

ORACLE®

ORACLE®

# Oracle Database Optimizer: Harnessing the Power of Optimizer Hints

Maria Colgan  
Product Manager for the Oracle Optimizer

Hardware and Software  
Engineered to Work Together

ORACLE  
OPEN  
WORLD

# 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

- **What are hints?**
- How to use Optimizer hints
- Useful Optimizer hints to know
- Why are Optimizer hints ignored?
- If you can hint it, baseline it
- Managing an existing hinted application



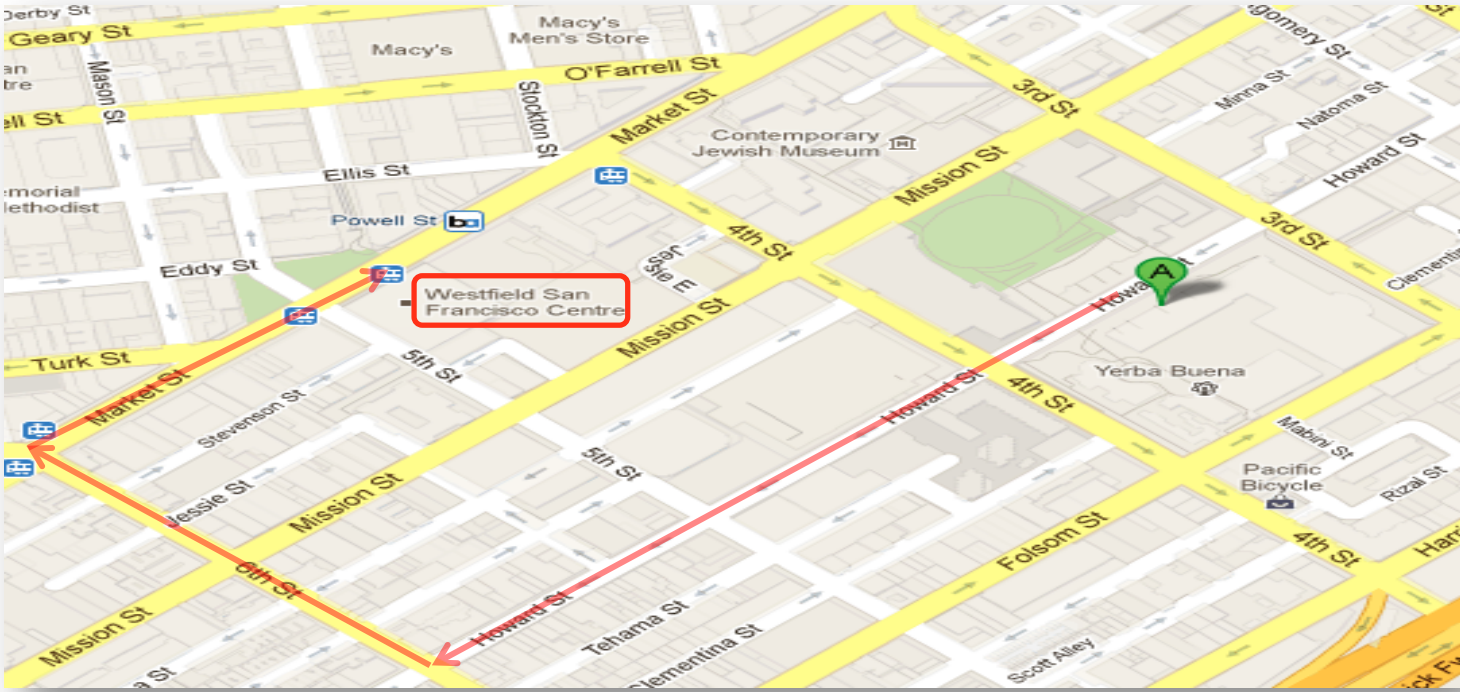
# What are hints?

## 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 the Optimizer mode used to each operation in the execution
- Automatically means the Cost Based Optimizer will be used
  - Only exception is the RULE hint but it must be used alone

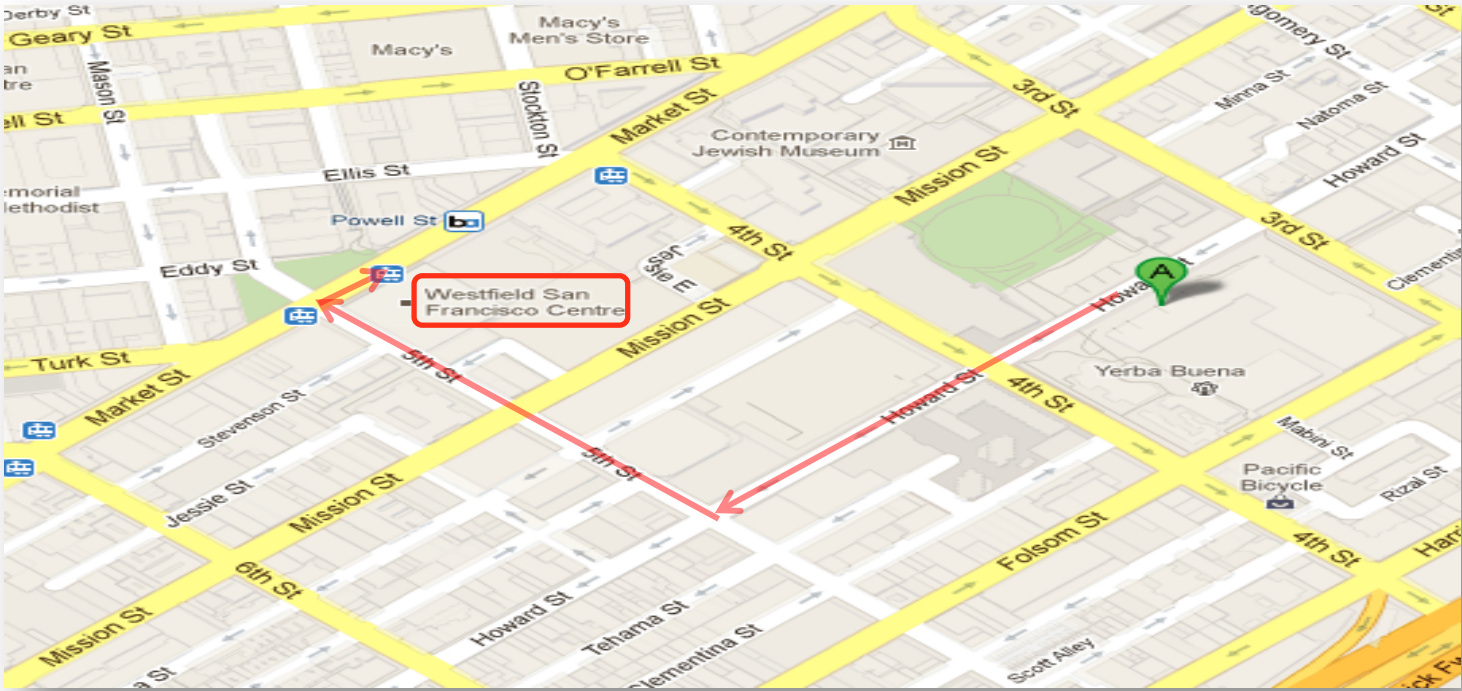
# What are hints?

Example - directions to the mall



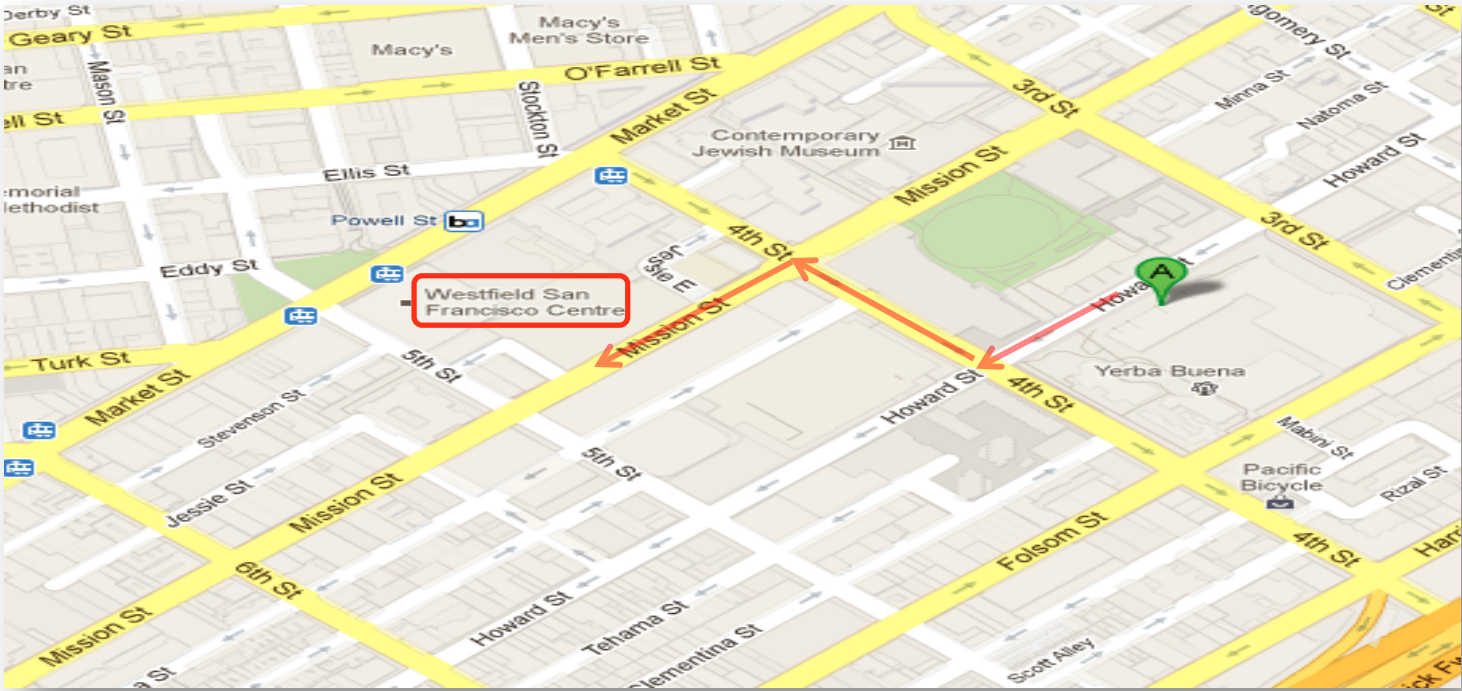
# What are hints?

Example - directions to the mall but don't take 6<sup>th</sup> St



# What are hints?

Example - directions to the mall if you have insider information



# 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 4<sup>th</sup>, 5<sup>th</sup>, or 6<sup>th</sup> street?
  - Best plan would be to go up 4<sup>th</sup> 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 5<sup>th</sup> and Mission garage is irrelevant since I decided to walk

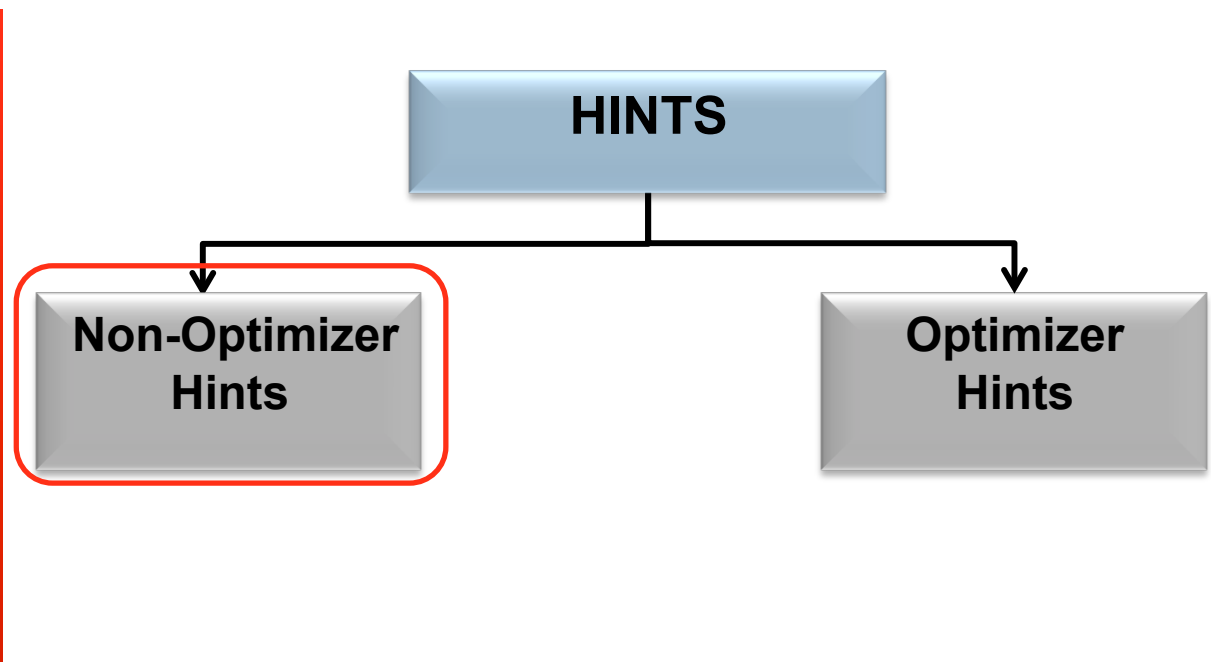
# Two different classes of hints

Hint mechanism is not exclusively used by the Optimizer

Two types of hints

Non-Optimizer hints

Optimizer hints



# Not all hints influence the Optimizer

## Overview

- The hint mechanism is not exclusively used by the Optimizer
- Several other functional area 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



# Checking cardinality estimates

**GATHER\_PLAN\_STATISTICS** hint

```
SQL> SELECT /*+ gather_plan_statistics */ p.prod_name, SUM(s.quantity_sold)
  2 FROM sales s, products p
  3 WHERE s.prod_id = p.prod_id
  4 GROUP By p.prod_name;

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

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time	Buffers
0	SELECT STATEMENT		1		71	00:00:00.34	1636
1	HASH GROUP BY		1	72	71	00:00:00.34	1636
* 2	HASH JOIN		1	72	72	00:00:00.34	1636
3	INDEX FULL SCAN	PROD_SUPP_ID_INDX	1	72	72	00:00:00.01	1
4	VIEW	VW_GBC_5	1	72	72	00:00:00.34	1635
5	HASH GROUP BY		1	72	72	00:00:00.34	1635
6	PARTITION RANGE ALL		1	918k	918k	00:00:00.43	1635
7	TABLE ACCESS STORAGE FULL	SALES	28	918k	918k	00:00:00.24	1635

Compare estimated rows returned for each operation in plan to actual rows returned



# Checking cardinality estimates

MONITOR hint

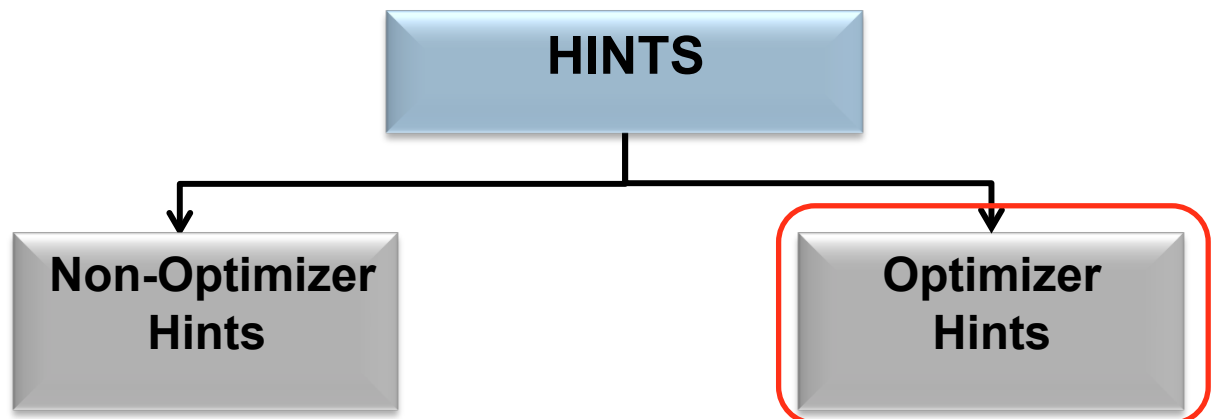
```
SQL> SELECT /*+ monitor */ p.prod_name, SUM(s.quantity_sold)
2 FROM      sales s, products p
3 WHERE     s.prod_id =p.prod_id
4 GROUP By  p.prod_name;
```

Operation	Name	Estimated Rows	Actual Rows	Cost
SELECT STATEMENT			71	
HASH GROUP BY		72	71	438
HASH JOIN		72	72	437
INDEX FULL SCAN	PROD_SUPP_ID_INDX	72	72	1
VIEW	VW_GBC_5	72	72	435
HASH GROUP BY		72	72	435
PARTITION RANGE ALL		919K	919K	318
TABLE ACCESS STORAGE FULL	SALES	919K	919K	318

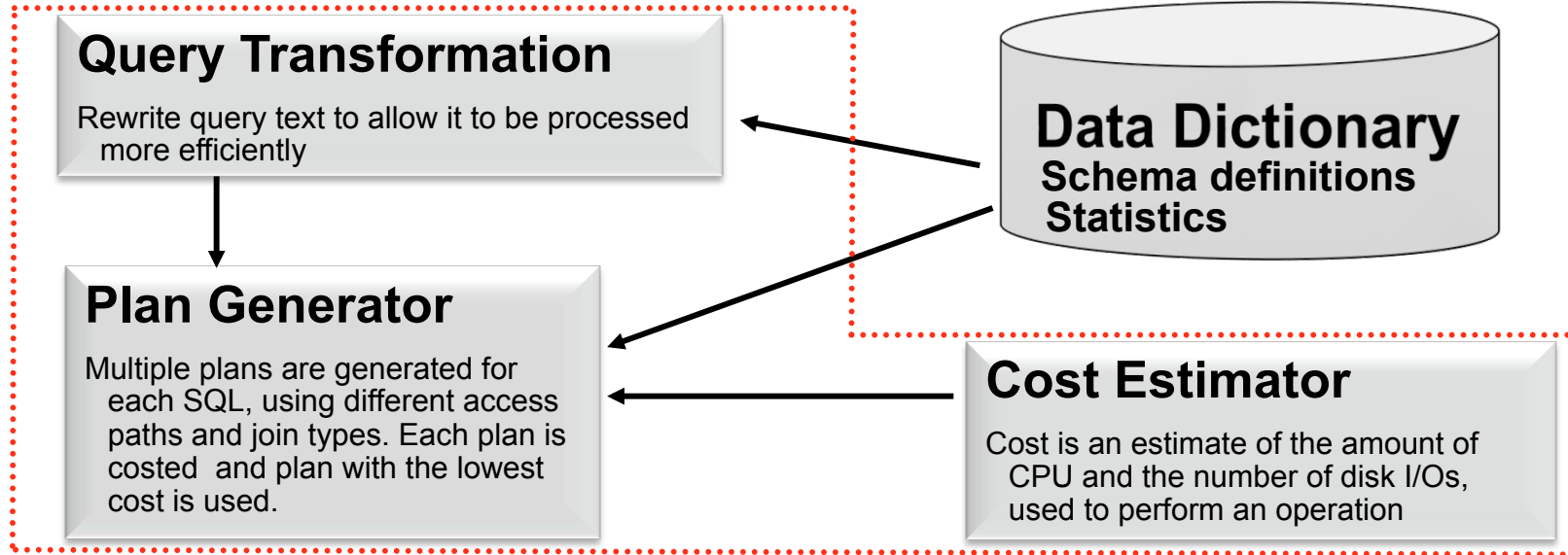
Compare estimated rows returned for each operation in plan to actual rows returned

# Two different classes of hints

This session focuses on  
Optimizer hints



# Inside the Oracle Optimizer



# Hints influencing Query Transformations

## Overview

- First thing the Optimizer does is try to transform (rewrite) your statement
  - This allows additional 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 join methods

- USE\_NL\_WITH\_INDEX
- USE\_HASH

### ▪ Hints to influence access paths

- FULL
- INDEX

### ▪ 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](#)

# 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

- What are hints?
- How to use Optimizer hints
- Useful Optimizer hints to know
- Why are Optimizer hints ignored?
- If you can hint it, baseline it
- 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)

```
SQL> Select /* This is a comment */ count(*) From sales;
```

```
SQL> Select /*+ This is a hint */ count(*) From sales;
```


Hint syntax is correct but it is not a valid hint so it 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



```
SQL> Select /*+ FULL(s) This is a hint and a comment */ count(*) From sales s;  
SQL> Select /*+ This_is_a_comment_and_a_hint FULL(s) */ count(*) From sales s;  
SQL> Select /*+ FULL(S) */ count(*) From sales s /* Comment in seperate block */;
```

# 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 ignored

```
Select /*+ index(e pk_emp) */ * From emp e;
```

# How to use Optimizer hints

Hints only apply to the query block in which they appear

```
SQL> Select /*+ FULL(e) FULL(d) */ e.ename, e.deptno  
2 From emp e  
3 Where e.deptno in (Select d.deptno  
4 From dept d  
5 Where d.loc='CHICAGO');
```

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

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				5 (100)	
* 1	HASH JOIN		5	100	5 (20)	00:00:01
2	TABLE ACCESS BY INDEX ROWID	DEPT	1	11	2 (0)	00:00:01
* 3	INDEX RANGE SCAN	DEPT_LOC_IND	1		1 (0)	00:00:01
4	TABLE ACCESS STORAGE FULL	EMP	14	126	2 (0)	00:00:01

# How to use Optimizer hints

Hints only apply to the query block in which they appear

```
SQL> Select /*+ FULL(e) */ e.ename, e.deptno
2 From emp e
3 Where e.deptno in (Select /*+ FULL(d) */ d.deptno
4 From dept d
5 Where d.loc='CHICAGO');
```

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

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				5 (100)	
* 1	HASH JOIN		5	100	5 (20)	00:00:01
* 2	TABLE ACCESS STORAGE FULL	DEPT	1	11	2 (0)	00:00:01
3	TABLE ACCESS STORAGE FULL	EMP	14	126	2 (0)	00:00:01

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

```
SQL> Select /*+ FULL(e) FULL(@MY_SUBQ d) */ e.ename, e.deptno
2 From emp e
3 Where e.deptno =(Select /*+ QB_NAME(MY_SUBQ) */ d.deptno
4 From dept d
5 Where d.loc='CHICAGO');
```

# 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

## Un-hinted SQL statement

```
SQL> Select      c.cust_first_name, c.cust_last_name, sum(s.amount_sold)
 2 From          customers c, sales s
 3 Where         c.cust_id=s.cust_id
 4 And           c.cust_city='Los Angeles'
 5 And           c.cust_state_province='CA' And   c.country_id=52790
 6 And           s.time_id='09-NOV-00'
 7 Group by     c.cust_first_name, c.cust_last_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)

We want the query to use a nested loops join

# How to use Optimizer hints

Example showing how hints are used

## Hinted SQL statement

```
SQL> explain plan for
 2 Select      /*+ USE_NL(s) */ c.cust_first_name, c.cust_last_name, sum(s.amount_sold)
 3 From        customers c, sales s
 4 Where       c.cust_id=s.cust_id
 5 And         c.cust_city='Los Angeles'
 6 And         c.cust_state_province='CA' And   c.country_id=52790
 7 And         s.time_id='09-NOV-00'
 8 Group by c.cust_first_name, c.cust_last_name;
```

# How to use Optimizer hints

Example showing how hints are used

Even with hint we get hash join plan

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)

Why did it not use the hint?

# How to use Optimizer hints

## Example showing how hints are used

- Lets 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 when join order was customer, sales but not when it was sales, customer

# How to use Optimizer hints

Example showing how hints are used

Hinted SQL statement with both join method and join order hints

```
SSQL> explain plan for
 2 Select      /*+ ORDERED USE NL(s) */ c.cust_first_name, c.cust_last_name, sum(s.amount_sold)
 3 From        customers c, sales s
 4 Where       c.cust_id=s.cust_id
 5 And         c.cust_city='Los Angeles'
 6 And         c.cust_state_province='CA' And   c.country_id=52790
 7 And         s.time_id='09-NOV-00'
 8 Group by c.cust_first_name, c.cust_last_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
```

Cut and paste full outline for the plan

Easier to maintain a full outline using SQL Plan Management

ORACLE

# Program Agenda

- What are hints?
- How to use Optimizer hints
- **Useful Optimizer hints to know**
- Why are Optimizer hints ignored?
- If you can hint it, baseline it
- 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

- 40% of the rows in the employee table have department\_id = 50
- Default plan is a full table scan

```
SQL> Select employee_id, last_name, salary
      2 From employees
      3 Where department_id =50;
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT			
* 1	TABLE ACCESS STORAGE FULL	EMPLOYEES	45	1260

# Changing the Optimizer mode

## FRIST\_ROWS(n) hint example

```
SQL> Select /*+ first_rows(10) */ employee_id, last_name, salary  
2 From employees  
3 Where department_id =50;
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT			
1	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	10	190
* 2	INDEX RANGE SCAN	EMP_DEPARTMEN		

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

# Changing the Optimizer mode

## RULE hint

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

The RULE hint is ignored if

- Other hints are specified in the stmt
- One or more partitioned tables are used
- One or more IOTs are used
- One or more Materialized views exist
- A SAMPLE clauses is specified in a SELECT statement
- 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

```
SQL> select /*+ RULE */ count(*) From sales s;
```

```
COUNT(*)
```

```
918843
```

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

```
PLAN_TABLE_OUTPUT
```

```
SQL_ID 1x3udbbz9ddrn, child number 0
```

```
select /*+ RULE */ count(*) From sales s
```

```
Plan hash value: 1123225294
```

Id	Operation	Name	Rows	Cost (%CPU)
0	SELECT STATEMENT			27 (100)
1	SORT AGGREGATE		1	
2	PARTITION RANGE ALL		918K	27 (0)
3	BITMAP CONVERSION COUNT		918K	27 (0)
4	BITMAP INDEX STORAGE FAST FULL SCAN	SALES_PROMO_BIX		

RULE hint is ignored because SALES is a partitioned table has to use CBO



# Changing the Optimizer mode

## RULE hint

- RULE hint prevents bitmap index being used and triggers full scan

```
SQL> Select /*+ RULE */ count(*) From non_partitioned_sales;
```

Id	Operation
0	SELECT STATEMENT
1	SORT AGGREGATE
2	TABLE ACCESS STORAGE FULL NON_PARTITIONED_SALES

RULE hint is used but prevents bitmap index from being used thus triggers a full table scan

Note

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

# Changing initialization parameter for a query

## OPT\_PARAM hint

- Allows value for 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 related underscore parameters
  - STAR\_TRANSFORMATION\_ENABLED
  - PARALLEL\_DEGREE\_POLICY
  - PARALLEL\_DEGREE\_LIMIT

# 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 4vfmq488y8q0w, 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		1		2 (100)	
1	SORT AGGREGATE		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		1		2 (100)	
1	SORT AGGREGATE		1	12		
* 2	TABLE ACCESS STORAGE FULL	EMPLOYEES	22	264	2 (0)	00:00:01

Extended statistics created & statistics re-gathered as pending statistics  
OPT\_PARAM hint enables pending statistics for only this statement

# Changing Optimizer features enable

This parameter gets its 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	<b>SORT GROUP BY</b>		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

- What are hints?
- How to use Optimizer hints
- Useful Optimizer hints to know
- **Why are Optimizer hints ignored?**
- If you can hint it, baseline it
- 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 emp e;



Select /\*+ index(e emp\_pk) \*/ \* From emp e;



Select /\*+ index(e pk\_emp) \*/ \* From emp e;



# Why are Optimizer hints ignored?

## Invalid hint

- Specifying an index hint on a table with no indexes

```
SQL> explain plan for
2 Select /*+ index(p) */ p.promo_name, p.promo_cost
3 From my_promotions p
4 Where p.promo_category='TV'
5 And p.promo_begin_date='05-OCT-99';
```

Explained.

```
SQL>
```

```
SQL>
```

```
SQL> select * from table(dbms_xplan.display());
```

PLAN\_TABLE\_OUTPUT

Plan hash value: 1905114768

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		1	44
* 1	TABLE ACCESS STORAGE FULL	MY_PROMOTIONS	1	44

Invalid hint because no indexes exist on the table

# Why are Optimizer hints ignored?

## Illegal hint

- Specifying a hash join hint for non-equality join

```
SQL> explain plan for
 2 Select /*+ USE_HASH(e s) */ e.first_name, e.last_name
 3 From employees e, salary_grade s
 4 Where e.salary between s.low_sal and s.high_sal;
```

Explained.

```
SQL>
SQL> select * from table(dbms_xplan.display());
```

PLAN\_TABLE\_OUTPUT

Plan hash value: 1973554736

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

```
SQL> explain plan for
2  Select /*+ index(e empno_pk_ind) parallel(e 8) */ empno, ename
3  From   emp e
4  Where  empno<7700;
```

Explained.

```
SQL> Select * From table(dbms_xplan.display());
```

PLAN\_TABLE\_OUTPUT

-----  
Plan hash value: 468684285

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	

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

# Why are Optimizer hints ignored?

## Contradictory hints

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

```
SQL> explain plan for
 2 select /*+ full(e) index(e deptno fk ind) */ e.empno, e.ename
 3 from emp e
 4 where e.empno<7300;
```

Explained.

```
SQL> select * from table(dbms_xplan.display());
```

PLAN\_TABLE\_OUTPUT

-----  
Plan hash value: 468684285

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		1	11
1	TABLE ACCESS BY INDEX ROWID BATCHED	EMP	1	11
* 2	INDEX RANGE SCAN	EMPNO_PK_IND	1	

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

# 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

```
SQL> SELECT /*+ ORDERED */ e1.last_name, j.job_title, e1.salary, v.avg_salary
  2 FROM     employees e1,
  3         jobs j,
  4         (SELECT e2.department_id, avg(e2.salary) avg_salary
  5         FROM   employees e2, departments d
  6         WHERE  d.location_id=1700
  7         AND    e2.department_id = d.department_id
  8         GROUP BY e2.department_id) v
  9 WHERE     e1.job_id =j.job_id
 10 AND       e1.department_id = v.department_id
 11 AND       e1.salary > v.avg_salary
 12 ORDER BY e1.last_name;
```

# Why are Optimizer hints ignored?

Hint becomes invalid due to transformation

- Actual join order used

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

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

```
SQL> SELECT /*+ NO_MERGE(v) ORDERED */ e1.last_name, j.job_id, v.avg_salary
 2 FROM employees e1,
 3      jobs j,
 4      (SELECT e2.department_id, avg(e2.salary) avg_salary
 5       FROM employees e2, departments d
 6       WHERE d.location_id=1700
 7       AND e2.department_id = d.department_id
 8       GROUP BY e2.department_id) v
 9 WHERE e1.job_id = j.job_id
10 AND e1.department_id = v.department_id
11 AND e1.salary > v.avg_salary
12 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	VIEW		11	176
7	HASH GROUP BY		11	154
8	NESTED LOOPS			
9	3 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

ORACLE

# Program Agenda

- What are hints?
- How to use Optimizer hints
- Useful Optimizer hints to know
- Why are Optimizer hints ignored?
- **If you can hint it, baseline it**
- Managing an existing hinted application

# If you can hint it, baseline it

## Alternative approach to hints

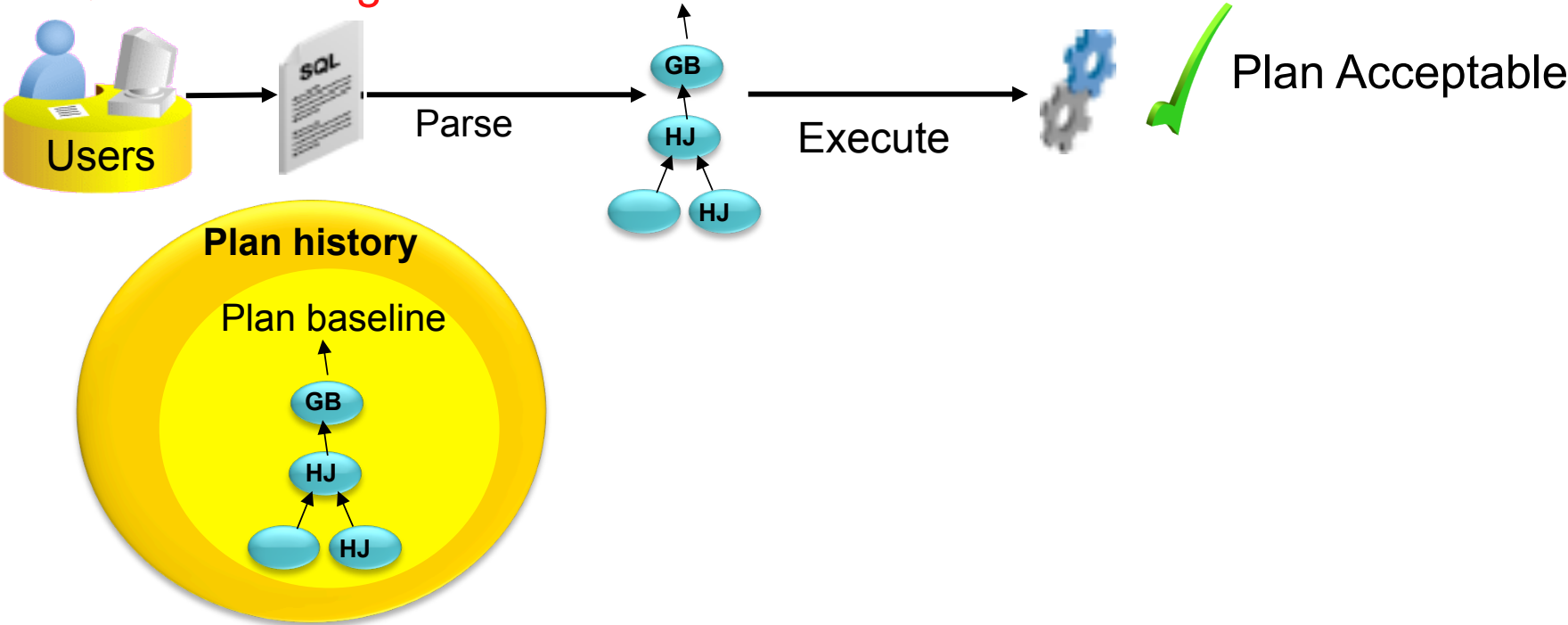
- It is not always possible to add hints to third party applications
- Hints can be extremely difficult to manage over time
- Once added never removed

## Solution

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

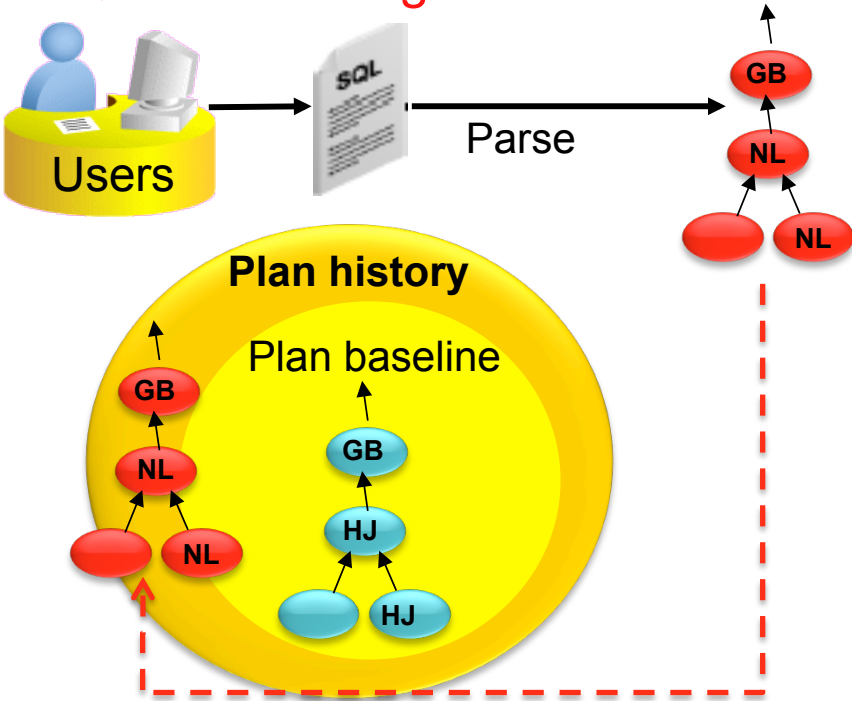
# If you can hint it, baseline it

## SQL Plan Management



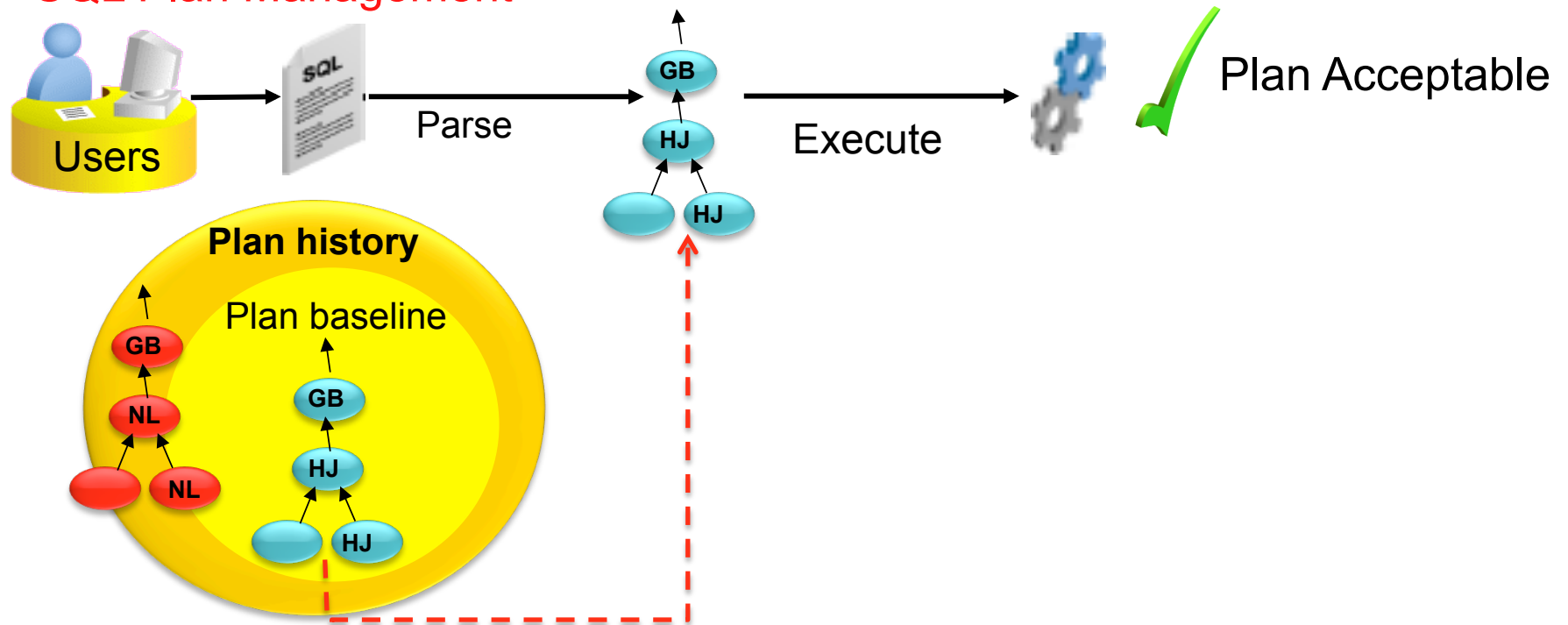
# If you can hint it, baseline it

## SQL Plan Management



# If you can hint it, baseline it

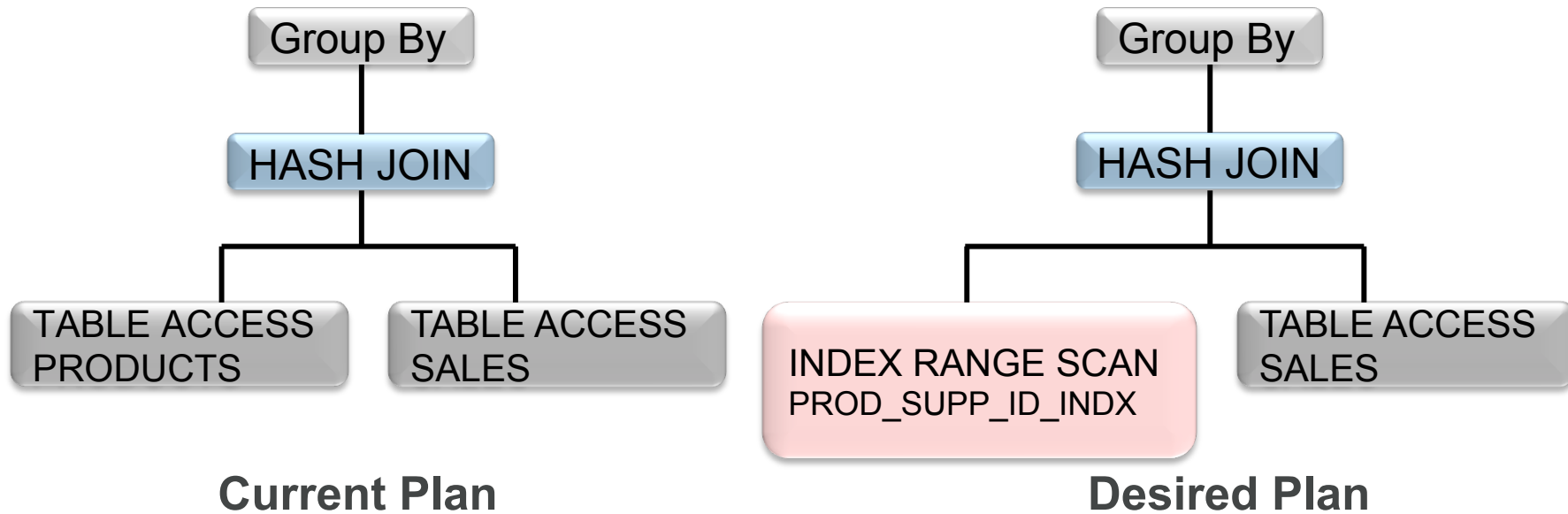
## SQL Plan Management



# Influence execution plan without adding hints

## Example Overview

- Simple two table join between the SALES and PRODUCTS tables



# Influence execution plan without adding hints

## Step 1. Execute the non-hinted SQL statement

```
SQL> VARIABLE sup_id number
SQL> exec :sup_id := 1;

PL/SQL procedure successfully completed.

SQL>
SQL> SELECT      p.prod_name, sum(s.amount_sold) amt
  2 FROM        Sales s, Products p
  3 WHERE       s.prod_id=p.prod_id
  4 AND         p.supplier_id = :sup_id
  5 group by p.prod_name;

PROD_NAME                                AMT
-----
Envoy External 6X CD-ROM                  645586.12
Model SM26273 Black Ink Cartridge         617732.28
Model K8822S Cordless Phone Battery       582640.54
Bounce                                    244595.65
```

ORACLE



# Influence execution plan without adding hints

Default plan uses full table scans followed by a hash join

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT			
1	HASH GROUP BY		72	5688
* 2	HASH JOIN		72	5688
* 3	TABLE ACCESS STORAGE FULL	PRODUCTS	72	3816
4	VIEW	VW_GBC_5	72	1872
5	HASH GROUP BY		72	648
6	PARTITION RANGE ALL		918K	8075K
7	TABLE ACCESS STORAGE FULL	SALES	918K	8075K

# Influence execution plan without adding hints

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

```
SQL> SELECT sql_id, sql_fulltext
 2 FROM v$sql
 3 where sql_text like '%SELECT      p.prod_name%';
```

SQL_ID	SQL_FULLTEXT
bn5p8hp266tah	SELECT      p.prod_name, sum(s.amount_sold) amt FROM        Sales s, Products p WHERE

# Influence execution plan without adding hints

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

```
SQL> variable cnt number;
SQL>
SQL> execute :cnt := dbms_spm.load_plans_from_cursor_cache(sql_id=>'bn5p8hp266tah');

PL/SQL procedure successfully completed.

SQL>
SQL> column sql_text format a45
SQL> 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_10ed3803a09c8fe1	SELECT p.prod_name, sum(s.amount_sold) amt FROM Sales s, Products p WHERE	SQL_PLAN_11v9s0fh9t3z1c47b6be0	YES

# Influence execution plan without adding hints

Step 4. Captured Plan is not our desired plan so it should be disabled

```
SQL> exec :cnt := DBMS_SPM.ALTER_SQL_PLAN_BASELINE(SQL_HANDLE =>'SQL_10ed3803a09c8fe1', -
>                                                PLAN_NAME =>'SQL_PLAN_11v9s0fh9t3z1c47b6be0', -
>                                                ATTRIBUTE_NAME => 'enabled', -
>                                                ATTRIBUTE_VALUE => 'NO');
```

PL/SQL procedure successfully completed.

```
SQL>
SQL> 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_10ed3803a09c8fe1	SELECT p.prod_name, sum(s.amount_sold) amt FROM Sales s, Products p WHERE	SQL_PLAN_11v9s0fh9t3z1c47b6be0	NO

# Influence execution plan without adding hints

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

```
SQL> SELECT /*+ INDEX(p) */ p.prod_name, sum(s.amount_sold) amt
  2 FROM Sales s, Products p
  3 WHERE s.prod_id=p.prod_id
  4 AND p.supplier_id = :sup_id
  5 group by p.prod_name;
```

PROD_NAME	AMT
Bounce	244595.65
Comic Book Heroes	101214.6
Envoy External 6X CD-ROM	645586.12
Finding Fido	78881.08
Model K8822S Cordless Phone Battery	582640.54

# Influence execution plan without adding hints

Step 6. Find SQL\_ID & PLAN\_HASH\_VALUE for hinted SQL stmt in V\$SQL

```
SQL> SELECT sql_id, plan_hash_value, sql_fulltext
2 FROM v$sql
3 where sql_text like '%SELECT /*+ INDEX(p)%';
```

SQL_ID	PLAN_HASH_VALUE	SQL_FULLTEXT
cn29d9b5wp9u7	903671040	SELECT sql_id, plan_hash_value, sql_fulltext FROM v\$sql where sql_text like '
ac7jyxhg9mj0c	187119048	SELECT /*+ INDEX(p) */ p.prod_name, sum(s.amount_sold) amt FROM Sales s, P

# Influence execution plan without adding hints

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

```
SQL> exec :cnt:=dbms_spm.load_plans_from_cursor_cache sql_id =>'ac7jyxhg9mj0c', -  
> plan_hash_value => 187119048, -  
> sql_handle =>'SQL_10ed3803a09c8fe1');  
  
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

```
SQL> SELECT sql_handle, sql_text, plan_name, enabled  
2 FROM dba_sql_plan_baselines  
3 WHERE sql_text like '%SELECT      p.prod_name%';
```

SQL_HANDLE	SQL_TEXT	PLAN_NAME	ENA
SQL_10ed3803a09c8fe1	SELECT      p.prod_name, sum(s.amount_sold) amt FROM Sales s, Products p WHERE	SQL_PLAN_11v9s0fh9t3z1aa1ba510	YES
SQL_10ed3803a09c8fe1	SELECT      p.prod_name, sum(s.amount_sold) amt FROM Sales s, Products p WHERE	SQL_PLAN_11v9s0fh9t3z1c47b6be0	NO

Hinted plan only enabled plan for non-hinted SQL statement

ORACLE



# Influence execution plan without adding hints

## Step 9. Confirm hinted plan is being used

```
SELECT p.prod_name, sum(s.amount_sold) amt FROM Sales s,
Products p WHERE s.prod_id=p.prod_id AND p.supplier_id =
:sup_id group by p.prod_name
Plan hash value: 187119048
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT				573 (100)
1	HASH GROUP BY		71	3550	573 (10)
* 2	HASH JOIN		72	3600	572 (10)
3	VIEW	VW_GBC_5	72	1224	570 (10)
4	HASH GROUP BY		72	648	570 (10)
5	PARTITION RANGE ALL		918K	8075K	530 (3)
6	TABLE ACCESS FULL	SALES	918K	8075K	530 (3)
* 7	INDEX RANGE SCAN	PROD_SUPP_ID_INDX	72	2376	1 (0)

Predicate Information (identified by operation id):

```
2 - access("ITEM_1"="P"."PROD_ID")
7 - access("P"."SUPPLIER_ID"=:SUP_ID)
```

Note

```
- SQL plan baseline SQL_PLAN_11v9s0fh9t3z1aa1ba510 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

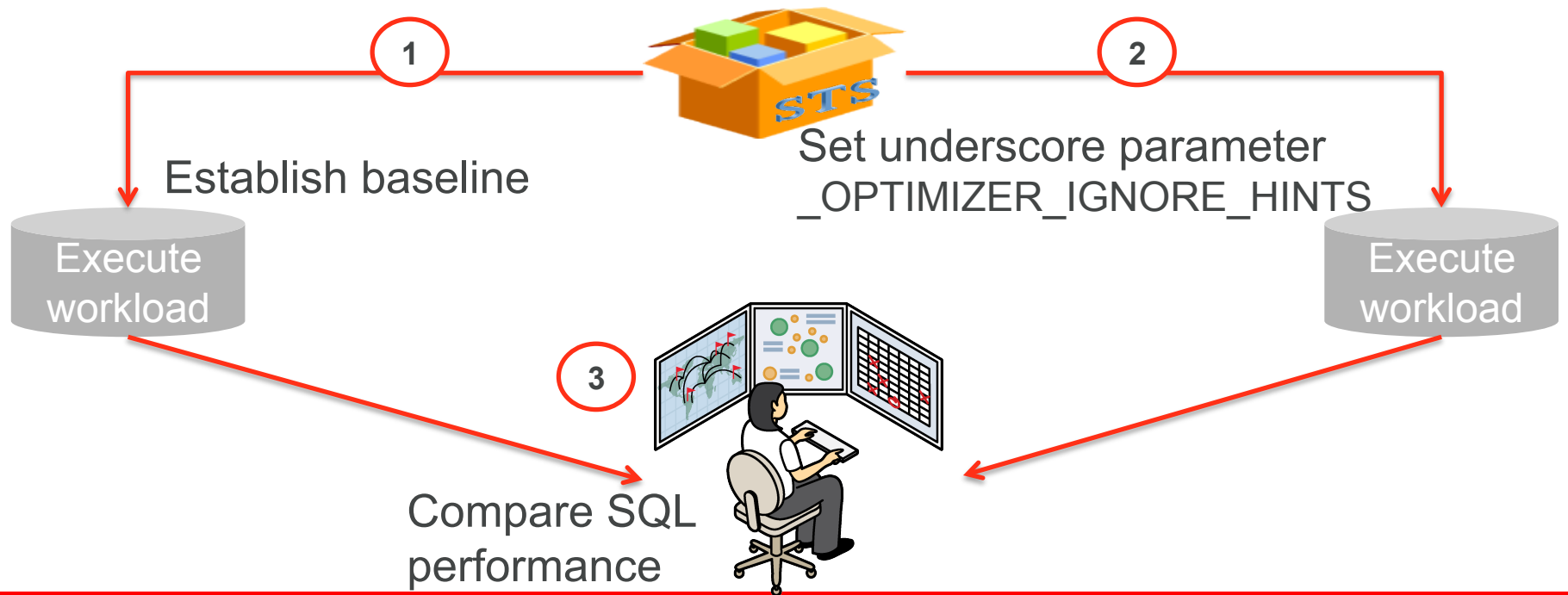
ORACLE

# Program Agenda

- What are hints?
- How to use Optimizer hints
- Useful Optimizer hints to know
- Why are Optimizer hints ignored?
- If you can hint it, baseline it
- **Managing an existing hinted application**

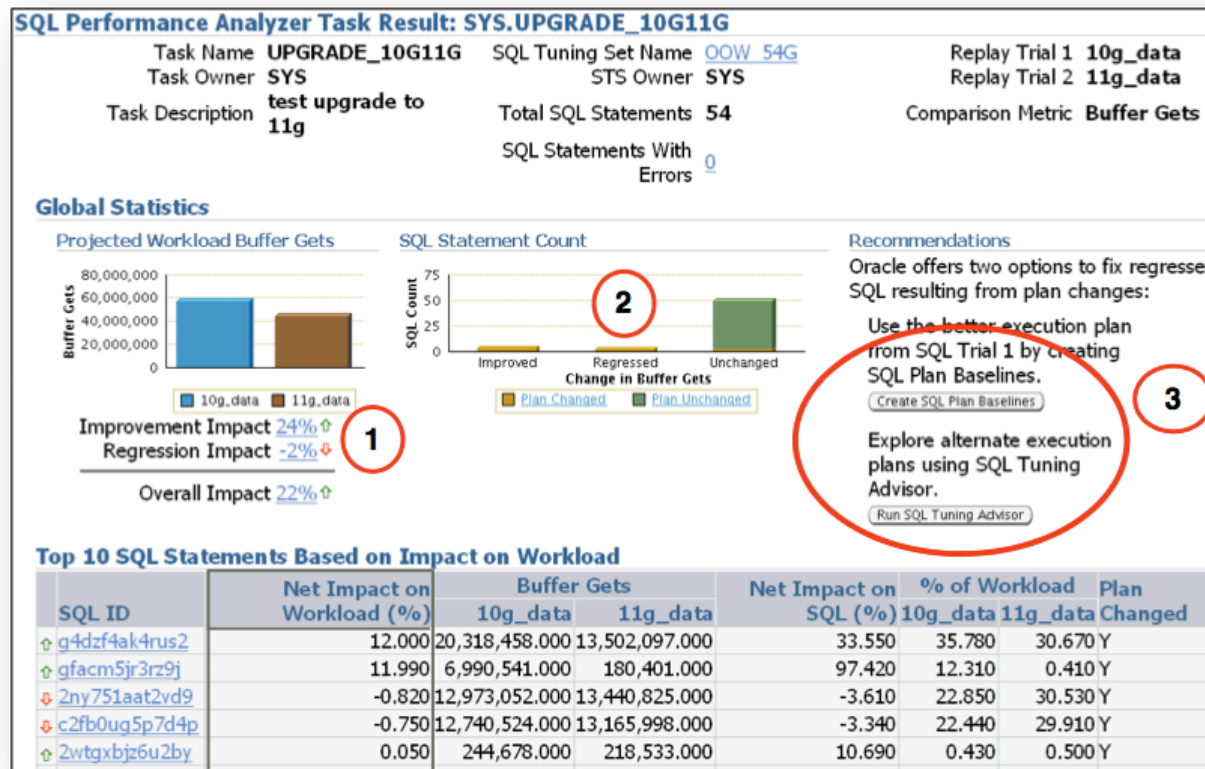
# Managing an existing hinted application

## SQL Performance Analyzer



ORACLE

# Managing an existing hinted application



# 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



**ORACLE  
OPEN  
WORLD**

**Hardware and Software  
Engineered to Work Together**

ORACLE

ORACLE®