ORACLE

Harnessing the Power of Optimizer Hints

Maria Colgan
Master Product Manager
Mission Critical Database Technologies
January 2020
@SQLMaria

SQLMaria

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, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Harnessing the power of Optimizer hints

Expectations

This session will not instantly make you an Optimizer hint expert!

Adding hints won't magically improve every query you encounter

Optimizer hints should only be used with extreme care

Program Agenda

1. What are hints?
2. How to use Optimizer hints
3. Useful Optimizer hints to know
4. Why are Optimizer hints ignored?
5. If you can hint it, baseline it
6. Managing an existing hinted application

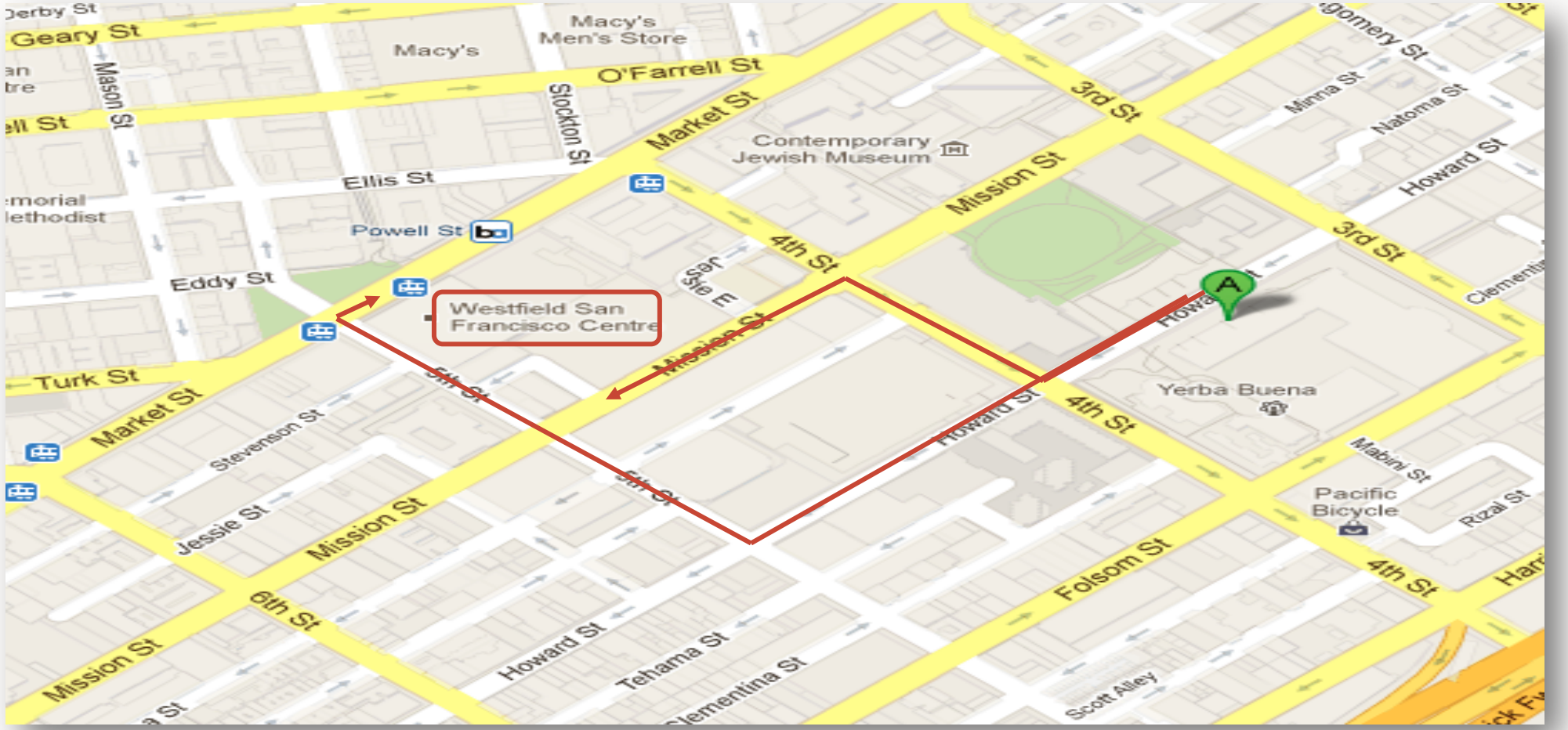
What are hints?

Overview

- Hints allow you to influence the Optimizer when it has to choose between several possibilities
- A hint is a directive that will be followed when applicable
- Can influence everything from Optimizer mode to every operation in plan
- Automatically means the Cost Based Optimizer will be used
  - Only exception is the RULE hint, but it must be used alone

What are hints?

Example - directions to the mall



What are hints?

Hints only evaluated when they apply to a decision that has to be made

Should I walk or drive to the mall?

- Best plan would be to walk

Should I go up 4th, 5th, or 6th street?

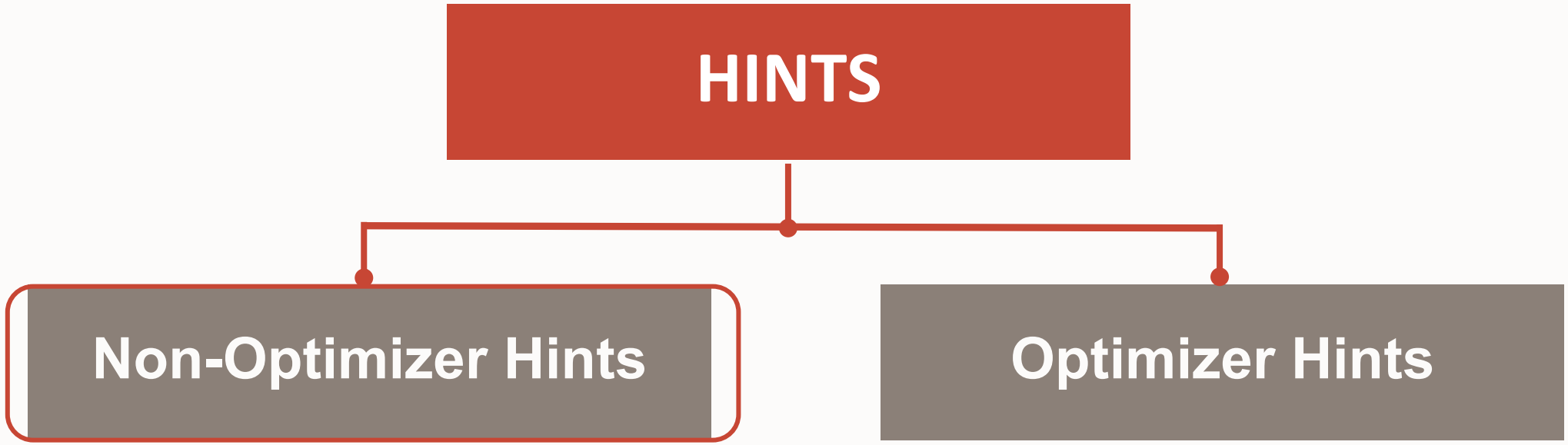
- Best plan would be to go up 4th street

Should I go in the front or the back door of the mall?

- Best plan would be to go in the back door

Telling me the cheapest parking is at 5th and Mission garage is irrelevant since I decided to walk

Two different classes of hints



Not all hints influence the Optimizer

Overview

The hint mechanism is not exclusively used by the Optimizer

Several other functional areas use hints too

- Direct path load can be controlled by `APPEND` hint
- Parallel statement queuing can be controlled by `STATEMENT_QUEUING` hint
- Data management in buffer cache can be influenced by `CACHE` hint
- What SQL statements get monitored by SQL Monitor can be controlled by `MONITOR` hint
- Use of In-Memory column store can be controlled by `INMEMORY` hint
- Take advantage of new fast ingest for IoT controlled by `MEMOPTIMIZE_WRITE`

Checking cardinality estimates

GATHER_PLAN_STATISTICS hint

```
SELECT /*+ GATHER_PLAN_STATISTICS*/
       p.prod_name,
        SUM(s.quantity_sold)
FROM   Products p, Sales s
WHERE  s.prod_id = p.prod_id
GROUP BY p.prod_name;
```

Checking cardinality estimates

GATHER_PLAN_STATISTICS hint

```
SELECT * FROM table(
DBMS_XPLAN.DISPLAY_CURSOR(FORMAT=>'ALLSTATS LAST'));
```

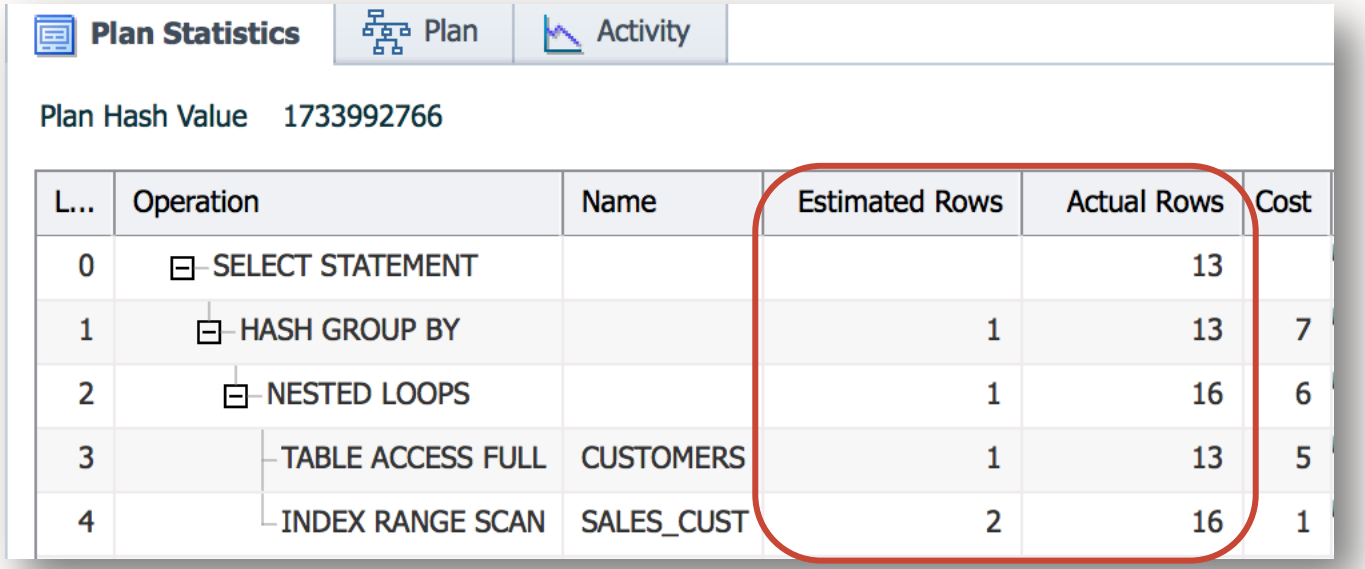
Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers	OMem	1Mem	Used-Mem
0	SELECT STATEMENT		1		71	00:00:00.57	1638			
1	HASH GROUP BY		1	71	71	00:00:00.57	1638	799K	799K	3079K (0)
* 2	HASH JOIN		1	918K	918K	00:00:00.85	1638	933K	933K	1279K (0)
3	TABLE ACCESS STORAGE FULL	PRODUCTS	1	72	72	00:00:00.01	3			
4	PARTITION RANGE ALL		1	918K	918K	00:00:00.37	1635			
5	TABLE ACCESS STORAGE FULL	SALES	28	918K	918K	00:00:00.20	1635			

- Compare estimated rows returned for each operation in plan to actual rows returned
- A-Time allows you to see where the time is spent

Checking cardinality estimates

MONITOR hint

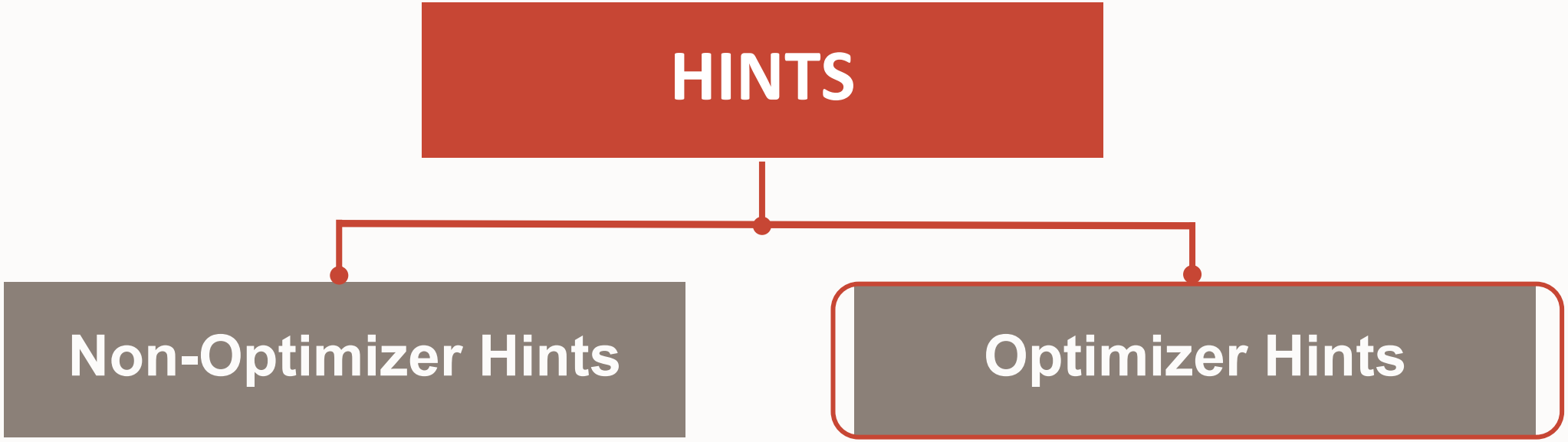
```
SELECT /*+ MONITOR*/ CUST_LASTNAME, SUM(AMOUNT_SOLD)
FROM   Customers c, Sales s
WHERE  s.cust_id = c.cust_id …
```



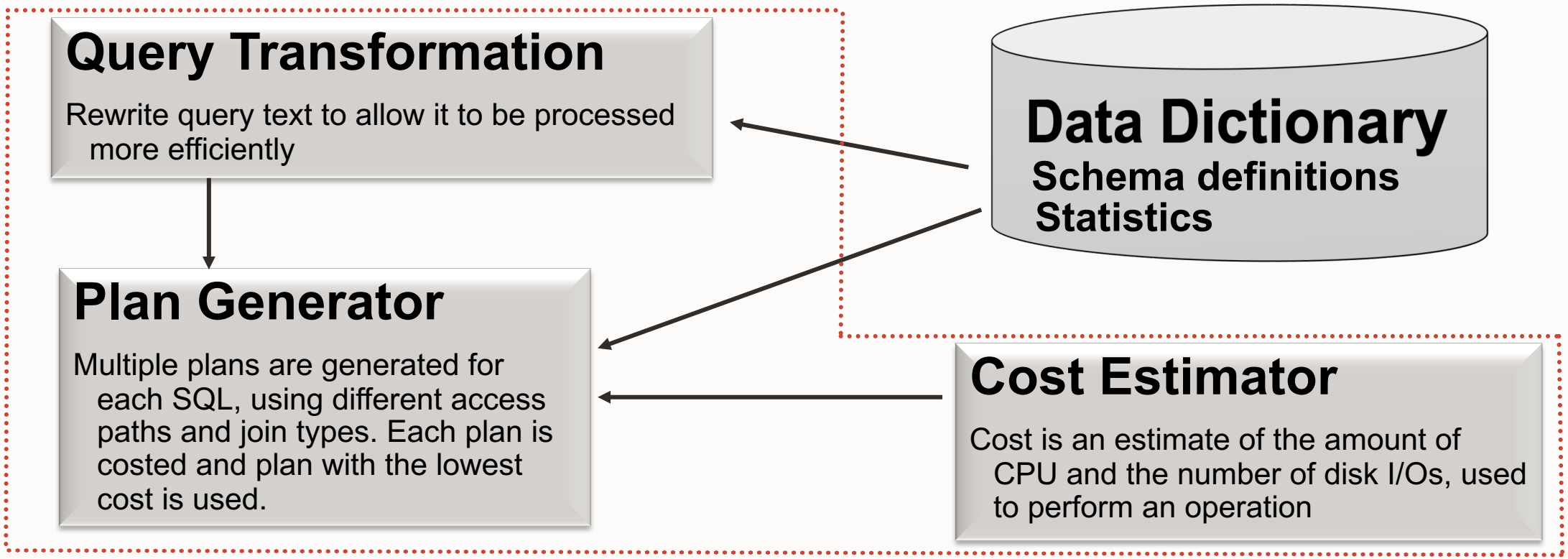
L...	Operation	Name	Estimated Rows	Actual Rows	Cost
0	SELECT STATEMENT			13	
1	HASH GROUP BY		1	13	7
2	NESTED LOOPS		1	16	6
3	TABLE ACCESS FULL	CUSTOMERS	1	13	5
4	INDEX RANGE SCAN	SALES_CUST	2	16	1

- Compare estimated rows returned for each operation in plan to actual rows returned
- Timeline allows you to see where the time is spent

Two different classes of hints



Inside the Oracle Optimizer



Hints influencing Query Transformations

Overview

First thing the Optimizer does is try to transform (rewrite) your statement

- The goal is to allow additional access or join methods and join orders to be used

Some transformations are always done but some are cost-based

Hints can be used to influence the transformations the Optimizer does

- NO_QUERY_TRANSFORMATION
- MERGE
- USE_CONCAT
- REWRITE
- STAR_TRANSFORMATION
- UNNEST

Hints can also influence all aspects of a plan

Overview

Hints to influence cardinality

- DYNAMIC_SAMPLING
- CARDINALITY

Hints to influence access paths

- FULL
- INDEX

Hints to influence join methods

- USE_NL_WITH_INDEX
- USE_HASH

Hints to influence join order

- LEADING
- ORDERED

Most hints have corresponding negative hint preceded by word ‘NO_’

More information on hints can be found in chapter 3 of SQL Reference Guide & chapter 19 of the SQL Tuning Guide

Hints Classifications

Overview

Single-table - hints that are specified on one table or view

- `FULL , INDEX` or `USE_NL`

Multi-table - hint that can be specified on one or more tables or views

- `LEADING` or `ORDERED`

Query block - hints that operate on single query blocks

- `STAR_TRANSFORMATION` or `UNNEST`

Statement – hints that apply to the entire SQL statement

- `ALL_ROWS` or `OPTIMIZER_FEATURES_ENABLE`

Program Agenda

1. What are hints?
2. How to use Optimizer hints
3. Useful Optimizer hints to know
4. Why are Optimizer hints ignored?
5. If you can hint it, baseline it
6. Managing an existing hinted application

How to use Optimizer hints

Overview

Hints are inserted in a SQL statement in the form of a comment with an additional + sign

They go immediately after the keyword (SELECT, INSERT, etc)

```
SELECT /* This is a comment */ count(*) FROM   Sales;

SELECT /*+ This is a hint   */ count(*) FROM   Sales;
```

Hint syntax is correct, but it is not a valid hint so is treated as comment

How to use Optimizer hints

Overview

Hints and comments can be combined

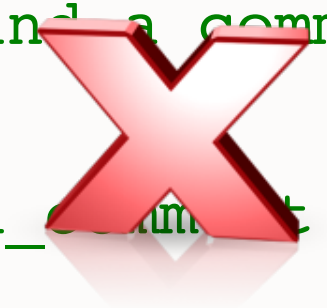
But best practice is to keep comment and hints in different blocks

- Comments can be put anywhere in a SQL statement not just after keyword

```
SELECT /*+ FULL(s) and a comment*/ count(*) FROM Sales s;

SELECT /*+ This_is_a_comment FULL(s) */ count(*) FROM Sales s;

SELECT /*+ FULL(s)*/ count(*) FROM Sales s /* comment */;
```



How to use Optimizer hints

Correctly identifying the object in the hint

Which one of the following hints will trigger the pk_emp index to be used in this query?

```
SELECT /*+ index(scott.emp pk_emp)*/ * FROM emp e;

SELECT /*+ index(emp pk_emp)*/ * FROM emp e;

SELECT /*+ index(pk_emp)*/ * FROM emp e;
```

None of them

How to use Optimizer hints

Correctly identifying the object in the hint

If you use a table alias in the query than you must specify the table alias name in the hint

Otherwise the hint is not valid

```
SELECT /*+ index(e pk_emp)*/ * FROM emp e;
```

How to use Optimizer hints

Hints only apply to the query block in which they appear

```
SELECT /*+ FULL(e) FULL(d) */ e.last_name, e.dept_id
FROM   employees e
WHERE e.dept_id in (SELECT d.dept_id
                    FROM    departments d
                     WHERE d.location_id=51);
```

```
-----------------------------------------------------------------------------
| Id  | Operation                    | Name             | Rows  |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                  |     5 |
|*  1 |  HASH JOIN                   |                  |     5 |
|   2 |   TABLE ACCESS BY INDEX ROWID| DEPARTMENTS      |       |
|*  3 |    INDEX RANGE SCAN          | DEPT_LOCATION_IX |     1 |
|   4 |   TABLE ACCESS FULL          | EMPLOYEES        |   107 |
-----------------------------------------------------------------------------
```

The dept table only appears in the sub-query, which is treated as separate query block.
Hint has no effect.

How to use Optimizer hints

Hints only apply to the query block in which they appear

```
SELECT /*+ FULL(e) */ e.last_name, e.dept_id
FROM   employees e
WHERE  e.dept_id in (SELECT /*+ FULL(d) */ d.dept_id
                     FROM   departments d
                     WHERE  d.location_id=51);
```

```
-------------------------------------------------------------------
| Id  | Operation          | Name        | Rows  | By
-------------------------------------------------------------------
|   0 | SELECT STATEMENT   |             |     5 |
|*  1 |  HASH JOIN         |             |       |
|*  2 |   TABLE ACCESS FULL| DEPARTMENTS |     1 |
|   3 |   TABLE ACCESS FULL| EMPLOYEES   |   107 |  1177 |     3   (0)|
-------------------------------------------------------------------
```

The hint on dept now has an effect as it appears in the correct query block, the sub-query

Only exception are statement level hints

How to use Optimizer hints

Query block names

Oracle automatically names each query block in a SQL statement

- sel$1, ins$2, upd$3, del$4, cri$5, mrg$6, set$7, misc$8
- Displayed using '+alias' format parameter in DBMS_XPLAN procedures

Query block names can be used to specify which block a hint applies to

- `/*+ FULL(@SEL$2 D) */`

The QB_NAME hint can be used to explicitly labels each query block

- `/*+ QB_NAME(my_name_for_block) */`

How to use Optimizer hints

Query block names

```
SELECT /*+ FULL(e) FULL(@MY_SUBQ d) */
       e.last_name, e.dept_id
FROM   employees e
WHERE  e.dept_id in (SELECT /*+ QB_NAME(MY_SUBQ) */
                            d.dept_id
                     FROM   departments d
                     WHERE  d.location_id=51);
```

How to use Optimizer hints

How do I know if my hints are used or not?

Any valid hint will be used

Can check if a hint is valid in hint section of 10053 trace

```
Dumping Hints
=============
  atom_hint=(@=0x124360178 err=0 resol=0 used=1 token=454 org=1 lvl=1 txt=ALL_ROWS )
  atom_hint=(@=0x2af785e0c260 err=0 resol=1 used=1 token=448 org=1 lvl=3 txt=FULL ("E") )
======================== END SQL Statement Dump ========================
```

ERR indicates if there is an error with hint

USED indicates the hint was used during the evaluation of the part of the plan it pertains to
Doesn't mean the final plan will reflect it

How to use Optimizer hints

Example showing how hints are used

SQL Statement

```
SELECT c.cust_name, sum (s.amount_sold)
FROM   customers c, sales s
WHERE  c.cust_id       = s.cust_id
AND    c.cust_city     = 'Los Angeles'
AND    c.cust_province = 'CA'
AND    s.time_id       = '09-SEP-18'
GROUP BY c.cust_name;
```

How to use Optimizer hints

Example showing how hints are used

Default plan is a hash join between sales and customers

```
-----------------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |       |       |   250 (100)|
|   1 |  HASH GROUP BY                |           |     1 |    64 |   250   (4)|
|*  2 |   HASH JOIN                   |           |     4 |   256 |   249   (4)|
|*  3 |    TABLE ACCESS STORAGE FULL  | CUSTOMERS |     3 |   138 |   226   (3)|
|   4 |    PARTITION RANGE SINGLE     |           |   535 |  9630 |    23  (18)|
|*  5 |     TABLE ACCESS STORAGE FULL | SALES     |   535 |  9630 |    23  (18)|
-----------------------------------------------------------------------------------
```

But we want the query to use a nested loops join

How to use Optimizer hints

Example showing how hints are used

Hinted SQL statement

```
SELECT /*+ USE_NL(s) */
       c.cust_name, sum (s.amount_sold)
FROM   customers c, sales s
WHERE  c.cust_id             = s.cust_id
AND    c.cust_city     = 'Los Angeles'
AND    c.cust_province = 'CA'
AND    s.time_id       = '09-SEP-18'
GROUP BY c.cust_name;
```

How to use Optimizer hints

Example showing how hints are used

Even with the hint we still get a hash join plan

```
-----------------------------------------------------------------------------------
| Id  | Operation                | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |           |       |       |     9 (100)|          |
|   1 |  HASH GROUP BY           |           |     1 |    52 |     9  (23)| 00:00:01 |
|*  2 |   HASH JOIN              |           |     1 |    52 |     8  (13)| 00:00:01 |
|   3 |    PARTITION RANGE SINGLE|           |     2 |    34 |     2   (0)| 00:00:01 |
|*  4 |     TABLE ACCESS FULL    | SALES     |     2 |    34 |     2   (0)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL     | CUSTOMERS |    13 |   455 |     5   (0)| 00:00:01 |
-----------------------------------------------------------------------------------
```

Why did it not use the hint?

How to use Optimizer hints

Example showing how hints are used

Let's look in the 10053 trace file

```
Dumping Hints
=============
  atom_hint=(@=0x13c6e7e20 err=0 resol=1 used=1 token=924 org=1 lvl=3 txt=USE_NL ("S") )
====================== END SQL Statement Dump ======================
```

Hint is valid and was used

Why did it not change the plan?

We only hinted the join method we didn’t hint the join order

Hint only valid when sales is on right side

Hint considered ONLY when join order was customers, sales

How to use Optimizer hints

Example showing how hints are used

NEW IN 19c

```
-------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |       |       |     9 (100)|          |
|   1 |  HASH GROUP BY               |           |     1 |    64 |     9  (23)| 00:00:01 |
|*  2 |   HASH JOIN                  |           |     4 |   256 |     8  (13)| 00:00:01 |
|   3 |    PARTITION RANGE SINGLE    |           |     3 |   138 |     2   (0)| 00:00:01 |
|*  4 |     TABLE ACCESS FULL        | SALES     |   535 |  9630 |     2   (0)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL         | CUSTOMERS |   535 |  9630 |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   2 - access("C"."CUST_ID"="S"."CUST_ID")
   4 - filter("S"."TIME_ID"='30-DEC-99')
   5 - filter(("C"."CUST_CITY"='Los Angeles' AND "CUST_S

Hint Report (identified by operation id / Query Block Name / Object Alias):
Total hints for statement: 1 (U - Unused (1))
---------------------------------------------------------------------------

   4 -  SEL$1 / S@SEL$1
         U -  USE_NL(s)
```

New Unused hint info under the plan in 19c with DBMS_XPLAN.DISPLAY_CURSOR

How to use Optimizer hints

Example showing how hints are used

Hinted SQL statement with both join method and join order hints

```
SELECT /*+ ORDERED USE_NL(s) */
       c.cust_name, sum (s.amount_sold)
FROM   customers c, sales s
WHERE  c.cust_id             = s.cust_id
AND    c.cust_city     = 'Los Angeles'
AND    c.cust_province = 'CA'
AND    s.time_id       = '09-SEP-17'
GROUP BY c.cust_name;
```

How to use Optimizer hints

Example showing how hints are used

Hinted plan

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		1	64	292 (7)
1	HASH GROUP BY		1	64	292 (7)
2	NESTED LOOPS		4	256	291 (6)
* 3	TABLE ACCESS STORAGE FULL	CUSTOMERS	3	138	226 (3)
4	PARTITION RANGE SINGLE		1	18	22 (19)
* 5	TABLE ACCESS STORAGE FULL	SALES	1	18	22 (19)

How to use Optimizer hints

Guaranteeing the same plan every time

Partial hints can't guarantee the same plan every time

Only way to guarantee the same plan every time is with a full outline

A full outline is a complete set of hints for all aspects of a plan

Full outline for a plan can be displayed using '+outline' option with `FORMAT` parameter in `DBMS_XPLAN.DISPLAY_CURSOR`

```
SELECT * FROM

TABLE(DBMS_XPLAN.display_cursor(format=>'+outline'));
```

How to use Optimizer hints

Guaranteeing the same plan every time

```
SQL> select * from table(dbms_xplan.display_cursor(format=>'TYPICAL +outline'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
SQL_ID  fbyb04j4qgpm8, child number 0
-------------------------------------
 select  e.empno, e.ename from emp e where empno <8000

Plan hash value: 169057108

-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |        |       |       |     2 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMP    |    14 |   140 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | PK_EMP |    14 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

Outline Data
-------------

  /*+
      BEGIN_OUTLINE_DATA
      IGNORE_OPTIM_EMBEDDED_HINTS
      OPTIMIZER_FEATURES_ENABLE('11.2.0.3')
      DB_VERSION('11.2.0.3')
      ALL_ROWS
      OUTLINE_LEAF(@"SEL$1")
      INDEX_RS_ASC(@"SEL$1" "E"@"SEL$1" ("EMP"."EMPNO"))
      END_OUTLINE_DATA
  */
```

Full outline for the plan

How to use Optimizer hints

Guaranteeing the same plan every time

```
SQL> Select /*+
  2         BEGIN_OUTLINE_DATA
  3         IGNORE_OPTIM_EMBEDDED_HINTS
  4         OPTIMIZER_FEATURES_ENABLE('11.2.0.3')
  5         DB_VERSION('11.2.0.3')
  6         ALL_ROWS
  7         OUTLINE_LEAF(@"SEL$1")
  8         FULL(@"SEL$1" "E"@"SEL$1")
  9         END_OUTLINE_DATA
 10         */
 11  e.empno, e.ename
 12  From   emp e
 13  Where  e.empno<8000;
```

Cut and paste full outline for the plan

Easier to maintain a full outline using SQL Plan Management

Program Agenda

1. What are hints?
2. How to use Optimizer hints
3. Useful Optimizer hints to know
4. Why are Optimizer hints ignored?
5. If you can hint it, baseline it
6. Managing an existing hinted application

Changing the Optimizer mode

The following hints control the Optimizer mode

- `ALL_ROWS (default mode)`
- `FIRST_ROWS(n)`
- `RULE`

`FIRST_ROWS(n)` choose plan that returns the first n rows most efficiently

- Use of old `FIRST_ROWS` hint is not recommended
  - Not fully cost based

`RULE` hint reverts back to Rule Based Optimizer (RBO)

- Not recommended as RBO is de-supported and severely limits plan options

Changing the Optimizer mode

Default Optimizer mode example

45% of the rows in the employee table have department_id = 50

Default plan is a full table scan

```
SELECT e.emp_id, e.last_name, e.salary
FROM   employees e
WHERE e.dept_id = 50;
```

```
-------------------------------------------------------------------------
| Id  | Operation         | Name      | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |           |    45 |   855 |     3   (0)|
|*  1 |  TABLE ACCESS FULL| EMPLOYEES |    45 |   855 |     3   (0)|
-------------------------------------------------------------------------
```

Changing the Optimizer mode

FIRST_ROWS(n) hint example

```
SELECT /*+ FIRST_ROWS(10) */
       e.emp_id, e.last_name, e.salary
FROM   employees e
WHERE e.dept_id = 50;
```

Plan changed because the assumption is you are going to stop fetching after first 10 rows

```
-------------------------------------------------------------------------------
| Id  | Operation                    | Name              | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                   |    10 |   190 |     2   (0)|
|   1 |  TABLE ACCESS BY INDEX ROWID| EMPLOYEES         |    10 |   190 |     2   (0)|
|*  2 |   INDEX RANGE SCAN           | EMP_DEPARTMENT_IX |       |       |     1   (0)|
-------------------------------------------------------------------------------
```

Changing the Optimizer mode

RULE hint

RULE hint specifies that the Rule Based Optimizer (RBO) be used

The RULE hint is not applicable if

- Other hints are specified in the stmt
- One or more tables are partitioned
- One or more IOTs are used
- One or more Materialized views exist
- A SAMPLE clauses is specified
- A spreadsheet clause is specified
- Parallel execution is used
- Grouping sets are used
- Group outer-join is used
- A create table with a parallel clause
- A left or full outer join (ANSI) is specified
- Flashback cursor (AS OF) is used
- .....

Changing the Optimizer mode

Rule hint

RULE hint ignored when partitioned table is used

```
SELECT /*+ RULE */ count(*)
FROM   sales s;
```

```
-------------------------------------------------------------------------------------------------------
| Id  | Operation                 | Name              | Rows  | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |                   |       |    12 (100)|          |       |       |
|   1 |  SORT AGGREGATE           |                   |     1 |            |          |       |       |
|   2 |   PARTITION RANGE ALL     |                   |   960 |    12   (0)| 00:00:01 |     1 |    16 |
|   3 |    BITMAP CONVERSION COUNT|                   |   960 |    12   (0)| 00:00:01 |       |       |
|   4 |     BITMAP INDEX FULL SCAN| SALES_CHANNEL_BIX |       |            |          |     1 |    16 |
-------------------------------------------------------------------------------------------------------
```

RULE hint is ignored because SALES is a partitioned table therefore CBO is used

Changing the Optimizer mode

RULE hint

RULE hint prevents bitmap index being used and triggers full scan

```
SELECT /*+ RULE */ count(*)
FROM   non_partitioned_sales s;
```

```
-----------------------------------------------------
| Id  | Operation           | Name                  |
-----------------------------------------------------
|   0 | SELECT STATEMENT    |                       |
|   1 |  SORT AGGREGATE     |                       |
|   2 |   TABLE ACCESS FULL| NON_PARTITIONED_SALES  |
-----------------------------------------------------

Note
-----
   - rule based optimizer used (consider using cbo)
```

Changing initialization parameter for a query

OPT_PARAM hint

- Allows init.ora Optimizer parameters to be changed for a specific query
- Useful way to prevent setting non-default parameter value system-wide
- Only the following Optimizer influencing init.ora parameters can be set:
  - OPTIMIZER_DYNAMIC_SAMPLING
  - OPTIMIZER_INDEX_CACHING
  - OPTIMIZER_INDEX_COST_ADJ
  - OPTIMIZER_USE_PENDING_STATISTICS
  - OPTIMIZER_USE_INVISIBLE_INDEXES
  - OPTIMIZER_SECURE_VIEW_MERGING
  - Optimizer related underscore parameters
  - STAR_TRANSFORMATION_ENABLED
  - PARALLEL_DEGREE_POLICY
  - PARALLEL_DEGREE_LIMIT
  - OPTIMIZER_ADAPTIVE_PLANS
  - OPTIMIZER_ADAPTIVE_REPORTING_ONLY
  - OPTIMIZER_ADAPTIVE_STATISTICS
  - OPTIMIZER_INMEMORY_AWARE

Changing initialization parameter for a query

OPT_PARAM hint example

```
SQL> Select count(*)
  2  From  employees e
  3  Where e.job_id='SA_REP'
  4  And   (e.salary*e.commission_pct)*12 > 13000;

  COUNT(*)
----------
        22

SQL>
SQL> select * from table(dbms_xplan.display_cursor());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------
SQL_ID  4vfmg488y8q0w, child number 0
-------------------------------------
Select count(*) From  employees e Where e.job_id='SA_REP' And
(e.salary*e.commission_pct)*12 > 13000

Plan hash value: 1756381138

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |       |       |     2 (100)|          |
|   1 |  SORT AGGREGATE              |           |     1 |    22 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| EMPLOYEES |     2 |    44 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Cardinality under-estimated due to complex expression
Extended statistics would help

Changing initialization parameter for a query

OPT_PARAM hint example

```
SQL> Select /*+ OPT_PARAM('OPTIMIZER_USE_PENDING_STATISTICS' 'TRUE') */ count(*)
  2  From  employees e
  3  Where e.job_id='SA_REP'
  4  And   (e.salary*e.commission_pct)*12 > 13000;

  COUNT(*)
----------
        22

SQL>
SQL> select * from table(dbms_xplan.display_cursor());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
SQL_ID  0b37sbu2rr7xp, child number 1
-------------------------------------
Select /*+ OPT_PARAM('OPTIMIZER_USE_PENDING_STATISTICS' 'TRUE') */
count(*) From  employees e Where e.job_id='SA_REP' And
(e.salary*e.commission_pct)*12 > 13000

Plan hash value: 1756381138

-------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |       |       |     2 (100)|          |
|   1 |  SORT AGGREGATE              |           |     1 |    12 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| EMPLOYEES |    22 |   264 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

Create extended statistics & re-gather statistics as pending statistics
OPT_PARAM hint enables pending statistics for only this statement

Changing Optimizer features enable

This parameter gets it own hint

`OPTIMIZER_FEATURES_ENABLE` parameter allows you to switch between optimizer versions

Setting it to previous database version reverts the Optimizer to that version

- Disables any functionality that was not present in that version

Easy way to work around unexpected behavior in a new release

Hint allows you to revert the Optimizer for just a single statement

Changing Optimizer features enable

Example

```
SQL> explain plan for
  2  Select object_type, count(*)
  3  From t
  4  Group by object_type;

Explained.

SQL>
SQL> select * from table(dbms_xplan.display(format=>'+outline'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 47235625

--------------------------------------------------------------------------------
| Id  | Operation                  | Name | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |      |    49 |   490 |   212   (8)| 00:00:01 |
|   1 |  HASH GROUP BY             |      |    49 |   490 |   212   (8)| 00:00:01 |
|   2 |   TABLE ACCESS STORAGE FULL| T    | 88766 |   866K|   201   (2)| 00:00:01 |
--------------------------------------------------------------------------------
```

Changing Optimizer features enable

Example

```
SQL> explain plan for
  2  Select /*+ optimizer_features_enable('9.2.0') */ object_type, count(*)
  3  From t
  4  Group by object_type;

Explained.

SQL>
SQL> select * from table(dbms_xplan.display(format=>'+outline'));

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1476560607

-------------------------------------------------------------------------
| Id  | Operation                  | Name | Rows  | Bytes | Cost  |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |      |    49 |   490 |   363 |
|   1 |  SORT GROUP BY             |      |    49 |   490 |   363 |
|   2 |   TABLE ACCESS STORAGE FULL| T    | 88766 |   866K|   202 |
-------------------------------------------------------------------------
```

Hash GROUP BY introduced in 10g not an option for 9.2 Optimizer, so traditional sort based GROUP BY selected

Program Agenda

1. What are hints?
2. How to use Optimizer hints
3. Useful Optimizer hints to know
4. Why are Optimizer hints ignored?
5. If you can hint it, baseline it
6. Managing an existing hinted application

Why are Optimizer hints ignored?

Syntax and Spelling

Which one of the following hints will trigger the pk_emp index to be used in this query?

```
SELECT /*+ IND(e pk_emp)*/ * FROM employees e;
```

```
SELECT /*+ INDEX(e emp_pk)*/ * FROM employees e;
```

```
SELECT /*+ INDEX(e pk_emp)*/ * FROM employees e;
```

Why are Optimizer hints ignored?

Invalid hint

Specifying an index hint on a table with no indexes

```
SELECT /*+ INDEX(p) */ Count(*)
FROM   my_promotions p
WHERE  promo_category = 'TV'
AND    promo_begin_date = '05-OCT-17';
```

Invalid hint because no indexes exist on the table

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |       |       |     3 (100)|          |
|   1 |  SORT AGGREGATE    |               |     1 |    15 |            |          |
|*  2 |   TABLE ACCESS FULL| MY_PROMOTIONS |     1 |    15 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Why are Optimizer hints ignored?

Illegal hint

Specifying a hash join hint for non-equality join

```
SELECT /*+ USE_HASH(e s) */ e.first_name, e.last_name
FROM   employees e, salary_grade s
WHERE  e.salary BETWEEN s.low_sal AND s.high_sal;
```

```
-------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |     1 |    45 |     4   (0)|
|   1 |  NESTED LOOPS                |              |     1 |    45 |     4   (0)|
|*  2 |   TABLE ACCESS STORAGE FULL| SALARY_GRADE |     1 |    26 |     2   (0)|
|*  3 |   TABLE ACCESS STORAGE FULL| EMPLOYEES    |     1 |    19 |     2   (0)|
-------------------------------------------------------------------------------
```

Illegal hint because a hash join can't be used for a non-equality join predicate

Why are Optimizer hints ignored?

Invalid hint combinations

Specifying a parallel hint for an index range scan

```
SELECT /*+ index(e empno_pk_ind) parallel(e 8) */
       e.empno, e.name
FROM   employees e
WHERE  e.empno < 7700;
```

Invalid hint combination because an index range scan can't be parallelized on non-partitioned table

```
-------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Rows  | Bytes |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |     8 |    80 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP           |     8 |    80 |
|*  2 |   INDEX RANGE SCAN                  | EMPNO_PK_IND  |     8 |       |
-------------------------------------------------------------------------------
```

Why are Optimizer hints ignored?

Invalid hint combinations

If two hints contradict each other, they will both be ignored

```
SELECT /*+ full(e) index(e empno_fk_ind) */
       e.empno, e.name
FROM   employees e
WHERE  e.empno < 7700;
```

Conflicting hints you can't do a full table scan and index lookup on same table

```
-------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Rows  | Bytes |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |     8 |    80 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| EMP           |     8 |    80 |
|*  2 |   INDEX RANGE SCAN                  | EMPNO_PK_IND  |     8 |       |
-------------------------------------------------------------------------------
```

Why are Optimizer hints ignored?

Hint becomes invalid due to transformation

Ordered hint dictates the join order as the order of tables in FROM clause

```
SELECT /*+ ordered */ e1.last_name,
j.job_title, e1.salary, v.avg_salary
FROM      employees e1, jobs j
          ( SELECT    e2.dept_id, avg(e2.salary) avg_salary
            FROM      employees e2,  departments d
            WHERE     d.location=1700 AND e2.depet_id=d.dept_id
            GROUP BY e2.dept_id)v
WHERE     e1.job_id = j.job_id
AND       e1.dept_id = v.dept_id
AND       e1.salary > a.avg_salary
ORDER BY e1.last_name;
```

Expected join order

1. Employees
2. Jobs
3. V

Why are Optimizer hints ignored?

Hint becomes invalid due to transformation

View merging occurred
Order of tables in FROM clause (e1,j,v) lost
Optimizer picks join order with lowest cost

Actual join order used

Id	Operation	Name		
0	SELECT STATEMENT			
*1	FILTER			
2	SORT GROUP BY		1	83
*3	HASH JOIN		3265	264K
4	4 TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	21	147
*5	INDEX RANGE SCAN	DEPT_LOCATION_IX	21	
*6	HASH JOIN		3296	244K
7	3 TABLE ACCESS STORAGE FULL	EMPLOYEES	107	749
*8	HASH JOIN		107	7383
9	1 TABLE ACCESS STORAGE FULL	EMPLOYEES	107	3852
10	2 TABLE ACCESS STORAGE FULL	JOBS	19	627

Why are Optimizer hints ignored?

Hint becomes invalid due to transformation

NO_MERGE hint prevents transformation from taking place

```
SELECT /*+ no_merge(v) ordered */
          e1.last_name, j.job_title, e1.salary, v.avg_salary
FROM      employees e1, jobs j
          ( SELECT    e2.dept_id, avg(e2.salary) avg_salary
            FROM      employees e2,  departments d
            WHERE     d.location=1700 AND e2.depet_id=d.dept_id
            GROUP BY e2.dept_id)v
WHERE     e1.job_id = j.job_id
AND       e1.dept_id = v.dept_id
AND       e1.salary > a.avg_salary
ORDER BY e1.last_name;
```

Preserves FROM clause order

Why are Optimizer hints ignored?

Hint becomes invalid due to transformation

Actual join order used

```
-------------------------------------------------------------------------------
| Id  | Operation                      | Name                | Rows  | Bytes |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                     |       |       |
|   1 |  SORT ORDER BY                 |                     |    17 |  1139 |
|*  2 |   HASH JOIN                    |                     |    17 |  1139 |
|*  3 |    HASH JOIN                   |                     |   107 |  5457 |
|   4 |  1 TABLE ACCESS STORAGE FULL   | EMPLOYEES           |   107 |  2568 |
|   5 |  2 TABLE ACCESS STORAGE FULL   | JOBS                |    19 |   513 |
|   6 |  3 VIEW                        |                     |    11 |   176 |
|   7 |     HASH GROUP BY              |                     |    11 |   154 |
|   8 |      NESTED LOOPS              |                     |       |       |
|   9 |       NESTED LOOPS             |                     |    37 |   518 |
|  10 |        TABLE ACCESS BY INDEX ROWID| DEPARTMENTS      |     4 |    28 |
|* 11 |         INDEX RANGE SCAN       | DEPT_LOCATION_IX    |     4 |       |
|* 12 |        INDEX RANGE SCAN        | EMP_DEPARTMENT_IX   |    10 |       |
|  13 |       TABLE ACCESS BY INDEX ROWID | EMPLOYEES        |    10 |    70 |
-------------------------------------------------------------------------------
```

Inline View v

Program Agenda

1. What are hints?
2. How to use Optimizer hints
3. Useful Optimizer hints to know
4. Why are Optimizer hints ignored?
5. If you can hint it, baseline it
6. Managing an existing hinted application

If you can hint it, baseline it

Alternative approach to hints

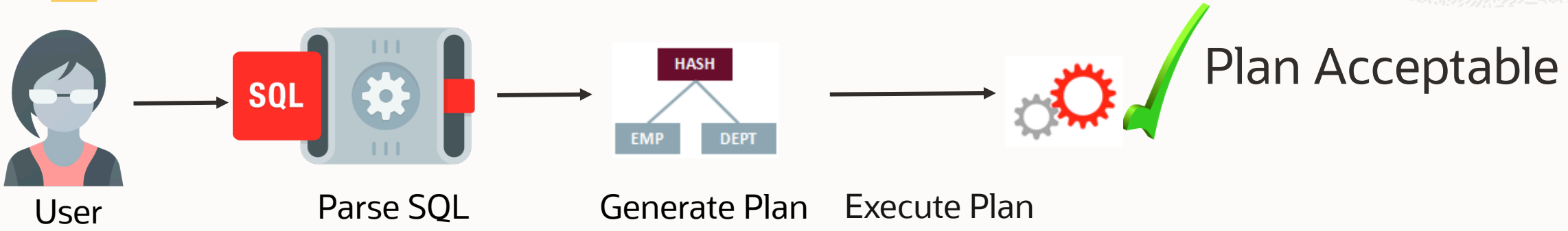
- It is not always possible to add hints to third party applications
- Hints can be extremely difficult to manage over time
- Once added never removed

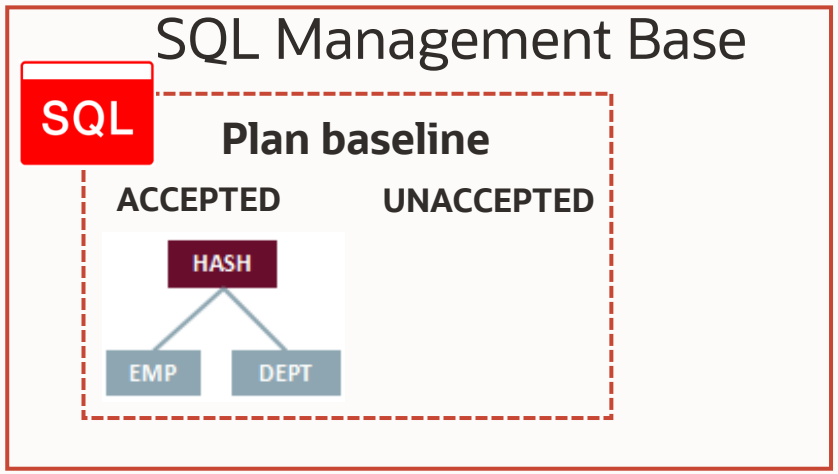
Solution

- Use SQL Plan Management (SPM)
- Influence the execution plan without adding hints directly to queries
- SPM available in Enterprise Edition*, no additional options required

If you can hint it, baseline it

SQL Plan Management

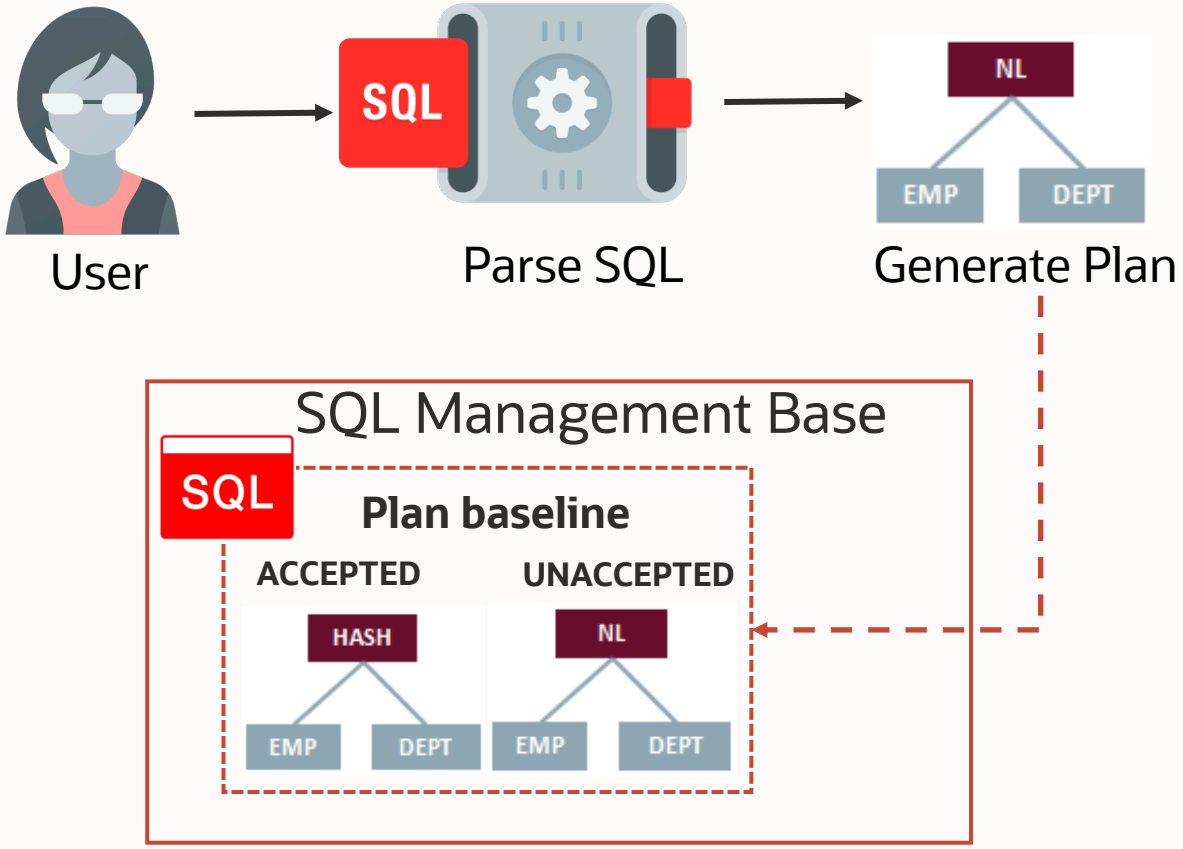




NOTE:: Actual execution plans stored in SQL plan baseline in Oracle Database 12c

If you can hint it, baseline it

SQL Plan Management

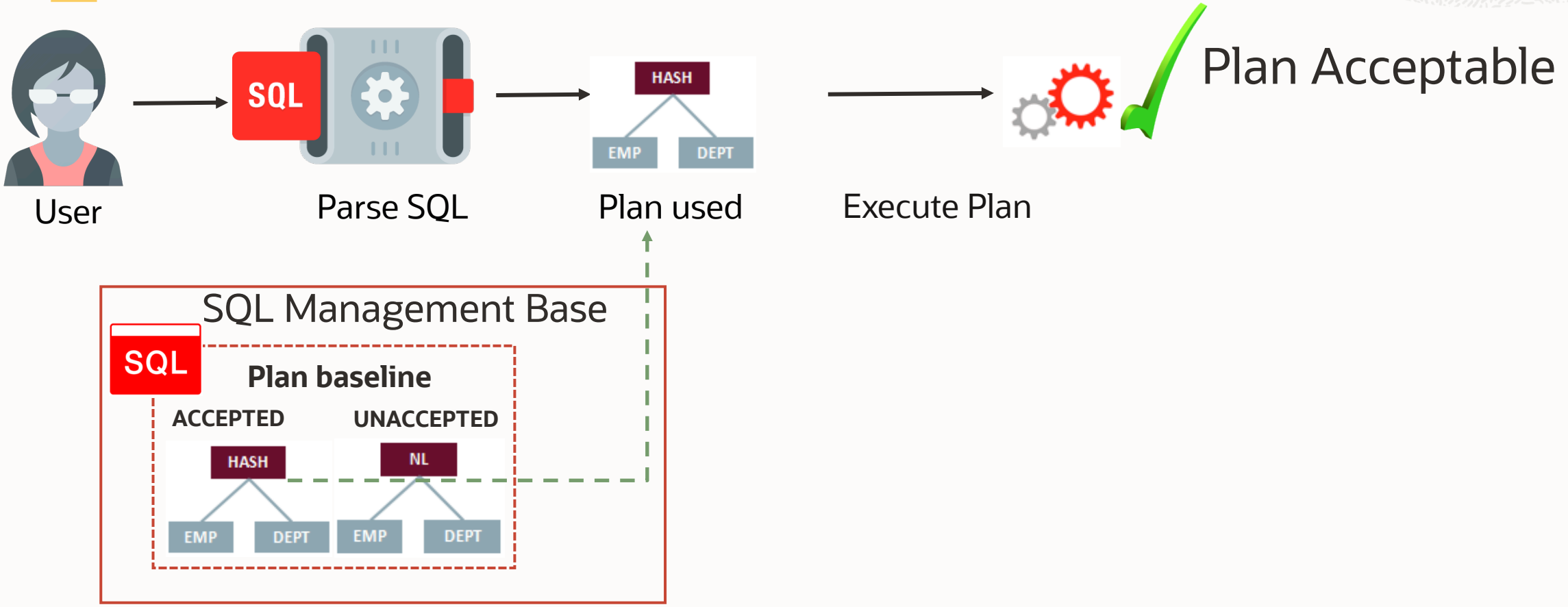


NOTE:: You **do not** need to be in auto-capture mode to have a new plan added to an existing SQL plan baseline

Additional fields such as fetches, row processed etc. are not populated because new plan has never executed

If you can hint it, baseline it

SQL Plan Management

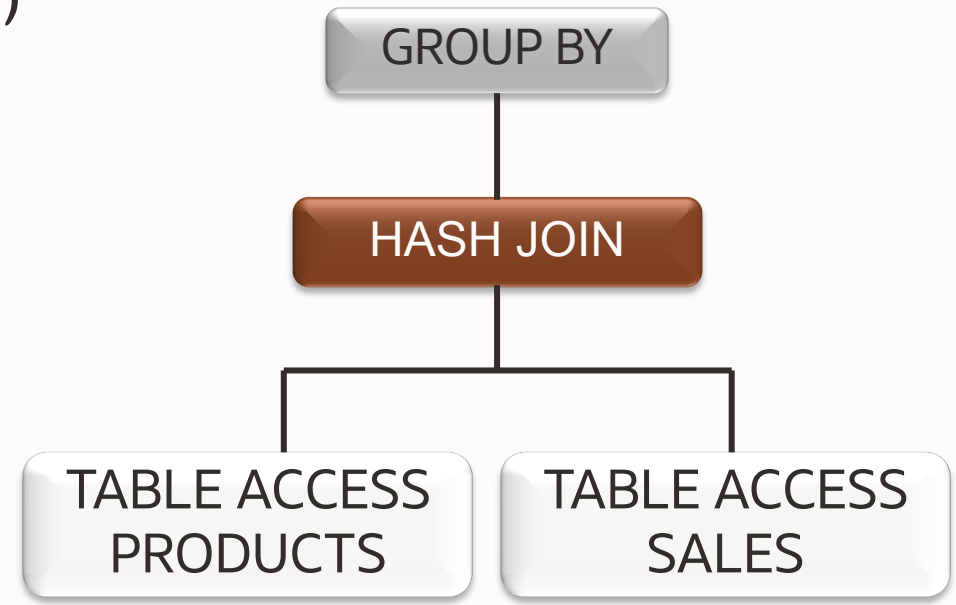


Influence execution plan without adding hints

Example Overview

Simple two table join between the SALES and PRODUCTS tables

```
SELECT p.prod_name,SUM(s.amount_sold)
FROM   products p, sales s
WHERE  p.prod_id = s.prod_id
AND    p.supplier_id = :sup_id
GROUP  BY p.prod_name;
```



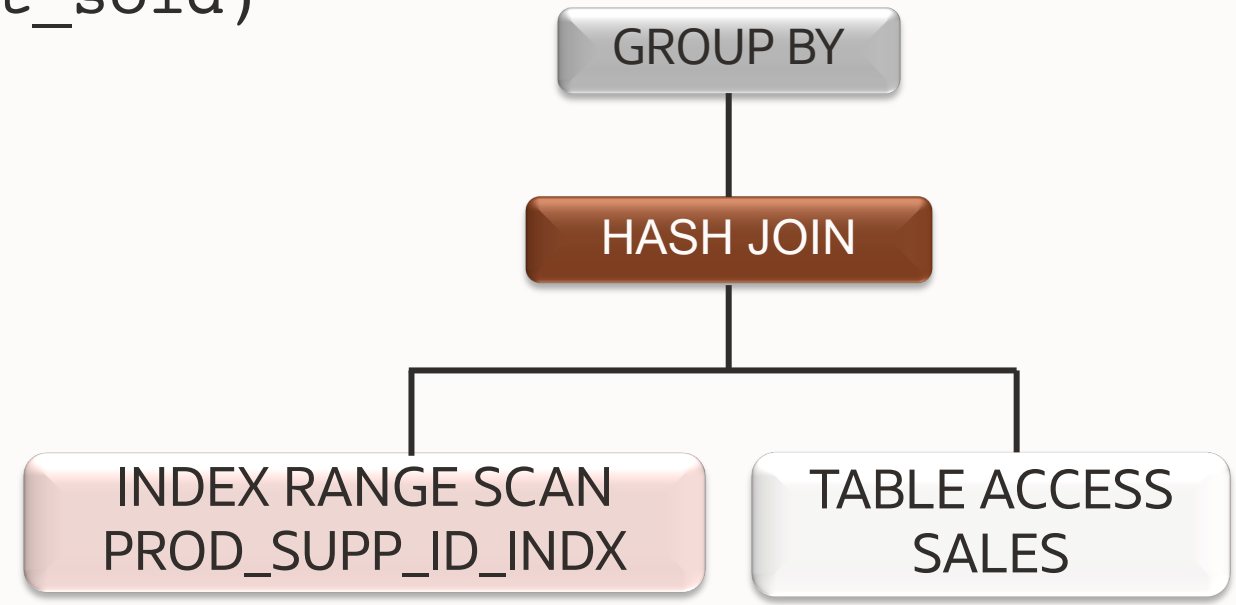
Current Plan

Influence execution plan without adding hints

Example Overview

Simple two table join between the SALES and PRODUCTS tables

```
SELECT p.prod_name,SUM(s.amount_sold)
FROM   products p, sales s
WHERE  p.prod_id = s.prod_id
AND    p.supplier_id = :sup_id
GROUP  BY p.prod_name;
```



Desired Plan

Influence execution plan without adding hints

Step 1. Execute the non-hinted SQL statement

```
SELECT p.prod_name,SUM(s.amount_sold)
FROM   products p, sales s
WHERE  p.prod_id = s.prod_id
AND    p.supplier_id = :sup_id
GROUP  BY p.prod_name;
```

```
PROD_NAME                                  SUM(S.AMOUNT_SOLD)
-----------------------------------------  ------------------
Baseball trouser Kids                                      91
Short Sleeve Rayon Printed Shirt $8.99                     32
```

Influence execution plan without adding hints

Default plan uses full table scans followed by a hash join

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------
SQL_ID  akuntdurat7yr, child number 0
-------------------------------------
SELECT p.prod_name, sum(s.amount_sold) FROM   products p , sales s
WHERE  p.prod_id = s.prod_id AND    p.supplier_id = :sup_id GROUP BY
p.prod_name

Plan hash value: 3535171836

------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)|
------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |       |       |    15 (100)|
|   1 |  HASH GROUP BY        |          |     2 |    90 |    15   (7)|
|*  2 |   HASH JOIN           |          |     3 |   135 |    14   (0)|
|*  3 |    TABLE ACCESS FULL  | PRODUCTS |     2 |    72 |     9   (0)|
|   4 |    PARTITION RANGE ALL|          |   960 |  8640 |     5   (0)|
|   5 |     TABLE ACCESS FULL | SALES    |   960 |  8640 |     5   (0)|
------------------------------------------------------------------------

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

   2 - access("P"."PROD_ID"="S"."PROD_ID")
   3 - filter("P"."SUPPLIER_ID"=:SUP_ID)
```

Influence execution plan without adding hints

Step 2. Find the SQL_ID for the non-hinted statement in V$SQL

```
SELECT sql_id,
       sql_fulltext
FROM   v$sql
WHERE  sql_text LIKE 'SELECT p.prod_name, %';
```

```
SQL_ID          SQL_FULLTEXT
-------------   ----------------------------------------
akuntdurat7yr   SELECT p.prod_name, SUM(s.amount_sold)
                FROM   products p , sales s
                WHERE  p.prod
```

Influence execution plan without adding hints

Step 3. Create a SQL plan baseline for the non-hinted SQL statement

```
VARIABLE cnt NUMBER
EXECUTE :cnt := dbms_spm.load_plans_from_cursor_cache(sql_id=>'akuntdurat7yr');

PL/SQL PROCEDURE successfully completed.
```

```
SELECT sql_handle, sql_text, plan_name, enabled
FROM   dba_sql_plan_baselines
WHERE  sql_text LIKE 'SELECT p.prod_name, %';
```

SQL_HANDLE	SQL_TEXT	PLAN_NAME	ENA
SQL_8f876d84821398cf	SELECT p.prod_name, sum(s.amount_sold) FROM products p , sales s	SQL_PLAN_8z1vdhk1176g42949306	YES

Influence execution plan without adding hints

Step 4. Disable plan in SQL plan baseline for the non-hinted SQL statement

```
EXECUTE :cnt := dbms_spm.alter_sql_plan_baseline(sql_handle=>'SQL_8f876d84821398cf',-
                                                 plan_name=>'SQL_PLAN_8z1vdhk11766g42949306',-
                                                 attribute_name => 'enabled', -
                                                 attribute_value => 'NO');

PL/SQL PROCEDURE successfully completed.

SELECT sql_handle, sql_text, plan_name, enabled
FROM   dba_sql_plan_baselines
WHERE  sql_text LIKE 'SELECT p.prod_name, %';
SQL_HANDLE          SQL_TEXT                                        PLAN_NAME                 ENA
------------------- ----------------------------------------------- ------------------------- ---
SQL_8f876d84821398cf SELECT p.prod_name, sum(s.amount_sold)          SQL_PLAN_8z1vdhk1176       NO
                     FROM   products p , sales s                     g42949306
```

Influence execution plan without adding hints

Step 5. Manually modify the SQL statement to use the hint(s) & execute it

```
SELECT /*+ index(p) */ p.prod_name,SUM(s.amount_sold)
FROM   products p, sales s
WHERE  p.prod_id = s.prod_id
AND    p.supplier_id = :sup_id
GROUP  BY p.prod_name;
```

```
PROD_NAME                                  SUM(S.AMOUNT_SOLD)
-----------------------------------------  ------------------
Baseball trouser Kids                                      91
Short Sleeve Rayon Printed Shirt $8.99                     32
```

Influence execution plan without adding hints

Step 6. Find SQL_ID & PLAN_HASH_VALUE for hinted SQL stmt in V$SQL

```
SELECT sql_id, plan_hash_value,  sql_fulltext
FROM   v$sql
WHERE  sql_text LIKE 'SELECT /*+ index(p) */ p.prod_name, %';
```

```
SQL_ID         PLAN_HASH_VLAUE SQL_FULLTEXT
-------------- --------------- --------------------------------
avph0nnq5pfc2  2567686925      SELECT /*+ index(p) */ p.prod_name,
                               SUM( s.amount_sold) FROM products p,
                               sales
```

Influence execution plan without adding hints

Step 7. Associate hinted plan with original SQL stmt's SQL HANDLE

```
VARIABLE cnt NUMBER
EXECUTE :cnt := dbms_spm.load_plans_from_cursor_cache(sql_id=>'avph0nnq5pfc2',-
                                                      plan_hash_value=>'2567686925', -
                                                      sql_handle=>'SQL_8f876d84821398cf');

PL/SQL PROCEDURE successfully completed.
```

SQL_ID & PLAN_HASH_VALUE belong to hinted statement
SQL_HANDLE is for the non-hinted statement

Influence execution plan without adding hints

Step 8. Confirm SQL statement has two plans in its SQL plan baseline

```
SELECT sql_handle, sql_text, plan_name, enabled
FROM   dba_sql_plan_baselines
WHERE  sql_text LIKE 'SELECT p.prod_name, %';
```

```
SQL_HANDLE          SQL_TEXT                                        PLAN_NAME                ENA
------------------- ----------------------------------------------- ------------------------ ---
SQL_8f876d84821398cf SELECT p.prod_name, sum(s.amount_sold)          SQL_PLAN_8z1vdhk1176      NO
                     FROM   products p , sales s                     g42949306

SQL_8f876d84821398cf SELECT p.prod_name, sum(s.amount_sold)          SQL_PLAN_8z1vdhk1176      YES
                     FROM   products p , sales s                     6ge1c67f67
```

Hinted plan is the only enabled plan for non-hinted SQL statement

Influence execution plan without adding hints

Step 9. Confirm hinted plan is being used

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------
SQL_ID  akuntdurat7yr, child number 0
-------------------------------------
SELECT p.prod_name, sum(s.amount_sold) FROM   products p , sales s
WHERE  p.prod_id = s.prod_id AND    p.supplier_id = :sup_id GROUP BY
p.prod_name

Plan hash value: 2567686925

------------------------------------------------------------------------------------
| Id  | Operation             | Name               | Rows  | Bytes | Cost (%CPU)|
------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                    |       |       |     8 (100)|
|   1 |  HASH GROUP BY        |                    |     2 |    90 |     8  (13)|
|*  2 |   HASH JOIN           |                    |     3 |   135 |     7   (0)|
|*  3 |    INDEX RANGE SCAN   | PROD_SUPP_ID_INDEX |     2 |    72 |     2   (0)|
|   4 |    PARTITION RANGE ALL|                    |   960 |  8640 |     5   (0)|
|   5 |     TABLE ACCESS FULL | SALES              |   960 |  8640 |     5   (0)|
------------------------------------------------------------------------------------

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

   2 - access("P"."PROD_ID"="S"."PROD_ID")
   3 - access("P"."SUPPLIER_ID"=:SUP_ID)

Note
-----
   - SQL plan baseline SQL_PLAN_8z1vdhk11766ge1c67f67 used for this statement
```

Non-hinted SQL text but it is using the plan hash value for the hinted statement

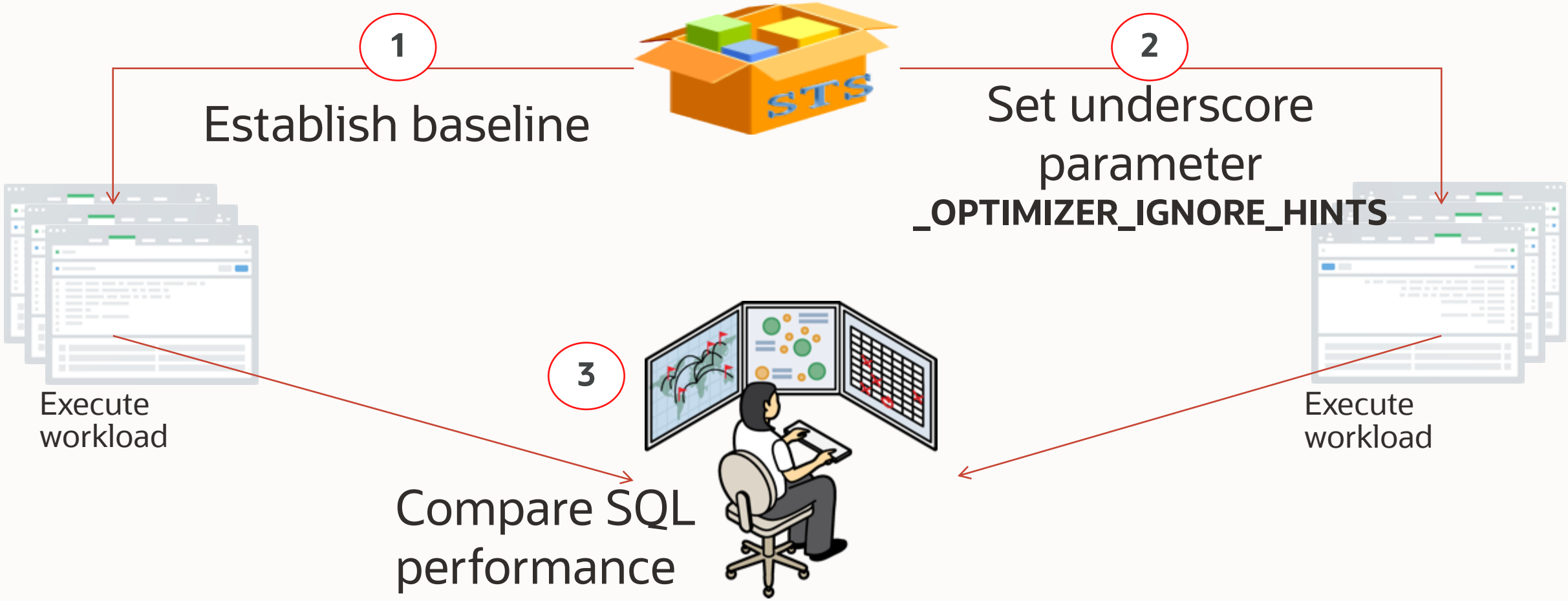
Note section also confirms SQL plan baseline used for statement

Program Agenda

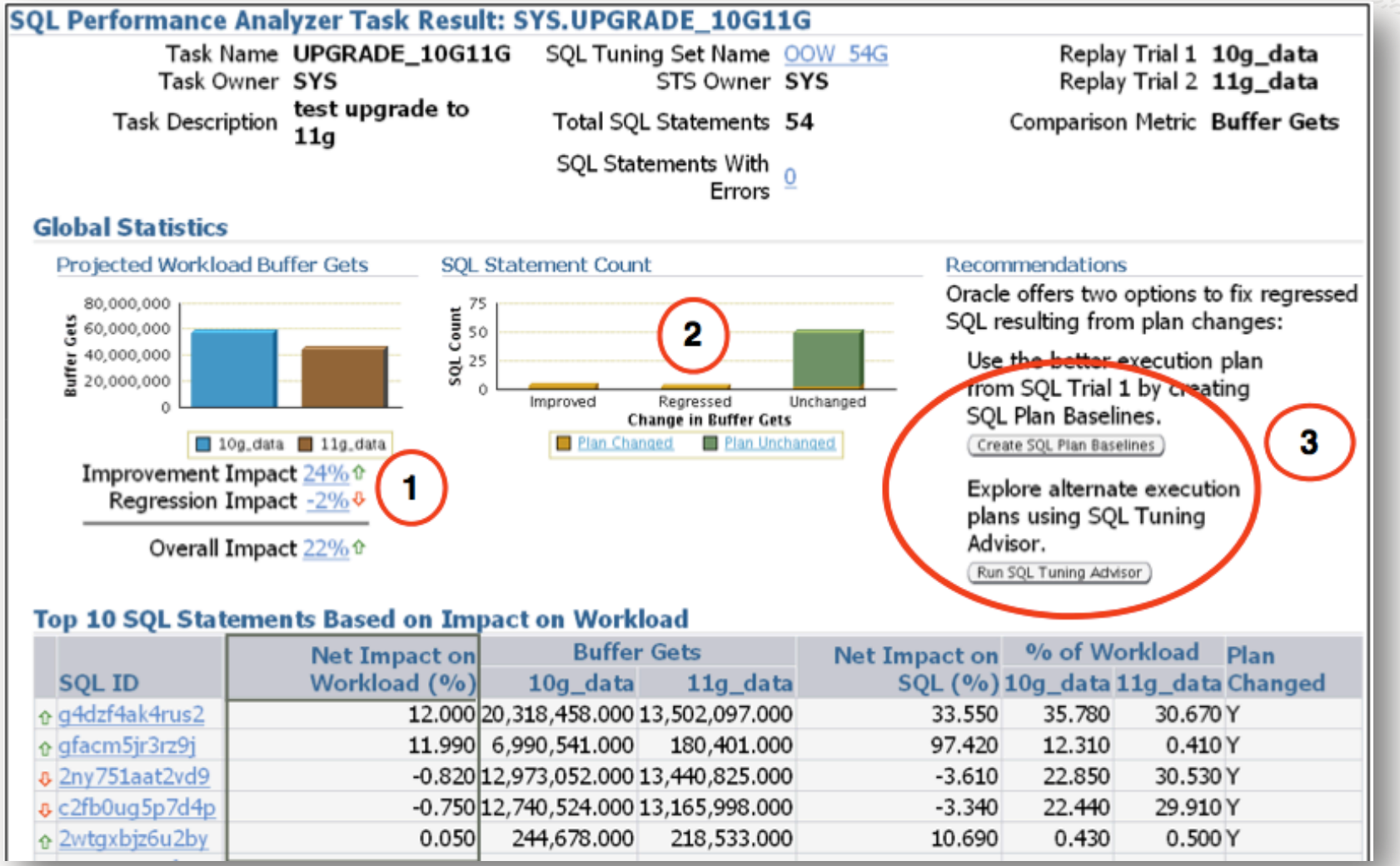
1. What are hints?
2. How to use Optimizer hints
3. Useful Optimizer hints to know
4. Why are Optimizer hints ignored?
5. If you can hint it, baseline it
6. Managing an existing hinted application

Managing an existing hinted application

SQL Performance Analyzer



Managing an existing hinted application



SQL Performance Analyzer Task Result: SYS.UPGRADE_10G11G

Task Name UPGRADE_10G11G
Task Owner SYS
Task Description test upgrade to 11g

SQL Tuning Set Name OOW_54G
STS Owner SYS
Total SQL Statements 54
SQL Statements With Errors 0

Replay Trial 1 10g_data
Replay Trial 2 11g_data
Comparison Metric Buffer Gets

Global Statistics

Projected Workload Buffer Gets

Improvement Impact 24%
Regression Impact -2%
Overall Impact 22%

1

SQL Statement Count

2

Recommendations

Oracle offers two options to fix regressed SQL resulting from plan changes:

Use the better execution plan from SQL Trial 1 by creating SQL Plan Baselines.

Create SQL Plan Baselines

Explore alternate execution plans using SQL Tuning Advisor.

Run SQL Tuning Advisor

3

Top 10 SQL Statements Based on Impact on Workload

SQL ID	Net Impact on Workload (%)	Buffer Gets 10g_data	Buffer Gets 11g_data	Net Impact on SQL (%)	% of Workload 10g_data	% of Workload 11g_data	Plan Changed
g4dzf4ak4rus2	12.000	20,318,458.000	13,502,097.000	33.550	35.780	30.670	Y
gfacm5jr3rz9j	11.990	6,990,541.000	180,401.000	97.420	12.310	0.410	Y
2ny751aat2vd9	-0.820	12,973,052.000	13,440,825.000	-3.610	22.850	30.530	Y
c2fb0ug5p7d4p	-0.750	12,740,524.000	13,165,998.000	-3.340	22.440	29.910	Y
2wtgxbjz6u2by	0.050	244,678.000	218,533.000	10.690	0.430	0.500	Y

Summary

Optimizer hints should only be used with extreme caution

- Exhaust all other tuning mechanisms first
  - Statistics, SQL Tuning Advisor, etc.

To guarantee the same plan every time supply a complete outline

- Easier to do this with SQL Plan Management

Hints live forever!

- Use `_OPTIMIZER_IGNORE_HINTS` parameter to test query performance without hints

SQLMaria®

Join the Conversation

- https://twitter.com/SQLMaria
- https://blogs.oracle.com/optimizer/
- https://sqlmaria.com
- https://www.facebook.com/SQLMaria

Related White Papers

- What to expect from the Optimizer in 12c
- What to expect from the Optimizer in 11g

ORACLE