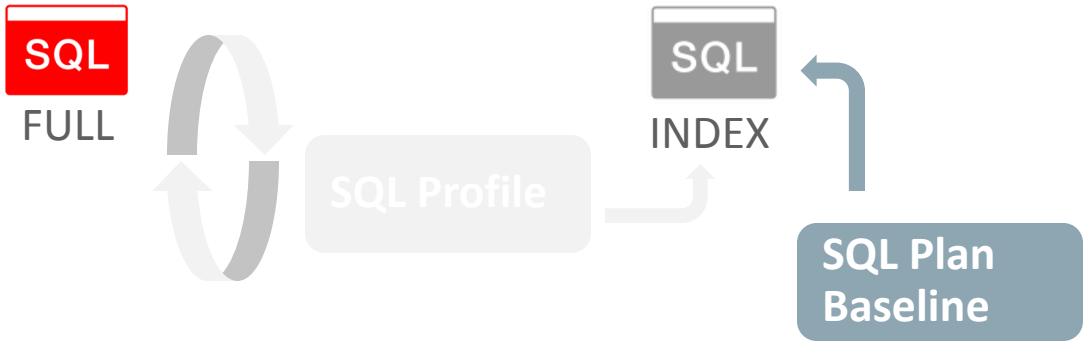
The SQL profile can be disabled




```
dbms_sqltune.alter_sql_profile(
                    name           =>'SYS_SQLPROF_015c1c35866c0017',
                    attribute_name =>'STATUS',
                    value          =>'DISABLED');
```

SQL Profiles and SQL Plan Management

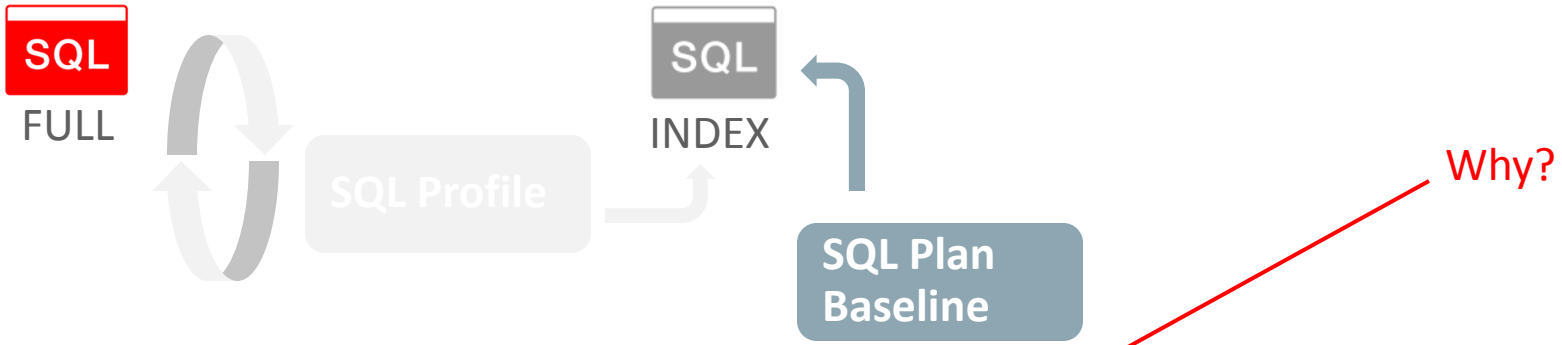
The SQL profile can be disabled




```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |         |       |       |  1720 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES   |   333K|  4233K|  1720   (1)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | SALESI  |   333K|       |   890   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------

Note
-----
   - SQL plan baseline SQL_PLAN_93b2gkgw80vz7f244f78f used for this statement
```

SQL Profiles and SQL Plan Management

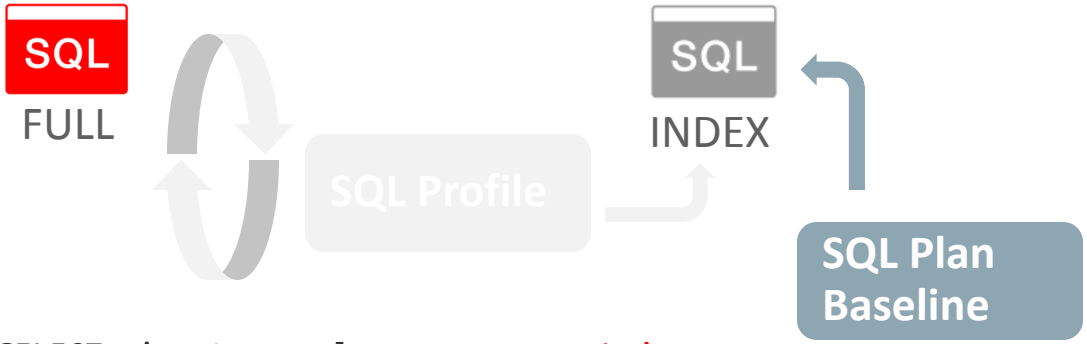
The SQL profile can be disabled




```
-------------------------------------------------------------------------------------
| Id  | Operation                    | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |        |       |       |  1720 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| SALES  |   333K|  4233K|  1720   (1)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | SALESI |   333K|       |   890   (1)| 00:00:01 |
-------------------------------------------------------------------------------------

Note
-----
   - SQL plan baseline SQL_PLAN_93b2gkgw80vz7f244f78f used for this statement
```

SQL Profiles and SQL Plan Management

Once the SQL profile has been disabled, new plans might be found

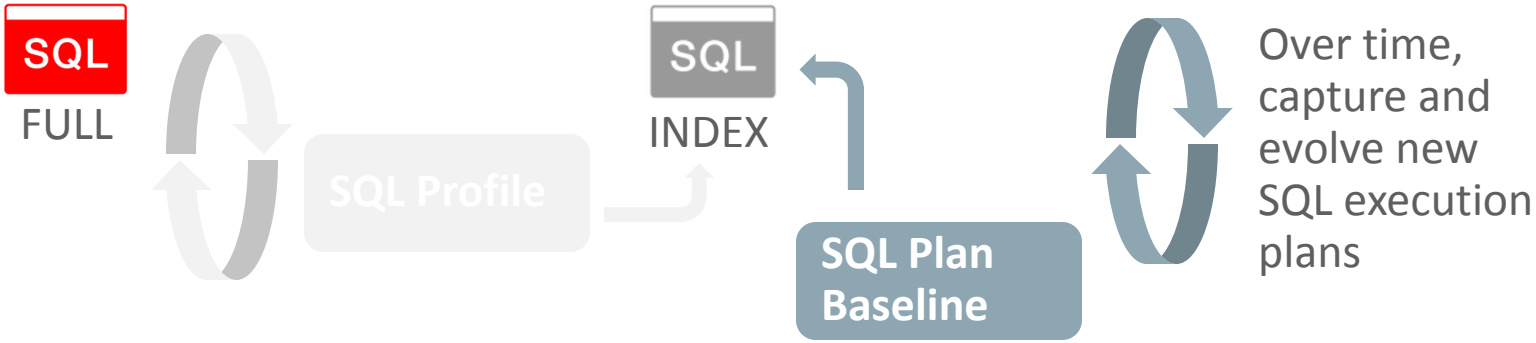



```
SELECT signature, plan_name, accepted
FROM   dba_sql_plan_baselines
WHERE  sql_text LIKE '%PROFTEST%';

                    SIGNATURE PLAN_NAME                      ACC
------------------------------ ------------------------------ ---
        10496852328441737191 SQL_PLAN_93b2gkgw80vz797367831 NO
        10496852328441737191 SQL_PLAN_93b2gkgw80vz7f244f78f YES
```

FULL plan

Example Flow

The SQL execution plan is managed in the long term

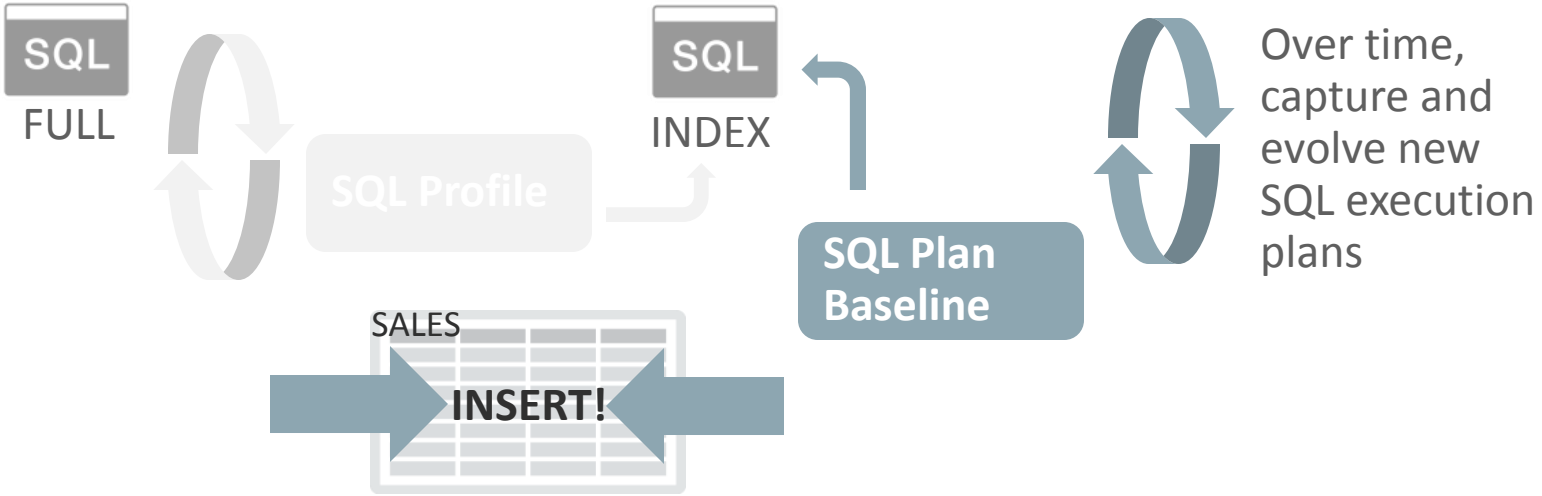



```
exec :report := dbms_spm.evolve_sql_plan_baseline();
```

Note: Automated in Oracle Database 12c

Example Flow

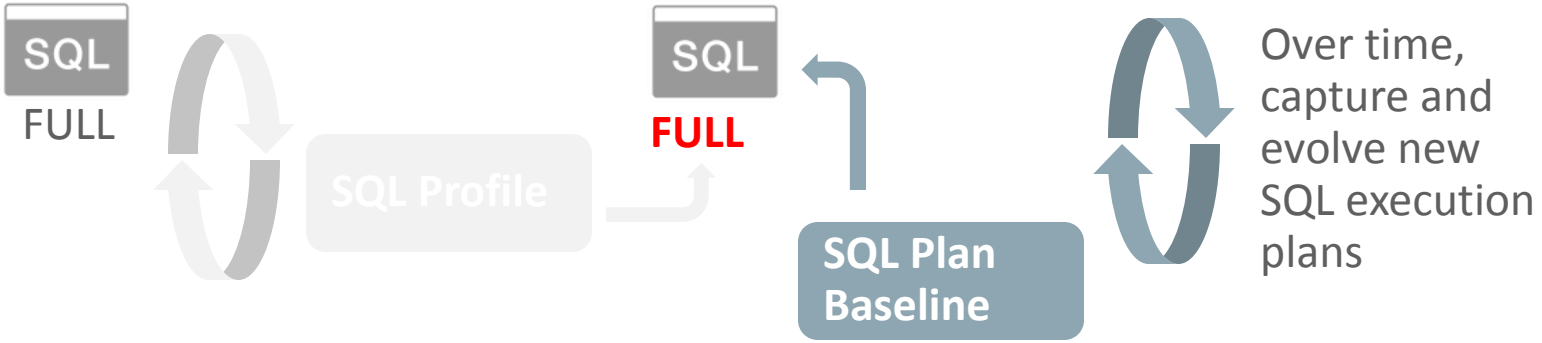
Data changes in the underlying tables...




```
exec :report := dbms_spm.evolve_sql_plan_baseline();
```

Example Flow

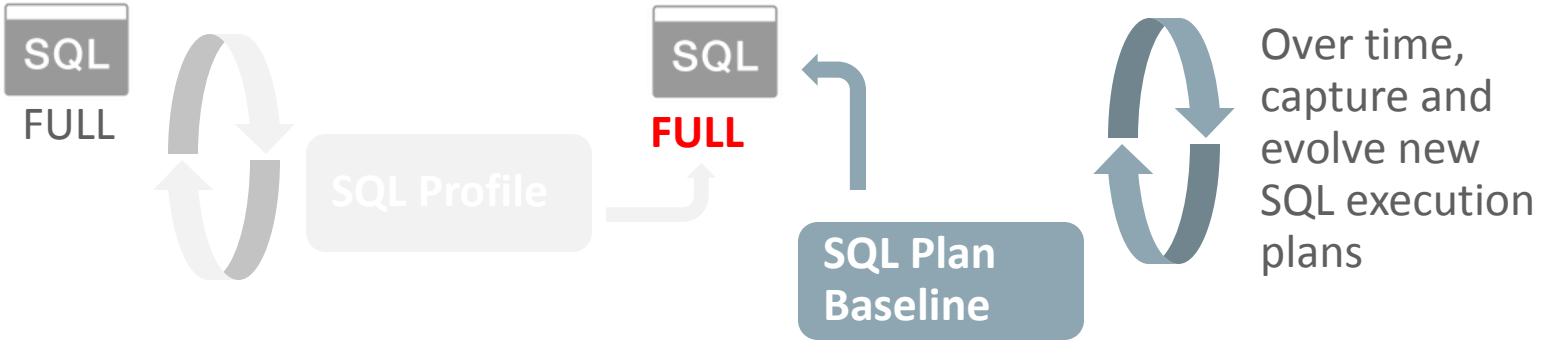
The full table scan plan is more appropriate now – it is verified before use




```
SELECT signature, plan_name, accepted
FROM   dba_sql_plan_baselines
WHERE  sql_text LIKE '%PROFTEST%';

                    SIGNATURE PLAN_NAME                          ACC
----------------------------- ------------------------------ ---
        10496852328441737191 SQL_PLAN_93b2gkgw80vz797367831 YES
        10496852328441737191 SQL_PLAN_93b2gkgw80vz7f244f78f YES
```

Example Flow

The full table scan plan is more appropriate now – it is verified before use




```
Plan hash value: 781590677

-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |       |       |  1053 (100)|          |
|*  1 |  TABLE ACCESS FULL| SALES |  1000K|    12M|  1053   (3)| 00:00:01 |
-------------------------------------------------------------------------------
```

Create SQL Plan Baselines for queries using SQL Profiles!

```
for rec in (select sql_id, sql_profile
            from   v$sqlarea
            where  sql_profile is not null
            and    sql_plan_baseline is null)
loop
   -- Note: loads all plans for given SQL statement

   ret := dbms_spm.load_plans_from_cursor_cache(
                                          sql_id  => rec.sql_id,
                                          enabled => 'YES');
end loop;
```

```
Note
-----
   - SQL profile SYS_SQLPROF_015a039fbe670000 used for this statement
   - SQL plan baseline SQL_PLAN_6bpdv812t0ckxea78bc40 used for this statement
```

Disable SQL Profile where a SQL Plan Baseline Exists

```
  for rec in (select sql_profile
              from   v$sqlarea
              where  sql_profile       is not null
              and    sql_plan_baseline is not null)
  loop
     dbms_sqltune.alter_sql_profile(
                             name           => rec.sql_profile,
                             attribute_name => 'STATUS',
                             value          => 'DISABLED');
  end loop;

Note
-----
   - SQL plan baseline SQL_PLAN_6bpdv812t0ckxea78bc40 used for this statement
```

Summary

- You can use SQL patch to apply hints in an ad-hoc manner
  - SE and EE
  - A reactive approach
- You can use SQL profiles and SQL plan baselines together
  - SQL profiles particularly good for finding better plans **reactively**
  - You can "hand over" plans controlled by SQL profiles to be managed by SQL plan management
- Gain experience in SQL plan baselines
  - Take a look at using this feature **strategically**

More Information

- GitHub
  - Go to https://github.com/oracle/oracle-db-examples
    ... look for *optimizer*, *creative_w_plans*…
- Blogs
  - http://blogs.oracle.com/optimizer
  - https://MikeDietrichDE.com (upgrading)

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Hardware and Software
Engineered to Work Together


ORACLE®