ORACLE

SQL Tuning Tips You Can't Do Without

Maria Colgan
Distinguished Product Manager
Database Server Technologies
May 2021

@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 making you an Optimizer expert instantly or give you the power to tune SQL statements with the flick of your wrist!

Problem Statement 1

Why Didn’t The Optimizer Pick The Join Method I Expected?

What Join Method Should The Optimizer Pick?

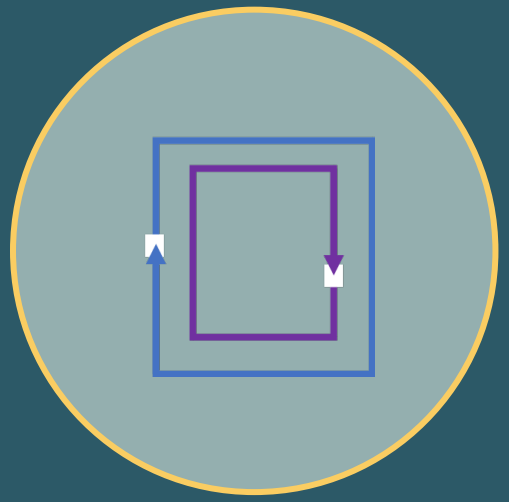
The Query

```
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-MAY-21'
GROUP BY p.prod_name;
```

SALES table has
10 million rows

PRODUCTS table
has 100,000 rows

Oracle Optimizer Has 3 Join Methods



NESTED LOOPS

For every row in the outer table, Oracle accesses all the rows in the inner table

HASH JOIN

Smaller of two tables is scan and resulting rows used to build a hash table on the join key in memory. Larger table is then scan, join column of the resulting rows are hashed and the values used to probe the hash table to find matching rows

SORT MERGE JOIN

Consists of two steps:
1. Sort both inputs on the join key
2. Merge the sorted lists together

Optimizer Chose A NESTED LOOP Instead of HASH JOIN 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-MAY-21';
GROUP BY p.prod_name;
```

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

SALES table has 10 million rows

PRODUCTS table has 100,000 rows

First Thing To Check Is the Cardinality Estimates

1. Always start by looking at the Cardinality Estimate

```
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-MAY-21';
GROUP BY p.prod_name;
```

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

First Thing To Check Is the Cardinality Estimates

```
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-MAY-21';
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 |
-------------------------------------------------------------------------
```

We are going to need more information

Let's see what additional information we can find in the database

Check Table Metadata

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

TABLE_NAME	PARTITIONED
PRODUCTS	NO
SALES	YES

Check Table Metadata and Statistics

```
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	

Check Table Metadata and Statistics

```
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

Next Check the WHERE Clause Predicate Versus The Data

Where Clause Predicate

```
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-MAY-21'
GROUP BY p.prod_name;
```

The Data

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

MIN(DATE_ID)	MAX(DATE_ID)
01-JAN-04	03-MAY-21

Compare Data To The Statistics Optimizer Used

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

```
MAX(DATE_ID)
____________

03-MAY-21
```

```
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-MAR-2021

PL/SQL procedure successfully completed.
```

What Do We Know So Far

01 Statistics for TIME_ID column show max value as '31-MAR-2021'

02 Our query is looking for sales on '03-MAY-2021'

03 Optimizer checks if predicate falls between the min, max value of column

What Will The Optimizer Do In This Situation

If the Optimizer doesn't find the value in the WHERE clause predicate between the MIN and MAX value of the column, it considers it **OUT_OF_RANGE**

What Will The Optimizer Do In This Situation

If the Optimizer doesn't find the value in the WHERE clause predicate between the MIN and MAX value of the column, it considers it **OUT_OF_RANGE**

Optimizer **prorates cardinality** based on the distance between the predicate value and the maximum value

What Will The Optimizer Do In This Situation

Alternatively we could have looked in the **Optimizer trace** file (10053 trace), where we would have found:

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

Stale Statistics Caused an “Out of Range” Issue

```
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-MAY-21';
GROUP BY p.prod_name;
```

Optimizer assumes there are no rows for time_id=‘03-MAY-21’ 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 !

But my statistics aren't stale?

What Will The Optimizer Do In This Situation

By default stats are only considered stale after 10% of the rows have changed (inserted, updated, deleted)

Our tables SALES has 10 million rows in it, therefore **1 million rows** need to be changed before the statistics are considered stale

Fix Out-of-Range Cardinality Misestimate

Option 1 - USE DBMS_STATS.SET_TABLE_PREFS To Change Staleness Threshold

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

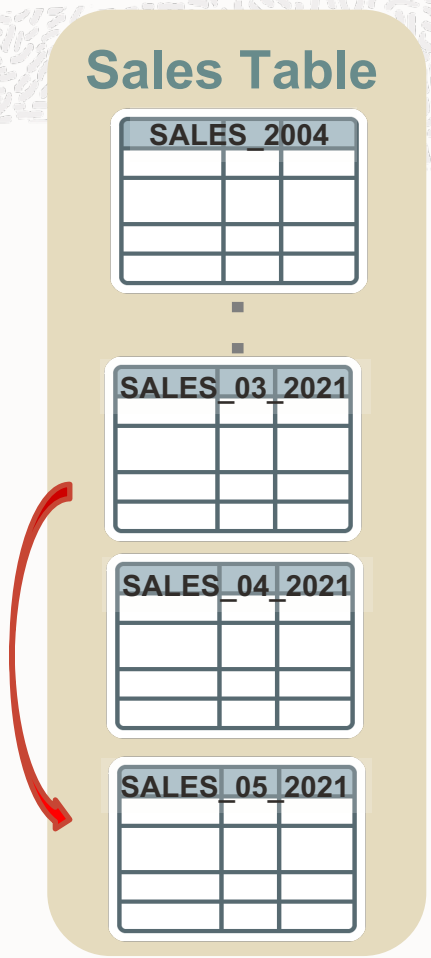
Option 2 - USE DBMS_STATS.COPY_TABLE_STATS()

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




```
BEGIN
    dbms_stats.copy_table_stats('SH','SALES','SALES_03_2021','SALES_05_2021');
  END;
/
```

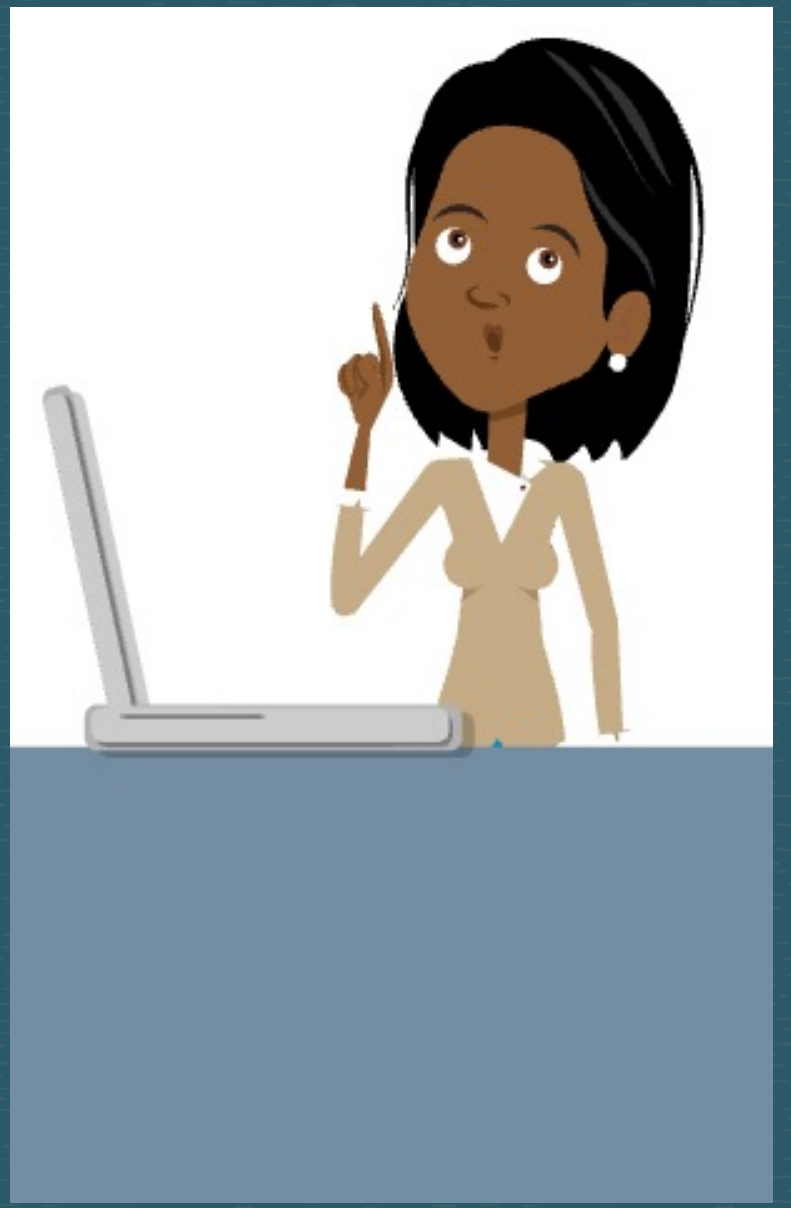
Fixing Out-of-Range Error Means Hash Join Plan Chosen Automatically

```
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-MAY-21'
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

Why Didn’t The Optimizer Pick The Index I Expected?



What Index Should The Optimizer Pick?

The Query

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

PROD_CUST_IND
(prod_id, cust_id)

PROD_CUST_COM_IND
(prod_id, comment, cust_id)

SALES table has
2 million rows

Why Didn't The Optimizer Use The Index With All The Columns Needed?

The Query

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

PROD_CUST_IND
(prod_id, cust_id)

PROD_CUST_COM_IND
(prod_id, comment, cust_id)

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

Let's Confirm Our Preferred Plan is Possible

The Query

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

The Plan is possible but why is the cost so high?

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

We are going to need more information

Let's see what additional information we can find in the database

Check Table Metadata

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

TABLE_NAME	PARTITIONED
SALES	NO

Check Table Metadata and Statistics

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

TABLE_NAME	PARTITIONED
SALES	NO

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

TABLE_NAME	STALE_STATS
SALES	NO

Check Index Metadata and Statistics

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

INDEX_NAME	STALE_STATS
PROD_CUST_COM_IND	NO
PROD_CUST	NO

Check Index Metadata and Statistics

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

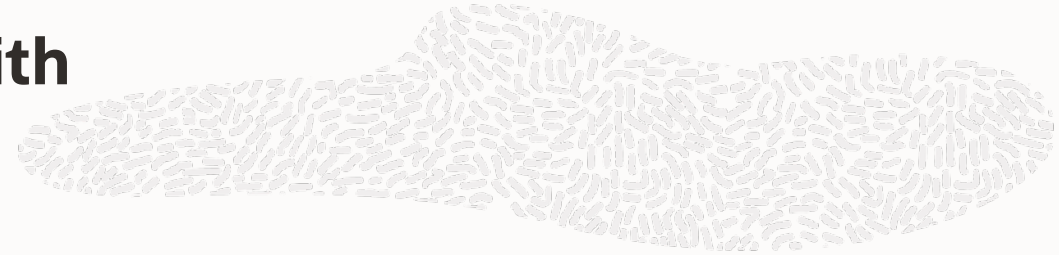
INDEX_NAME	STALE_STATS
PROD_CUST_COM_IND	NO
PROD_CUST	NO

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

INDEX_NAME	LEAF_BLOCKS	BLEVEL
PROD_CUST_COM_IND	699467	9
PROD_CUST	5468	2

Why Does PROD_CUST_COM_IND have so many leaf blocks?

Understand The Data You Are Dealing With



```
DESC sales
```

Name	Null?	Type
ORDER_ID	NOT NULL	NUMBER
CUST_ID	NOT NULL	NUMBER
PRODUCT_ID	NOT NULL	NUMBER
SUPPLIER_ID	NOT NULL	NUMBER
DATE_ID		DATE
AMOUNT_SOLD		NUMBER
PRICE		NUMBER
REVENUE		NUMBER
SHIPMODE		CHAR(10)
COMMENTS		VARCHAR2(2000)

Need To Look At How The Indexes Are Being Used

```
SELECT /*+ INDEX(PROD_CUST_COM_IND) */ comment
FROM   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`)

Access Predicates Versus Filter Predicates

```
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

Access Predicates Versus Filter Predicates

```
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

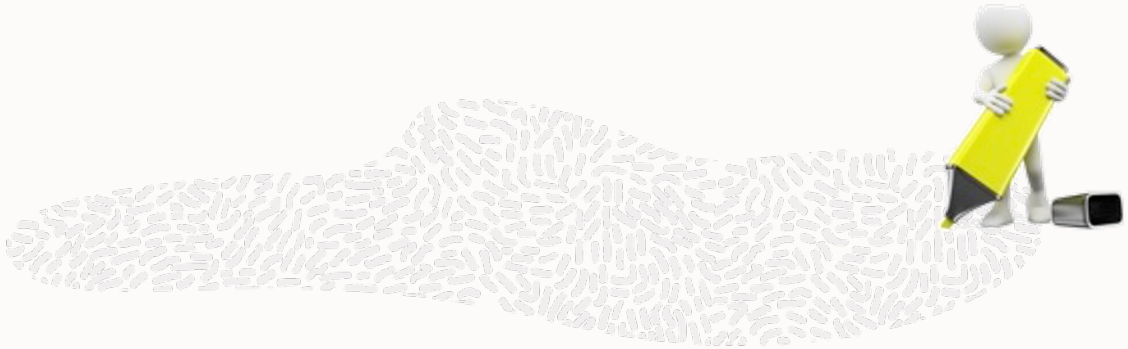
Need To Look At How The Indexes Are Being Used

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

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

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**

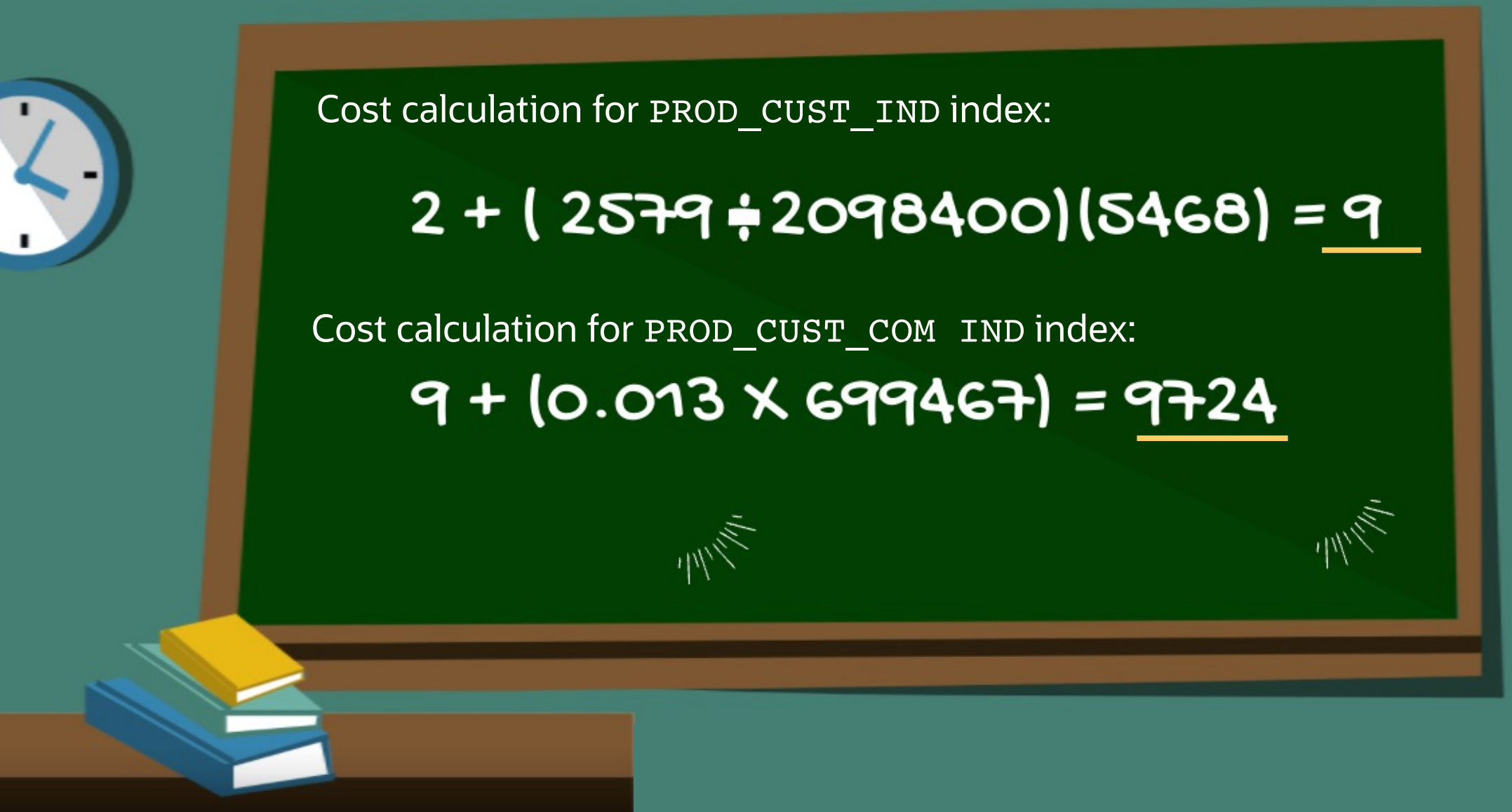
Cost calculation for this index:
blevel + (cardinality X leaf blocks)
#rows

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

Costing of Each Index Access is Different



Common Misconceptions On What Index Will Be Chosen

A common misconception is the Optimizer will always pick the index with all the necessary columns in it

Common Misconceptions On What Index Will Be Chosen

A common misconception is the Optimizer will always pick the index with all the necessary columns in it

Fact is the Optimizer picks the index based on the cost of the access

Common Misconceptions On What Index Will Be Chosen

A common misconception is the Optimizer will always pick the index with all the necessary columns in it

Fact is the 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: Recreate PROD_CUST_COM_IND With Different Column Order

```
CREATE INDEX Prod_cust_com_ind2
ON   Sales(prod_id, cust_id, comment);
```

PROD_CUST_COM_IND2
(prod_id, cust_id, comment)

Solution: Recreate PROD_CUST_COM_IND With Different Column Order

The Query

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

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

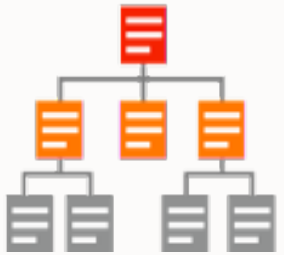
Why Didn’t The Optimizer Pick The Join Method I Expected?

What Join Method Should The Optimizer Pick?

The Query

```
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



SALES_CUST_BIX
(cust_id)

CUSTOMERS table
has 10,000 rows

Got a Nested Loops Join Instead of Hash Join

The Query

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

Let's Confirm Our Preferred Plan is Possible

The Query

```
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';
```

The Plan is possible but why is the cost so high?

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

We are going to need more information

Let’s see what actually happened during the statement execution

Need to Gather More Information

```
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'));
```

1. Always start by looking at the Cardinality Estimate

```
-------------------------------------------------------------------------------
| 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 Estimate So Wrong?

Always start by looking at the cardinality estimate for the table on the left-hand side of the join

Why is the Cardinality Estimate So Wrong?

Always start by looking at the cardinality estimate for the table on the left-hand side of the join

Customers is the table on the left hand side of our join

It has one `WHERE` clause predicates is

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

Why is the Cardinality Estimate So Wrong?

Always start by looking at the cardinality estimate for the table on the left-hand side of the join

Customers is the table on the left hand side of our join

It has one `WHERE` clause predicates is

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

What is the first thing you noticed about this predicate?

Why is the Cardinality Estimate So Wrong?

Always start by looking at the cardinality estimate for the table on the left-hand side of the join

Customers is the table on the left hand side of our join

It has one `WHERE` clause predicate:

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

There is a function being applied to the column in the `WHERE` clause predicate

Why is the Cardinality Estimate So Wrong?

Always start by looking at the cardinality estimate for the table on the left-hand side of the join

Customers is the table on the left hand side of our join

It has one `WHERE` clause predicate:

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

There is a function being applied to the column in the `WHERE` clause predicate

The Optimizer has no idea how the function affects the values in the column

Function Wrapped Column Blinds the Optimizer

Optimizer Doesn't Know How Function Affects Values In The Column

The Query

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

Remember CUSTOMERS table has 10,000 rows

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

Optimizer guesses the cardinality to be 1% of rows

Solution: Tell Optimizer How Function Affects Column Values

Option 1 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
```

Solution: Tell Optimizer How Function Affects Column Values

How To Identify 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
:		

Solution: Tell Optimizer How Function Affects Column Values

The Query

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

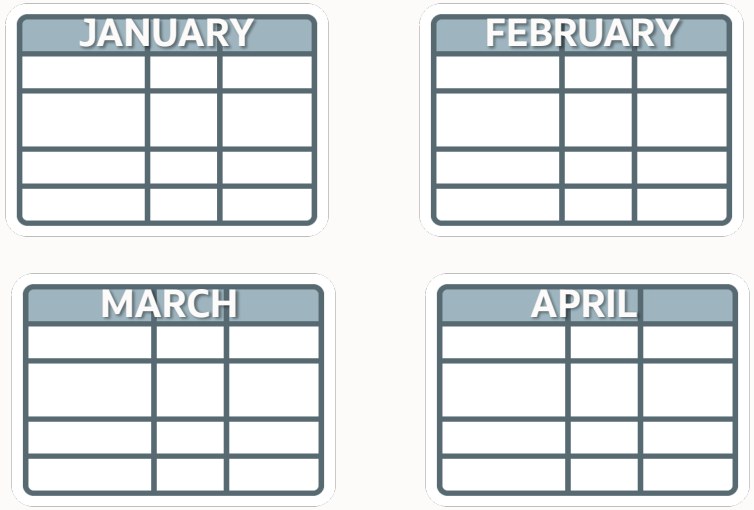
Problem Statement 4

Why Didn’t The Optimizer Use Partition Pruning?

Partitioning Provides Flexibility & Efficiency at Scale

Large Table

Difficult to Manage



Partitions

Divide and Conquer

Easier to Manage

Improve Performance

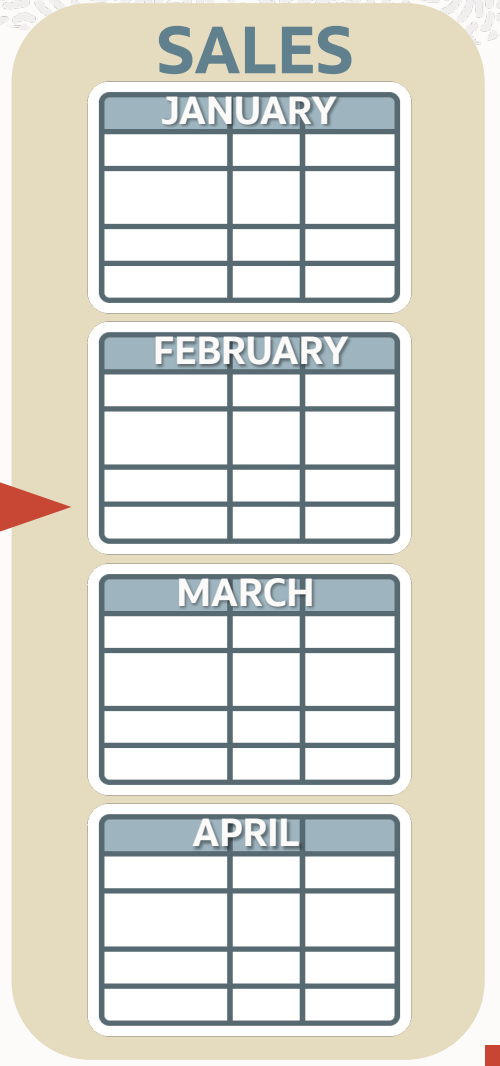
Transparent to applications

Partitioning Provides Performance Acceleration

The Query

```
SELECT SUM(s.sales_amount)
FROM sales s
WHERE s.sales_date = 'FEBRUARY';
```

Only the relevant partition is accessed




```
-------------------------------------------------------------------------------
| Id  | Operation                    | Name  | Rows  | Cost (%CPU)| Pstart| Pstop |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |       |       |    16 (100)|       |       |
|   1 |  SORT AGGREGATE              |       |     1 |            |       |       |
|   2 |   PARTITION RANGE SINGLE     |       |  4018 |    16   (7)|    12 |    12 |
|*  3 |    TABLE ACCESS STORAGE FULL | SALES |  4018 |    16   (7)|    12 |    12 |
-------------------------------------------------------------------------------
```

Expected Partition Pruning

The Query

```
SELECT    SUM(s.amount_sold)
FROM      sales s,
WHERE     TO_CHAR(s.time_id,'YYYYMMDD')
          BETWEEN '20200101' AND '20201231';
```

SALES table has
10 million rows

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

Expected Partition Pruning

The Query

```
SELECT    SUM(s.amount_sold)
FROM      sales s,
WHERE     TO_CHAR(s.time_id,'YYYYMMDD')
          BETWEEN '20200101' AND '20201231';
```

SALES table has
10 million rows

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

We are going to need more information

Let’s look at how the WHERE clause predicates are being used in the plan

Expected Partition Pruning

```
-------------------------------------------------------------------------------
| 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'))
```

For Partition Pruning To Occur We Need An Access Predicate

Expected Partition Pruning

```
-------------------------------------------------------------------------------
| 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'))
```

Why Has An Additional INTERNAL_FUNCTION Been Added To Our Predicate?

Why We Didn’t Get Partition Pruning

- The presences of an INTERNAL_FUNCTION typically means a data type conversion has occurred

Why We Didn't Get Partition Pruning

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

Why We Didn't Get Partition Pruning

- The presences of an INTERNAL_FUNCTION typically means a data type conversion has occurred
- A data type conversion is 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 affect the values in `TIME_ID` column

Why We Didn't Get Partition Pruning

- The presences of an INTERNAL_FUNCTION typically means a data type conversion has occurred
- A data type conversion is 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 affect the values in `TIME_ID` column
- Optimizer can't determine which partitions will be accessed now

Solution: Using Inverse Function On Other Side Of Predicate

The Query

```
SELECT SUM(s.amount_sold)
FROM   sales s,
WHERE  s.time_id BETWEEN TO_DATE('20200101', 'YYYMMDD') AND
                         TO_DATE(' 20201231', '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) |     22 |    24 |
|*  3 |    TABLE ACCESS FULL     | SALES |  10M |   552   (7) |     22 |    24 |
-------------------------------------------------------------------------------
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

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 pruning or use of 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 pruning or use of index
- 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 pruning or use of index
- 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

Question Number 5

Why Is The Optimizer Ignoring My Hints?

Some Times an Optimizer Hint is Your Only Option

- 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
- 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

How Can I tell What's Happening With My Hint?

Why Are Optimizer Hints Ignored?

How to find out what happened to you hint

- 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 State ump ============"

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?

How to find out what happened to you hint

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_STATE_PROVINC

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

Why Are Optimizer Hints Ignored?

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 Mistakes

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-20';
```

```
-------------------------------------------------------------------------------------
| 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 as No Indexes exist on

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-20';
```

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

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

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

Why Are Optimizer Hints Ignored?

Requesting a Bloom Filter not to be used

```
SELECT /*+ PX_no_px_join_filter */ SUM(REVENUE)
FROM   Sales S, Customers c
WHERE  s.cust_id=c.c_cust_id;
```

```
-------------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |       |       |  3792 (100)|
|   1 |  SORT AGGREGATE               |           |     1 |    17 |            |
|*  2 |   HASH JOIN                   |           |  4126 | 70142 |  3792   (5)|
|   3 |    JOIN FILTER CREATE         | :BF0000   |  1159 |  5795 |     2   (0)|
|   4 |     TABLE ACCESS STORAGE FULL | CUSTOMERS |  1159 |  5795 |     2   (0)|
|   5 |    JOIN FILTER USE            | :BF0000   |   10M|   114M|  3759   (5)|
|   6 |     PARTITION RANGE ALL       |           |   10M|   114M|  3759   (5)|
|*  7 |      TABLE ACCESS STORAGE FULL| SALES     |   10M|   114M|  3759   (5)|
-------------------------------------------------------------------------------
```

Why Are Optimizer Hints Ignored?

Requesting a Bloom Filter not to be used

```
SELECT /*+ PX_no_px_join_filter */ SUM(REVENUE)
FROM   Sales S, Customers c
WHERE  s.cust_id=c.c_cust_id;
```

```
-------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |       |       |  3792 (100)|
|   1 |  SORT AGGREGATE              |           |     1 |    17 |            |
|*  2 |   HASH JOIN                  |           |  4126 | 70142 |  3792   (5)|
|   3 |    JOIN FILTER CREATE        | :BF0000   |  1159 |  5795 |     2   (0)|
|   4 |     TABLE ACCESS STORAGE FULL| CUSTOMERS |  1159 |  5795 |     2   (0)|
|   5 |    JOIN FILTER USE           | :BF0000   |   10M |  114M |  3759   (5)|
|   6 |     PARTITION RANGE ALL      |           |   10M |  114M |  3759   (5)|
|*  7 |      TABLE ACCESS STORAGE FULL| SALES     |   10M |  114M |  3759   (5)|
-------------------------------------------------------------------------------
```

The hint is misspelt, which caused a syntax error.
The correct hint is NO_PX_JOIN_FILTER

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

   1 -  SEL$1
          E -  PX_no_px_join_filter
```

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

For More Information Please Visit

https://sqlmaria.com

https://blogs.oracle.com/Optimizer

https://twitter.com/@SQLMaria

https://www.facebook.com/SQLMaria

Questions?


ORACLE