ORACLE

SQL Tuning Tips You Can't Do Without

Maria Colgan

Distinguished Product Manager

@SQLMaria

Disclaimer

The goal of this session to provide you tips on how to correct common problems identified from a SQL execution plans

This session will not provide you with sudden enlightenment, making you an Optimizer expert or give you the power to tune SQL statements with the flick of your wrist!

Problem Statement 1

What Join Method Should The Optimizer Pick?

```
SELECT    p.prod_name, SUM(s.quantity_sold)
FROM      sales s, products p
WHERE     s.prod_id = p.prod_id
AND       s.time_id = '03-DEC-20';
GROUP BY p.prod_name;
```

SALES table has 10 million rows

PRODUCTS table has 100,000 rows

Got a Nested Loops Join Instead of Hash Join

```
SELECT    p.prod_name, SUM(s.quantity_sold)
FROM      sales s, products p
WHERE     s.prod_id = p.prod_id
AND       s.time_id = '03-DEC-20';
GROUP BY p.prod_name;
```

SALES table has 10 million rows

PRODUCTS table has 100,000 rows

```
-------------------------------------------------------------------------
| Id  | Operation                          | Name         | Starts | Rows  |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |              |      1 |       |
|   1 |  HASH GROUP BY                     |              |      1 |     1 |
|   2 |   NESTED LOOPS                     |              |      1 |     1 |
|   3 |    NESTED LOOPS                    |              |      1 |     1 |
|   4 |     PARTITION RANGE SINGLE         |              |      1 |     1 |
|*  5 |      TABLE ACCESS STORAGE FULL     | SALES        |      0 |     1 |
|*  6 |     INDEX UNIQUE SCAN              | PRODUCTS_PK  |      0 |     1 |
|   7 |    TABLE ACCESS BY INDEX ROWID     | PRODUCTS     |      0 |     1 |
-------------------------------------------------------------------------
```

Got a Nested Loops Join Instead of Hash Join

```
SELECT    p.prod_name, SUM(s.quantity_sold)
FROM      sales s, products p
WHERE     s.prod_id = p.prod_id
AND       s.time_id = '03-DEC-20';
GROUP BY p.prod_name;
```

1. Always start by looking at the Cardinality Estimate

```
-----------------------------------------------------------------------
| Id  | Operation                          | Name         | Starts | Rows  |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |              |      1 |       |
|   1 |  HASH GROUP BY                     |              |      1 |     1 |
|   2 |   NESTED LOOPS                     |              |      1 |     1 |
|   3 |    NESTED LOOPS                    |              |      1 |     1 |
|   4 |     PARTITION RANGE SINGLE         |              |      1 |     1 |
|*  5 |      TABLE ACCESS STORAGE FULL     | SALES        |      0 |     1 |
|*  6 |     INDEX UNIQUE SCAN              | PRODUCTS_PK  |      0 |     1 |
|   7 |    TABLE ACCESS BY INDEX ROWID     | PRODUCTS     |      0 |     1 |
-----------------------------------------------------------------------
```

Got a Nested Loops Join Instead of Hash Join

```
SELECT    p.prod_name, SUM(s.quantity_sold)
FROM      sales s, products p
WHERE     s.prod_id = p.prod_id
AND       s.time_id = '03-DEC-20';
GROUP BY p.prod_name;
```

1. Always start by looking at the Cardinality Estimate

2. The zero's in the Starts column also look suspicious

```
-----------------------------------------------------------------------
| Id  | Operation                      | Name         | Starts | Rows  |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT               |              |      1 |       |
|   1 |  HASH GROUP BY                 |              |      1 |     1 |
|   2 |   NESTED LOOPS                 |              |      1 |     1 |
|   3 |    NESTED LOOPS                |              |      1 |     1 |
|   4 |     PARTITION RANGE SINGLE     |              |      1 |     1 |
|*  5 |      TABLE ACCESS STORAGE FULL | SALES        |      0 |     1 |
|*  6 |     INDEX UNIQUE SCAN          | PRODUCTS_PK  |      0 |     1 |
|   7 |    TABLE ACCESS BY INDEX ROWID | PRODUCTS     |      0 |     1 |
-----------------------------------------------------------------------
```

Need to Gather More Information

```
SELECT table_name, partitioned
FROM   user_tables
WHERE  table_name in
       ('SALES','PRODUCTS');
```

TABLE_NAME	PARTITIONED
PRODUCTS	NO
SALES	YES

Need to Gather More Information

```
SELECT table_name, partitioned
FROM   user_tables
WHERE  table_name in
       ('SALES','PRODUCTS');
```

TABLE_NAME	PARTITIONED
PRODUCTS	NO
SALES	YES

```
SELECT table_name, stale_stats
FROM   user_tab_statistics
WHERE table_name in ('SALES','PRODUCTS');
```

TABLE_NAME	STALE_STATS
PRODUCTS	NO
SALES	NO
SALES	
SALES	
SALES	
SALES	
SALES	

Need to Gather More Information

```
SELECT table_name, partitioned
FROM   user_tables
WHERE  table_name in
       ('SALES','PRODUCTS');
```

TABLE_NAME	PARTITIONED
PRODUCTS	NO
SALES	YES

```
SELECT table_name, stale_stats
FROM   user_tab_statistics
WHERE table_name in ('SALES','PRODUCTS');
```

TABLE_NAME	STALE_STATS
PRODUCTS	NO
SALES	NO
SALES	
SALES	
SALES	
SALES	
SALES	

Table Stats

Partition Level Stats

Need to Gather More Information

Next Look at the Where Clause Predicates

```
AND       s.time_id = '03-DEC-20';


SELECT min(time_id),
       max(time_id)
FROM   sales;
```

MIN(DATE_ID)	MAX(DATE_ID)
01-JAN-04	03-DEC-20

Need to Gather More Information

Next Look at the Where Clause Predicates

```
AND       s.time_id = '03-DEC-20';
```

```
SELECT min(time_id),
       max(time_id)
FROM   sales;
```

MIN(DATE_ID)	MAX(DATE_ID)
01-JAN-04	03-DEC-20

```
DECLARE
  rv RAW(32) ;
  dt DATE;
BEGIN
  SELECT high_value INTO rv
  FROM   user_tab_col_statistics
  WHERE  table_name='SALES'
  AND    column_name='TIME_ID';
  dbms_stats.convert_raw_value(rv, dt);
  dbms_output.put_line( TO_CHAR(dt,'dd-MON-yy'));
END;
/

31-OCT-2020

PL/SQL procedure successfully completed.
```

Stale Statistics Cause an "Out of Range" Issues

- Statistics for `TIME_ID` column show max value as **'31-OCT-2020'**
- Our query is looking for sales on **'03-DEC-2020'**
- Optimizer checks if predicate falls between the min, max value of column

Stale Statistics Cause an "Out of Range" Issues

- Statistics for `TIME_ID` column show max value as **'31-OCT-2020'**
- Our query is looking for sales on **'03-DEC-2020'**
- Optimizer checks if predicate falls between the min, max value of column
- "Out of Range" means value supplied is outside the min, max range
- Optimizer prorates cardinality based on the distance between the predicate value and the maximum value

Stale Statistics Cause an "Out of Range" Issues

- Statistics for `TIME_ID` column show max value as **'31-OCT-2020'**
- Our query is looking for sales on **'03-DEC-2020'**
- Optimizer checks if predicate falls between the min, max value of column
- "Out of Range" means value supplied is outside the min, max range
- Optimizer prorates cardinality based on the distance between the predicate value and the maximum value

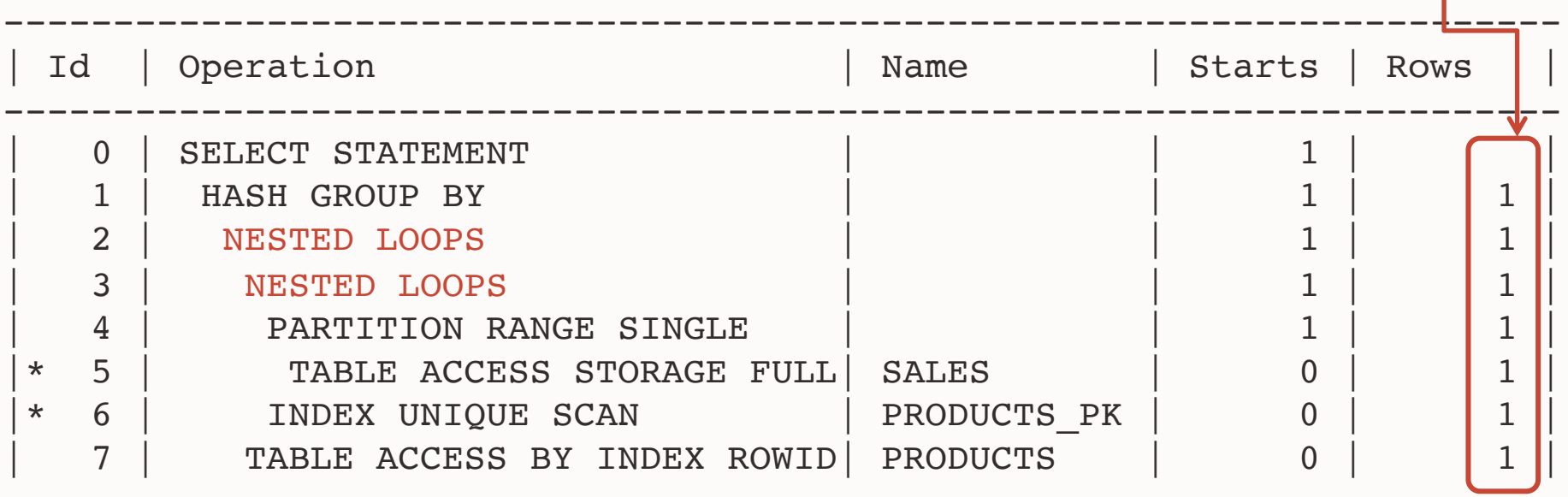
In **10053 trace file** you will find:

"Using prorated density: 0.000000 of col #1 as selectvity of out-of-range/non-existent value pred"

Stale Statistics Cause an "Out of Range" Issues

```
SELECT    p.prod_name, SUM(s.quantity_sold)
FROM      sales s, products p
WHERE     s.prod_id = p.prod_id
AND       s.time_id = '03-DEC-20';
GROUP BY p.prod_name;
```

Optimizer assumes there are no rows for time_id='03-DEC_20' because it is out side of the [MIN,MAX] range in the statistics



Id	Operation	Name	Starts	Rows
0	SELECT STATEMENT		1	
1	HASH GROUP BY		1	1
2	NESTED LOOPS		1	1
3	NESTED LOOPS		1	1
4	PARTITION RANGE SINGLE		1	1
* 5	TABLE ACCESS STORAGE FULL	SALES	0	1
* 6	INDEX UNIQUE SCAN	PRODUCTS_PK	0	1
7	TABLE ACCESS BY INDEX ROWID	PRODUCTS	0	1

Always be wary of a cardinality estimates of 1 !

Fix Out-of-Range Cardinality Misestimate

- By default stats are only considered stale after 10% of the rows have changed
- Our tables SALES has 10 million rows in it
  - 1 million rows need to be insert before the statistics are considered stale

Fix Out-of-Range Cardinality Misestimate

Option 1 - USE DBMS_STATS.SET_TABLE_PREFS To Change Staleness Threshold

- By default stats are only considered stale after 10% of the rows have changed
- Our tables SALES has 10 million rows in it
  - 1 million rows need to be insert before the statistics are considered stale

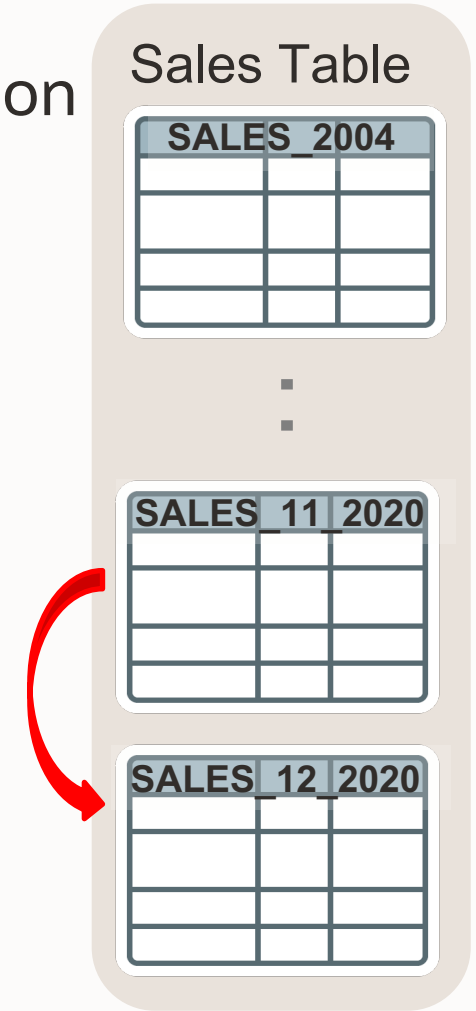
- To avoid out-of-range problems lower the staleness threshold for larger tables

```
BEGIN
    dbms_stats.set_table_prefs('SH', 'SALES', 'STALE_PERCENT', '1');
    dbms_stats.gather_table_stats('SH', 'SALES');
  END;
/
```

Fix Out-of-Range Cardinality Misestimate

Option 2 - USE DBMS_STATS.COPY_TABLE_STATS()

- Copies stats from source partition to destination partition
- Adjusts min & max values for partition column at both partition & global level
- Copies statistics of the dependent objects
  - Columns, local indexes etc.
  - Does not update global indexes




```
dbms_stats.copy_table_stats
          ('SH',
          'SALES',
          'SALES_11_2020',
          'SALES_12_2020');
```

Fixing Out-of-Range Error Automatic Hash Join Plan

```
SELECT    p.prod_name, SUM(s.quantity_sold)
FROM      sales3 s, products p
WHERE     s.prod_id = p.prod_id
AND       s.time_id = '03-DEC-20';
GROUP BY p.prod_name;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                          | Name     | Rows  | Bytes | Cost (%CPU)| Pstart| Pstop |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |          |     1 |    32 |    81  (20)|       |       |
|   1 |  HASH GROUP BY                     |          |     1 |    32 |    81  (20)|       |       |
|*  2 |   HASH JOIN                        |          |  3671 |   114K|    80  (19)|       |       |
|   3 |    JOIN FILTER CREATE              | :BF0000  |  3671 |   114K|    80  (19)|       |       |
|   4 |     PARTITION RANGE SINGLE         |          |  3596 | 61132 |    17  (12)|   KEY |   KEY |
|*  5 |      TABLE ACCESS STORAGE FULL     | SALES    |  3596 | 61132 |    17  (12)|   KEY |   KEY |
|   6 |    JOIN FILTER USE                 | :BF0000  |  1398K|    20M|    58  (14)|       |       |
|*  7 |     TABLE ACCESS STORAGE FULL      | PRODUCTS |  1398K|    20M|    58  (14)|       |       |
-------------------------------------------------------------------------------------------
```

Problem Statement 2

What Index Do You Think The Optimizer Should Pick?

```
SELECT comment
FROM   my_sales
WHERE  prod_id = 141
AND    cust_id < 8938;
```

MY_SALES table has 2 million rows

Two Index:

PROD_CUST_COM_IND (prod_id, comment, cust_id)

PROD_CUST (prod_id, cust_id)

Wrong Index is Chosen

```
SELECT comment
FROM   my_sales
WHERE  prod_id = 141
AND    cust_id < 8938;
```

MY_SALES table has 2 million rows

Two Index:
PROD_CUST_COM_IND (prod_id, comment, cust_id)

PROD_CUST (prod_id, cust_id)

```
-------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | COST (%CPU) |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |  2579 |  2580   (1) |
|   1 |  TABLE ACCESS BY INDEX ROWID | MY_SALES      |  2579 |  2580   (1) |
|   2 |   INDEX RANGE SCAN           | PROD_CUST_IND |  2579 |     9   (1) |
-------------------------------------------------------------------------
```

WHY DID IT NOT PICK THE INDEX WITH ALL THE COLUMNS NEEDED?

Need to Gather More Information

Confirm our Preferred Plan is Possible

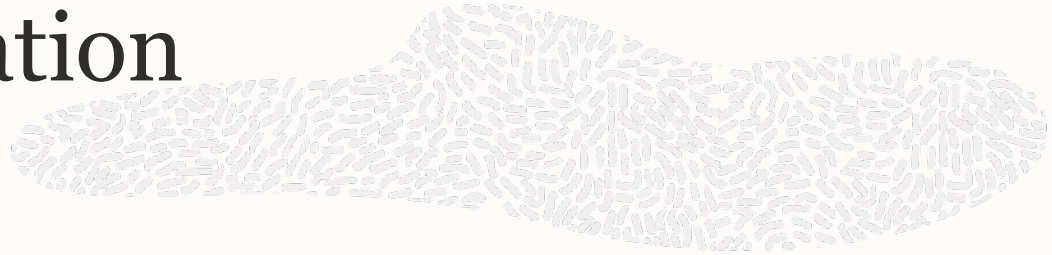
```
SELECT /* INDEX(PROD_CUST_COM_IND) */ comment
FROM   my_sales
WHERE  prod_id = 141
AND    cust_id < 8938;
```

```
-------------------------------------------------------------------------------
| Id  | Operation                    | Name               | Rows  | COST (%CPU) |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |  2579 |  9728   (1) |
|   1 |  INDEX RANGE SCAN            | PROD_CUST_COM_IND  |  2579 |  9728   (1) |
-------------------------------------------------------------------------------
```

WHY IS THE COST SO HIGH?

Need to Gather More Information

Check Table Stats



```
SELECT table_name, partitioned
FROM   user_tables
WHERE  table_name = 'MY_SALES';
```

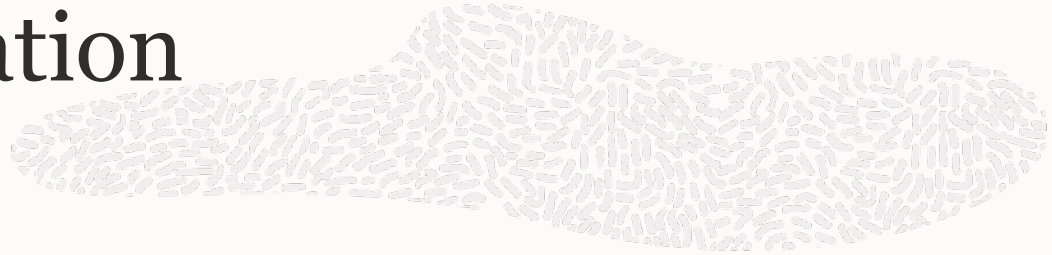
TABLE_NAME	PARTITIONED
MY_SALES	NO

```
SELECT table_name, stale_stats
FROM   user_tab_statistics
WHERE  table_name = 'MY_SALES';
```

TABLE_NAME	STALE_STATS
MY_SALES	NO

Need to Gather More Information

Check Index Stats

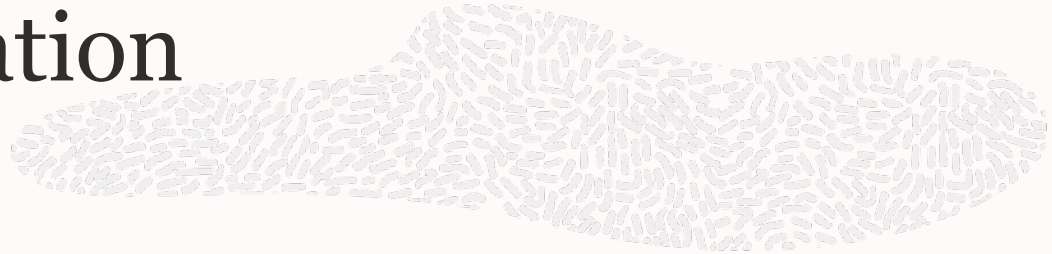


```
SELECT index_name, stale_stats
FROM   user_ind_statistics
WHERE table_name = 'MY_SALES';
```

INDEX_NAME	STALE_STATS
PROD_CUST_COM_IND	NO
PROD_CUST	NO

Need to Gather More Information

Check Index Stats



```
SELECT index_name, stale_stats
FROM   user_ind_statistics
WHERE table_name = 'MY_SALES';
```

INDEX_NAME	STALE_STATS
PROD_CUST_COM_IND	NO
PROD_CUST	NO

```
SELECT index_name, leaf_blocks, blevel
FROM   user_indexes
WHERE  table_name = 'MY_SALES';
```

INDEX_NAME	LEAF_BLOCKS	BLEVEL
PROD_CUST_COM_IND	699467	9
PROD_CUST	5468	2

Need to Gather More Information

Look At Predicate Information Under the Plan

```
SELECT /* INDEX(PROD_CUST_COM_IND) */ comment
FROM my_sales
WHERE prod_id = 141
AND   cust_id < 8938;
```

```
-------------------------------------------------------------------------------
| Id  | Operation                    | Name               | Rows  | COST (%CPU) |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |  2579 |  9728   (1) |
|   1 |  INDEX RANGE SCAN            | PROD_CUST_COM_IND  |  2579 |  9728   (1) |
-------------------------------------------------------------------------------
```

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

   1 – access("PROD_ID"=141 AND "CUST_ID"<8938)
       filter("CUST_ID"<8938)
```

Remember the order of the column in the PROD_CUST_COM_IND index (prod_id, comment, cust_id)

Additional Information Under The Execution Plan

```
SELECT /*+ gather_plan_statistics */ count(*)  FROM    sales2 WHERE
prod_id=to_number('139')

Plan hash value: 1631620387

-------------------------------------------------------------------------------------
| Id  | Operation         | Name        | Starts | E-Rows | Cost (%CPU)| A-Rows |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |             |      1 |        |    35 (100)|      1 |
|   1 |  SORT AGGREGATE   |             |      1 |      1 |            |      1 |
|* 2  |   INDEX RANGE SCAN| MY_PROD_IND |      1 |  12762 |    35   (0)|  11574 |
-------------------------------------------------------------------------------------

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

   2 - access("PROD_ID"=139)
```

Access predicate

- Where clause predicate used for data retrieval
  - The start and stop keys for an index
  - If rowids are passed to a table scan

Additional Information Under The Execution Plan

```
SQL> SELECT username
  2  FROM   my_users
  3  WHERE  username LIKE 'MAR%';

USERNAME
------------------------------------------------------------------------------
MARIA

Plan hash value: 2982854235

-----------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |       |       |     2 (100)|          |
|*  1 |  TABLE ACCESS FULL| MY_USERS |     1 |    66 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------

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

   1 - filter("USERNAME" LIKE 'MAR%')
```

Filter predicate

- Where clause predicate that is not used for data retrieval but to eliminate uninteresting row once the data is found
- Requires additional CPU resources to apply filters

Costing of Each Index Access is Different

```
-------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | COST (%CPU) |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |  2579 |  2580   (1) |
|   1 |  TABLE ACCESS BY INDEX ROWID | MY_SALES     |  2579 |  2580   (1) |
|   2 |   INDEX RANGE SCAN           | PROD_CUST_IND|  2579 |     9   (1) |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("PRODUCT_ID"=517538 AND "CUST_ID"<8938)
```

Both columns are used as **Access Predicates**

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name               | Rows  | COST (%CPU) |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                    |  2579 |  9728   (1) |
|   1 |  INDEX RANGE SCAN   | PROD_CUST_COM_IND  |  2579 |  9728   (1) |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("PRODUCT_ID"=517538 AND "CUST_ID"<8938)
     - filter("CUST_ID"<8938)
```

Only the PROD_ID is used as **Access Predicates**

Costing of Each Index Access is Different

```
-------------------------------------------------------------------------------
| Id  | Operation                     | Name         | Rows  | COST (%CPU) |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |  2579 |  2580   (1) |
|   1 |  TABLE ACCESS BY INDEX ROWID  | MY_SALES     |  2579 |  2580   (1) |
|   2 |   INDEX RANGE SCAN            | PROD_CUST_IND|  2579 |     9   (1) |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("PRODUCT_ID"=517538 AND "CUST_ID"<8938)
```

Both columns are used as **Access Predicates**

Cost calculation for this index:

blevel + (cardinality X leaf blocks)
#rows

2 + (2579 X 5468) = 9
2098400

```
-------------------------------------------------------------------------------
| Id  | Operation            | Name                | Rows  | COST (%CPU) |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                     |  2579 |  9728   (1) |
|   1 |  INDEX RANGE SCAN    | PROD_CUST_COM_IND   |  2579 |  9728   (1) |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("PRODUCT_ID"=517538 AND "CUST_ID"<8938)
     - filter("CUST_ID"<8938)
```

Only the PROD_ID is used as **Access Predicates**

Cost calculation for this index:

blevel + (1 X Leaf Block)
NDV of col1

9 + (0.013 X 699467) = 9724

The CPU cost of applying the filter makes up the additional cost of 4

Common Misconception

Common misconception Optimizer will pick the index with all the necessary columns in it

Common Misconception

Common misconception Optimizer will pick the index with all the necessary columns in it

Optimizer picks the index based on the cost of the access

Common Misconception

Common misconception Optimizer will pick the index with all the necessary columns in it

Optimizer picks the index based on the cost of the access

How the WHERE clause predicates are used and the order of the columns in the index have a massive impact on the Optimizers choice

Solution: Right Index is Chosen

```
SELECT comment
FROM my_sales
WHERE prod_id = 141
AND   cust_id < 8938;
```

MY_SALES table has 2 million rows

Three Index:

PROD_CUST_COM_IND (prod_id, comment, cust_id)

PROD_CUST (prod_id, cust_id)

PROD_CUST_COM_IND2(prod_id, cust_id, comment)

```
-------------------------------------------------------------------------
| Id  | Operation           | Name                 | Rows  | COST (%CPU) |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                      |  2579 |  1935 (100) |
|   1 |  INDEX RANGE SCAN   | PROD_CUST_COM_IND2   |  2579 |  1935   (0) |
-------------------------------------------------------------------------
```

Problem Statement 3

What Join Method Should the Optimizer Pick?

```
SELECT    COUNT(*)
FROM      sales s, customers c
WHERE     s.cust_id = c.cust_id
AND       substr(c.cust_state_province,1,2)='CA';
```

SALES table has 10 million rows

CUSTOMERS table has 10,000 rows

Sales table has index on CUST_ID

Got a Nested Loops Join Instead of Hash Join

```
SELECT    COUNT(*)
FROM      sales s, customers c
WHERE     s.cust_id = c.cust_id
AND       substr(c.cust_state_province,1,2)='CA';
```

SALES table has 10 million rows

CUSTOMERS table has 10,000 rows

Sales table has index on CUST_ID

```
-------------------------------------------------------------------------
| Id  | Operation                     | Name           | Rows  | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                |       |    26 (100)|
|   1 |  SORT AGGREGATE               |                |     1 |            |
|   2 |   NESTED LOOPS                |                |    41 |    26   (0)|
|   3 |    TABLE ACCESS FULL          | CUSTOMER       |   100 |     2   (0)|
|*  4 |    BITMAP CONVERSION COUNT    |                |     4 |    26   (0)|
|*  5 |     BITMAP INDEX SINGLE VALUE | SALES_CUST_BIX |       |            |
-------------------------------------------------------------------------
```

Need to Gather More Information

Confirm our Preferred Plan is Possible

```
SELECT /*+ USE_HASH(s) */ COUNT(*)
FROM      sales s, customers c
WHERE     s.cust_id = c.cust_id
AND       substr(c.cust_state_province,1,2)='CA';
```

```
-------------------------------------------------------------------------
| Id  | Operation                    | Name     | Rows      | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          |           |  1114 (100)|
|   1 |  SORT AGGREGATE              |          |         1 |            |
|   2 |   HASH JOIN                  |          |     14732 |  1114   (2)|
|*  4 |    TABLE ACCESS FULL         | CUSTOMER |       100 |   406   (1)|
|*  5 |    TABLE ACCESS FULL         | SALES    |       10M |   703   (2)|
-------------------------------------------------------------------------
```

WHY IS THE COST SO HIGH?

Need to Gather More Information

Compare Estimated Cardinality to Actual Cardinality

```
SELECT    /*+ GATHER_PLAN_STATISTICS */
          COUNT(*)
FROM      sales s, customers c
WHERE     s.cust_id = c.cust_id
AND       substr(c.cust_state_province,1,2)='CA';

SELECT * FROM table(dbms_xplan.display_cursor(FORMAT=>'ALLSTATS LAST'));
```

Id	Operation	Name	E-Rows	A-Rows	Buffer
0	SELECT STATEMENT		1	1	15984
1	SORT AGGREGATE		1	1	15984
2	NESTED LOOPS		1471	154K	15984
3	TABLE ACCESS FULL	CUSTOMER	100	6656	15984
* 4	BITMAP CONVERSION COUNT		4	154K	2914
* 5	INDEX RANGE SCAN	SALES_CUST_BIX			13034

Why Is The Cardinality Estimates So Wrong?

Start With Cardinality Estimate of the Table on the Left-Hand Side of Join

- `Customers` is the table on the left hand side of the join & predicates is

```
substr(c.cust_state_province,1,2) = 'CA'
```


Why Is The Cardinality Estimates So Wrong?

Start With Cardinality Estimate of the Table on the Left-Hand Side of Join

- `Customers` is the table on the left hand side of the join & predicates is

```
substr(c.cust_state_province,1,2) = 'CA'
```

Function wrapped column

- Optimizer doesn't know how function affects values in the column

Why Is The Cardinality Estimates So Wrong?

Function Wrapped Column Blinds the Optimizer

```
SELECT COUNT(*)
FROM   customers
WHERE  substr(c.cust_state_province,1,2) = 'CA';
```

```
-------------------------------------------------------------------------
| Id  | Operation                    | Name     | Rows      | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          |           |     2  (100)|
|   1 |  SORT AGGREGATE              |          |         1 |            |
|*  4 |    TABLE ACCESS FULL         | CUSTOMER |       100 |     2    (0)|
-------------------------------------------------------------------------
```

Remember CUSTOMERS table has 10,000 rows

- Optimizer doesn't know how function affects values in the column
- Optimizer guesses the cardinality to be 1% of rows

Tell the Optimizer How Function Affects Column Values

Create Extended Statistics

```
SELECT
dbms_stats.create_extended_stats(null,'CUSTOMERS',
                        '(SUBSTR(CUST_STATE_PROVINCE,1,2))')
FROM dual;
```

```
DBMS_STATS.CREATE_EXTENDED_STATS(NULL,'CUSTOMERS','(SUBSTR(CUST_STATE_PROVINCE,1,2))')
```

```
SYS_STUAJYEDA$07W8CRW1A18K4Q_G
```

Tell the Optimizer How Function Affects Column Values

Identifying Extended Statistics

```
SELECT column_name, num_distinct, histogram
FROM   user_tab_col_statistics
WHERE  table_name='CUSTOMERS';
```

COLUMN_NAME	NUM_DISTINCT	HISTOGRAM
SYS_STUAJYEDA$07W8CRW1A18K4Q_G	84	FREQUENCY
C_CUST_ID	335	HYBRID
CUST_YEAR_OF_BIRTH	69	FREQUENCY
CUST_VALID	2	FREQUENCY
CUST_TOTAL_ID	1	FREQUENCY
CUST_TOTAL	1	FREQUENCY
CUST_STREET_ADDRESS	340	HYBRID

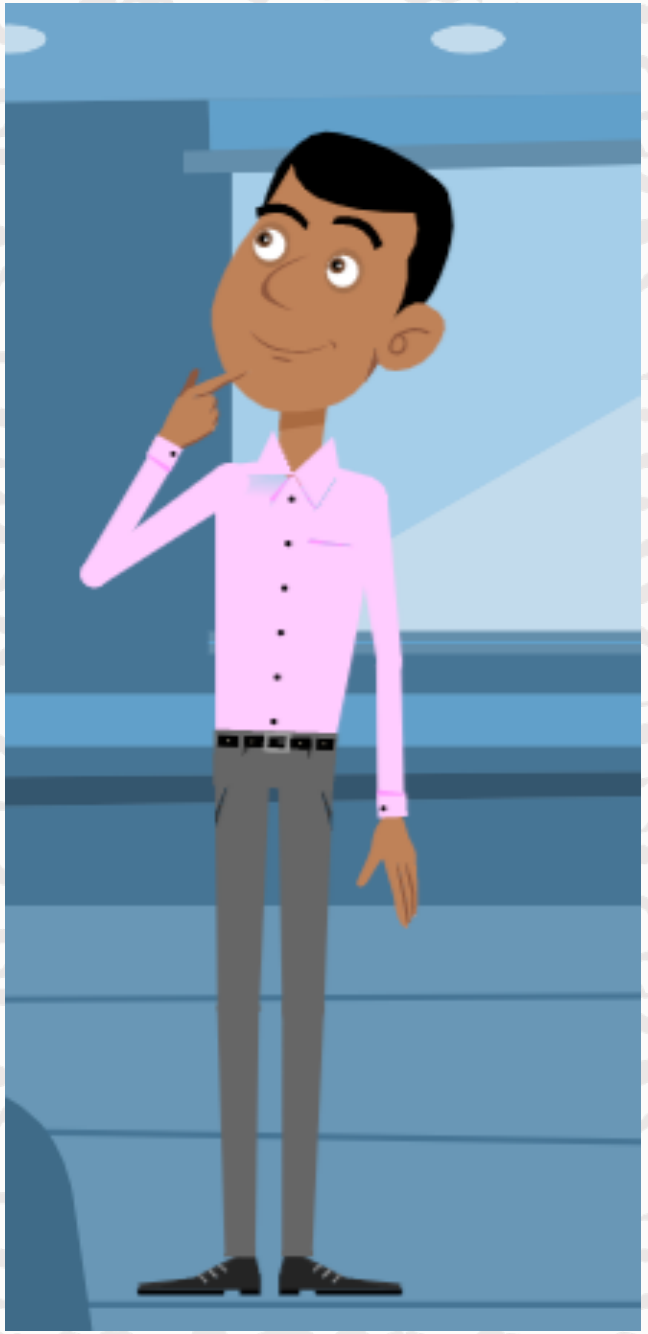
:

Fixing Cardinality Estimate on Left-Hand Side Automatic Produces a Hash Join Plan

```
SELECT COUNT(*)
FROM   sales s, customers c
WHERE  s.cust_id = c.cust_id
AND    substr(c.cust_state_province,1,2)='CA';
```

```
-------------------------------------------------------------------------
| Id  | Operation                    | Name     | Rows      | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |          |           |  1114 (100)|
|   1 |  SORT AGGREGATE              |          |         1 |            |
|   2 |   HASH JOIN                  |          |     14732 |  1114   (2)|
|*  4 |    TABLE ACCESS FULL         | CUSTOMER |      6656 |   406   (1)|
|*  5 |    TABLE ACCESS FULL         | SALES    |       10M |   703   (2)|
-------------------------------------------------------------------------
```

Problem Statement 4



Expected Partition Pruning

```
SELECT    SUM(s.amount_sold)
FROM      sales s,
WHERE     TO_CHAR(s.time_id,'YYYYMMDD')
          BETWEEN '20190101' AND '20191231';
```

SALES table has 10 million rows

SALES is partitioned on the time_id column on a quarterly basis and 28 partitions

Expected Partition Pruning But Didn't Get It

```
SELECT    SUM(s.amount_sold)
FROM      sales s,
WHERE     TO_CHAR(s.time_id,'YYYYMMDD')
          BETWEEN '20190101' AND '20191231';
```

SALES table has 10 million rows

SALES is partitioned on the time_id column on a quarterly basis and 28 partitions

```
-------------------------------------------------------------------------------
| Id  | Operation                | Name  | ROWS | Cost (%CPU) | Pstart | Pstop |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |       |      |   552 (100) |        |     2 |
|   1 |  SORT AGGREGATE          |       |   13 |             |        |     2 |
|   2 |   PARTITION RANGE ALL    |       |  10M |   552   (7) |      1 |    28 |
|*  3 |    TABLE ACCESS FULL     | SALES |  10M |   552   (7) |      1 |    28 |
-------------------------------------------------------------------------------

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

3 - filter((TO_CHAR(INTERNAL_FUNCTION("S"."TIME_ID"),'YYYYMMDD')>='20190101' AND
           TO_CHAR(INTERNAL_FUNCTION("S"."TIME_ID"),'YYYYMMDD')<='20191231'))
```

Expected Partition Pruning But Didn't Get It

```
SELECT    SUM(s.amount_sold)
FROM      sales s,
WHERE     TO_CHAR(s.time_id,'YYYYMMDD')
          BETWEEN '20190101' AND '20191231';
```

SALES table has 10 million rows

SALES is partitioned on the time_id column on a quarterly basis and 28 partitions

```
-------------------------------------------------------------------------------
| Id  | Operation                | Name  | ROWS | Cost (%CPU) | Pstart | Pstop |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |       |      |   552 (100) |        |     2 |
|   1 |  SORT AGGREGATE          |       |   13 |             |        |     2 |
|   2 |   PARTITION RANGE ALL    |       |  10M |   552   (7) |      1 |    28 |
|*  3 |    TABLE ACCESS FULL     | SALES |  10M |   552   (7) |      1 |    28 |
-------------------------------------------------------------------------------
```

WHY HAS AN ADDITIONAL INTERNAL_FUNCTION BEEN ADDED TO OUR PREDICATE?

```
---------------------------------------------------

3 - filter((TO_CHAR(INTERNAL_FUNCTION("S"."TIME_ID"),'YYYYMMDD')>='20190101' AND
           TO_CHAR(INTERNAL_FUNCTION("S"."TIME_ID"),'YYYYMMDD')<='20191231'))
```

Expected Partition Pruning But Didn't Get It

Expected query to access only 4 partitions but its accesses all

- INTERNAL_FUNCTION typically means a data type conversion has occurred

Expected Partition Pruning But Didn't Get It

Expected query to access only 4 partitions but its accesses all

- INTERNAL_FUNCTION typically means a data type conversion has occurred
- Data type conversion needed when column type & predicate type don't match
- Predicate is `TO_CHAR(s.time_id, 'YYYYMMDD')`

Expected Partition Pruning But Didn't Get It

Expected query to access only 4 partitions but its accesses all

- INTERNAL_FUNCTION typically means a data type conversion has occurred
- Data type conversion needed when column type & predicate type don't match
- Predicate is `TO_CHAR(s.time_id,'YYYYMMDD')`
- `TIME_ID` is a actually a date column
- Optimizer has no idea how function will effect the values in `TIME_ID` column
- Optimizer can't determine which partitions will be accessed now

Expected Partition Pruning But Didn't Get It

Expected query to access only 4 partitions but its accesses all

- INTERNAL_FUNCTION typically means a data type conversion has occurred
- Data type conversion needed when column type & predicate type don't match
- Predicate is `TO_CHAR(s.time_id,'YYYYMMDD')`
- `TIME_ID` is a actually a date column
- Optimizer has no idea how function will effect the values in `TIME_ID` column
- Optimizer can't determine which partitions will be accessed now

SIMPLE FIX: USING INVERSE FUNCTION ON OTHER SIDE OF PREDICATE

Solution Use Inverse Function On Other Side Of Predicate

```
SELECT SUM(s.amount_sold)
FROM   sales s,
WHERE  s.time_id BETWEEN TO_DATE('20190101', 'YYYMMDD') AND
                         TO_DATE(' 20191231', 'YYYMMDD');
```

```
-------------------------------------------------------------------------------
| Id  | Operation                | Name  | ROWS | Cost (%CPU) | Pstart | Pstop |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |       |      |   552 (100) |        |     2 |
|   1 |  SORT AGGREGATE          |       |   13 |             |        |     2 |
|   2 |   PARTITION RANGE ALL    |       |  10M |   552   (7) |      9 |    12 |
|*  3 |    TABLE ACCESS FULL     | SALES |  10M |   552   (7) |      9 |    12 |
-------------------------------------------------------------------------------

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

3 – Access("S"."TIME_ID">='20190101' TO_DATE('20190101','YYYMMDD') AND
        TO_DATE(' 20191231','YYYMMDD'))
```

Important Tip on Using Functions

Using inverse function on other side of predicate

Keep the following in mind when deciding where to place the function

- Try to place functions on top of constants (literals, binds) rather than on columns
- Avoid using a function on index columns or partition keys as it prevents index use or partition pruning

Important Tip on Using Functions

Using inverse function on other side of predicate

Keep the following in mind when deciding where to place the function

- Try to place functions on top of constants (literals, binds) rather than on columns
- Avoid using a function on index columns or partition keys as it prevents index use or partition pruning
- For function-based index to be considered, use that exact function as specified in index

Important Tip on Using Functions

Using inverse function on other side of predicate

Keep the following in mind when deciding where to place the function

- Try to place functions on top of constants (literals, binds) rather than on columns
- Avoid using a function on index columns or partition keys as it prevents index use or partition pruning
- For function-based index to be considered, use that exact function as specified in index
- If multiple predicates involve the same columns, write predicates such that they share common expressions For example,

```
WHERE f(a) = b
AND       a = c
```

Should be rewritten as

```
WHERE   a = inv_f(b)
AND       a = c
```

Important Tip on Using Functions

Using inverse function on other side of predicate

Keep the following in mind when deciding where to place the function

- Try to place functions on top of constants (literals, binds) rather than on columns
- Avoid using a function on index columns or partition keys as it prevents index use or partition pruning
- For function-based index to be considered, use that exact function as specified in index
- If multiple predicates involve the same columns, write predicates such that they share common expressions For example,

```
WHERE f(a) = b
AND       a = c
```

Should be rewritten as

```
WHERE   a = inv_f(b)
AND       a = c
```

This will allow transitive predicate c=inv_f(b) to be added by the optimizer

Quick Tips If You Must Use Hints

Some Times a Hint is Your Only Option

What are Optimizer Hints

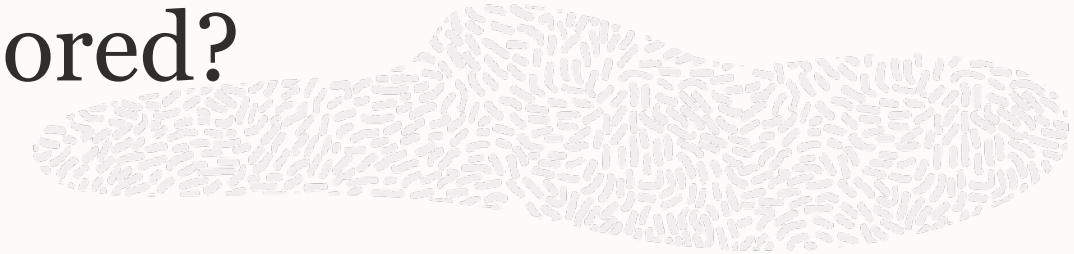
- Hints allow you to influence the Optimizer when it has to choose between several possibilities
- A hint is a directive that will be followed when applicable

Some Times a Hint is Your Only Option

What are Optimizer Hints

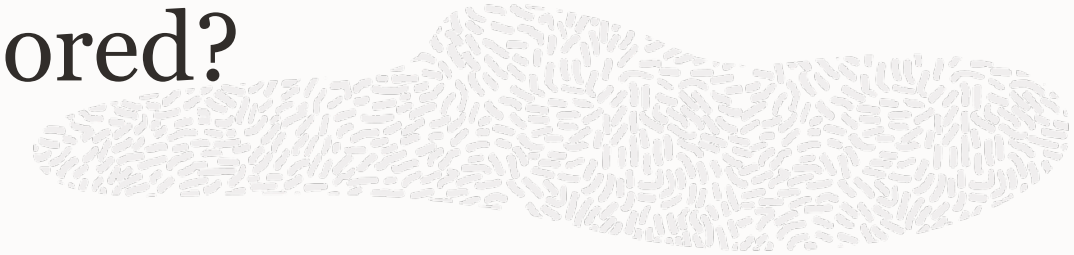
- Hints allow you to influence the Optimizer when it has to choose between several possibilities
- A hint is a directive that will be followed when applicable
- Can influence everything from the Optimizer mode used to each operation in the execution
- Automatically means the Cost Based Optimizer will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

Why are Optimizer Hints Ignored?



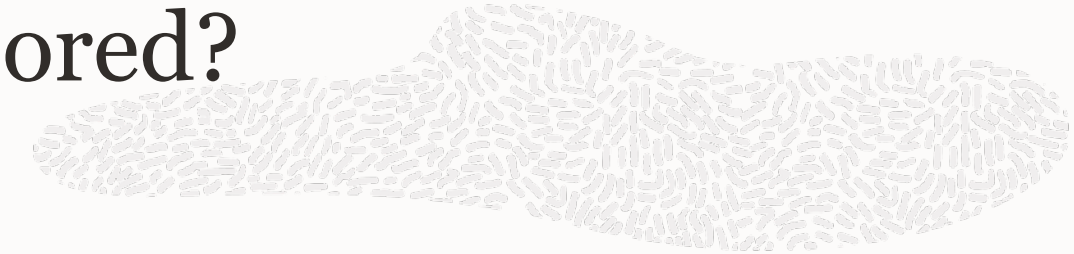
- An Optimizer hint is never ignored

Why are Optimizer Hints Ignored?



- An Optimizer hint is never ignored
- Hints only evaluated when they apply to a decision that has to be made
- Often times hint that's aren't used because they are irrelevant or not legal

Why are Optimizer Hints Ignored?



- An Optimizer hint is never ignored
- Hints only evaluated when they apply to a decision that has to be made
- Often times hint that's aren't used because they are irrelevant or not legal

In **10053 trace file** you will find:

"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 ============"

Why are Optimizer Hints Ignored?

- An Optimizer hint is never ignored
- Hints only evaluated when they apply to a decision that has to be made
- Often times hint that's aren't used because they are irrelevant or not legal

In **10053 trace file** you will find:

"Dumping Hints
============
Atom_hint=(@=0X124360178 err=0 resol=0 used=1 token=454 org=1 lvl=1 txt=ALL_ROWS)
Atom_hint=(@=0X2af785e0[illegible]260 err=0 resol=1[illegible]ed=1 token=448 org=1 lvl=3 txt=FULL ("E"))
==========[illegible]== END SQL Statem[illegible]n ============"

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 But Doesn't mean the final plan will reflect it

Why are Optimizer Hints Ignored?

New Hint Usage Report in 19c

NEW IN 19c

```
-------------------------------------------------------------------------------------------
| 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 |
-------------------------------------------------------------------------------------------

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 B ck Name / Object Alias):
Total hints for statement: 1 (U - Unused (1))
---------------------------------------------------------------------------

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

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

Why are Optimizer Hints Ignored?

Statement 1

Employees table has a unique index called pk_emp index

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

```
-------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | COST (%CPU) |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |  2579 |  8743   (1) |
|   1 |  TABLE ACCESS FULL  | EMPLOYEES   |  2579 |  8743   (1) |
-------------------------------------------------------------------
```

Why are Optimizer hints ignored?

Syntax and Spelling

Employees table has a unique index called pk_emp index

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

```
-------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |  2579 |  8743   (1) |
|   1 |  TABLE ACCESS FULL  | EMPLOYEES   |  2579 |  8743   (1) |
-------------------------------------------------------------------
```

Invalid hint the index name is wrong

Why are Optimizer hints ignored?

Syntax and Spelling

Employees table has a unique index called pk_emp index

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

-------------------------------------------------------------------
| Id  | Operation                   | Name        | Rows  | COST (%CPU) |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT            |             |  2579 |  8743   (1) |
|   1 |  TABLE ACCESS FULL          | EMPLOYEES   |  2579 |  8743   (1) |
-------------------------------------------------------------------

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

   1 -  SEL$1 /S@SEL$1
         U -  index (e emp_pk) / index specified in the hint doesn't exist
```

19c

Why are Optimizer hints ignored?

The MY_PROMOTIONS table has 5,000 rows and no indexes

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

```
-------------------------------------------------------------------------------
| 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?

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?

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 |
-------------------------------------------------------------------------------
```

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

   2 -  SEL$1 /P@SEL$1
         U -  INDEX(p)
```

19c

Why are Optimizer hints ignored?

Requesting 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)|
-------------------------------------------------------------------------------
```

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?

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

```
Hint Report (identified by operation id / Query Block Name / Object Alias):
Total hints for statement: 1 (U - Unused (1))
---------------------------------------------------------------------------
   2 -  SEL$1 /P@SEL$1
         U -  USE_HASH(e s)
```

19c

Tips To Remember

Statistics

Always check and correct your cardinality estimates first

Tips To Remember

Statistics

Always check and correct your cardinality estimates first

SQL Statement

Look at how the SQL statement is written and how the predicates are being use

Tips To Remember

Statistics

Always check and correct your cardinality estimates first

SQL Statement

Look at how the SQL statement is written and how the predicates are being use

Hints

Only add hints as a last resort and confirm your hint is really being used

Helpful Links

https://sqlmaria.com

https://twitter.com/@SQLMaria

https://www.facebook.com/SQLMaria


ORACLE