ORACLE

SQL Tuning Tips and Tricks

PART 5

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

SQLMaria

Safe harbor statement

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

Agenda

1. Use the right Tools
2. Functions Friends or Foe
3. Data Type Dilemmas
4. Influencing an execution without adding hints

Using the right tools

Expecting index range scan execution plan

Query – How many customers do we have in a specific zipcode?

```
SELECT Count(cust_first_name)
FROM   customers2
WHERE  zipcode = :n;
```

- Customers2 table has 20,000 rows
- A b-tree index exists on zipcode
- There is also a histogram on the zipcode column due to data skew

Using the right tools

Expected index range scan but got full table scan. Why?

```
SQL> var n number
SQL> exec :n :=94065;

PL/SQL procedure successfully completed.

SQL>
SQL> set autotrace traceonly explain
SQL> SELECT count(cust_email)
  2  FROM   customers2
  3  WHERE  zipcode = :n;

Execution Plan
----------------------------------------------------------

Plan hash value: 2704912892

-------------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            |     1 |    21 |   152   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |            |     1 |    21 |            |          |
|*  2 |   TABLE ACCESS FULL| CUSTOMERS2 | 10000 |   205K|   152   (1)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   2 - filter("ZIPCODE"=TO_NUMBER(:N))
```

Set bind variable :n to 94065

Using the right tools

Using literal value gets the right plan

```
SQL> set autotrace traceonly explain
SQL> SELECT count(cust_email)
  2  FROM   customers2
  3  WHERE  zipcode = 94065;

Execution Plan
----------------------------------------------------------

Plan hash value: 429287319

-------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |     1 |    21 |     2   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE              |              |     1 |    21 |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| CUSTOMERS2   |     1 |    21 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | ZIPCODE_IDX  |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

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

   3 - access("ZIPCODE"=94065)
```

Let's try using literal value to get the plan we want

Using the right tools

Let's double check our code

- Let's check our bind statement and plan again
- Why is there a `TO_NUMBER` function on the bind variable n after it was defined as a number?
- Why is a simple equality predicate being applied as filter and not as as access predicate?

```
SQL> var n number
SQL> exec :n :=94065;

PL/SQL procedure successfully completed.

SQL>
SQL> set autotrace traceonly explain
SQL> SELECT count(cust_email)
  2  FROM   customers2
  3  WHERE  zipcode = :n;

Execution Plan
----------------------------------------------------------

Plan hash value: 2704912892

-------------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            |     1 |    21 |   152   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |            |     1 |    21 |            |          |
|*  2 |   TABLE ACCESS FULL| CUSTOMERS2 | 10000 |   205K|   152   (1)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   2 - filter("ZIPCODE"=TO_NUMBER(:N))
```

Using the right tools

Bad plan is caused by using Autotrace

- Autotrace is not aware of binds at all
- All binds treated as strings hence `TO_NUMBER` on n
- Also no bind peeking takes place

```
SQL> var n number
SQL> exec :n :=94065;

PL/SQL procedure successfully completed.

SQL>
SQL> set autotrace traceonly explain
SQL> SELECT count(cust_email)
  2  FROM   customers2
  3  WHERE  zipcode = :n;

Execution Plan
----------------------------------------------------------

Plan hash value: 2704912892

-------------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            |     1 |    21 |   152   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |            |     1 |    21 |            |          |
|*  2 |   TABLE ACCESS FULL| CUSTOMERS2 | 10000 |   205K|   152   (1)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   2 - filter("ZIPCODE"=TO_NUMBER(:N))
```

Using the right tools

Bad plan is caused by using Autotrace

- No bind peeking means the histogram can't be used for cardinality estimate
- Calculated using

$$\frac{\text{ROW\_NUM}}{\text{NDV}} = \frac{20{,}000}{2}$$

```
SQL> var n number
SQL> exec :n :=94065;

PL/SQL procedure successfully completed.

SQL>
SQL> set autotrace traceonly explain
SQL> SELECT count(cust_email)
  2  FROM   customers2
  3  WHERE  zipcode = :n;

Execution Plan
----------------------------------------------------------

Plan hash value: 2704912892

-------------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            |     1 |    21 |   152   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |            |     1 |    21 |            |          |
|*  2 |   TABLE ACCESS FULL| CUSTOMERS2 | 10000 |   205K|   152   (1)| 00:00:01 |
-------------------------------------------------------------------------------------

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

   2 - filter("ZIPCODE"=TO_NUMBER(:N))
```

Using the right tools

Solution – use DBMS_XPLAN.DISPLAY_CURSOR

```
SQL> SELECT count(cust_email)
  2  FROM   customers2
  3  WHERE  zipcode = :n;

COUNT(CUST_EMAIL)
-----------------
                1

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

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
SQL_ID  bthh5g436vfr3, child number 0
-------------------------------------
SELECT count(cust_email) FROM   customers2 WHERE  zipcode = :n

Plan hash value: 429287319

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |       |       |     2 (100)|          |
|   1 |  SORT AGGREGATE              |             |     1 |    21 |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| CUSTOMERS2  |     1 |    21 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | ZIPCODE_IDX |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

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

   3 - access("ZIPCODE"=:N)
```

Execute the statement with the bind then run DBMS_XPLAN command

Using the right tools

Solution – use DBMS_XPLAN.DISPLAY_CURSOR

```
SQL> SELECT count(cust_email)
  2  FROM   customers2
  3  WHERE  zipcode = :n;

COUNT(CUST_EMAIL)
-----------------
                1

SQL> select * from table(dbms_xplan.display_cursor(format=> 'typical +peeked_binds'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------
SQL_ID  bthh5g436vfr3, child number 0
-------------------------------------
SELECT count(cust_email) FROM   customers2 WHERE  zipcode = :n

Plan hash value: 429287319

-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |       |       |     2 (100)|          |
|   1 |  SORT AGGREGATE              |             |     1 |    21 |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| CUSTOMERS2  |     1 |    21 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN          | ZIPCODE_IDX |     1 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------

   1 - :N (NUMBER): 94065

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

   3 - access("ZIPCODE"=:N)
```

Additional format option shows actual bind value under the plan

Agenda

1. Use the right Tools
2. Functions Friends or Foe
3. Data Type Dilemmas
4. Influencing an execution without adding hints

Functions Friend or Foe

```
CREATE table t
AS
SELECT *
FROM all_objects;

Table created.
```

ALL_OBJECTS

OWNER	VARCHAR2(128)	NOT NULL
OBJECT_NAME	VARCHAR2(128)	NOT NULL
SUBOBJECT_NAME	VARCHAR2(128)	
OBJECT_ID	NUMBER	NOT NULL
DATA_OBJECT_ID	NUMBER	
OBJECT_TYPE	VARCHAR2(23)	
CREATED	DATE	NOT NULL
LAST_DDL_TIME	DATE	NOT NULL
TIMESTAMP	VARCHAR2(19)	
STATUS	VARCHAR2(7)	
TEMPORARY	VARCHAR2(1)	
GENERATED	VARCHAR2(1)	
SECONDARY	VARCHAR2(1)	
NAMESPACE	NUMBER	NOT NULL
EDITION_NAME	VARCHAR2(128)	
SHARING	VARCHAR2(13)	
EDITIONABLE	VARCHAR2(1)	
ORACLE_MAINTAINED	VARCHAR2(1)	
APPLICATION	VARCHAR2(1)	
DEFAULT_COLLATION	VARCHAR2(100)	
DUPLICATED	VARCHAR2(1)	
SHARDED	VARCHAR2(1)	
CREATED_APPID	NUMBER	
CREATED_VSNID	NUMBER	
MODIFIED_APPID	NUMBER	
MODIFIED_VSNID	NUMBER	

Functions Friend or Foe

```
SELECT count(*)
FROM t
WHERE created >= to_date( '5-sep-2010', 'dd-mon-yyyy' )
AND   created <  to_date( '6-sep-2010', 'dd-mon-yyyy' );

  COUNT(*)
----------
     65925

SELECT count(*), 0.01 * count(*), 0.01 * 0.01 * count(*)
FROM t;

  COUNT(*) 0.01*COUNT(*) 0.01*0.01*COUNT(*)
---------- ------------- -------------------
     72926        729.26              7.2926
```

Functions Friend or Foe

```
EXEC dbms_stats.gather_table_stats( user, 'T' );

PL/SQL procedure successfully completed.
```

- Why did I wait till here to gather statistics?

Functions Friend or Foe

```
SELECT count(*)
FROM t
WHERE created >= to_date( '5-sep-2010', 'dd-mon-yyyy' )
AND   created <  to_date( '6-sep-2010', 'dd-mon-yyyy' );

  COUNT(*)
----------
     65925

select * from table(dbms_xplan.display_cursor);
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |   291 (100)|          |
|   1 |  SORT AGGREGATE    |      |     1 |     8 |            |          |
|*  2 |   TABLE ACCESS FULL| T    | 65462 |   511K|   291   (1)| 00:00:04 |
-------------------------------------------------------------------------------
```

Functions Friend or Foe

```
SELECT count(*)
FROM t
WHERE trunc(created) = to_date( '5-sep-2010', 'dd-mon-yyyy' );

COUNT(*)
----------
     65925

select * from table(dbms_xplan.display_cursor);
----------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |   294 (100)|          |
|   1 |  SORT AGGREGATE    |      |     1 |     8 |            |          |
|*  2 |   TABLE ACCESS FULL| T    |   729 |  5832 |   294   (2)| 00:00:04 |
----------------------------------------------------------------------------
```

Optimizer doesn't know impact of the function on NDV so assumes 1% of the tables

Functions Friend or Foe

```
SELECT count(*)
FROM t
WHERE trunc(created) = to_date( '5-sep-2010', 'dd-mon-yyyy' )
And   substr( owner, 1, 3 ) = 'SYS';

COUNT(*)
----------
 33535

select * from table(dbms_xplan.display_cursor);
-------------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |       |       |   292 (100)|          |
|   1 |  SORT AGGREGATE    |      |     1 |    14 |            |          |
|*  2 |   TABLE ACCESS FULL| T    |     7 |    98 |   292   (1)| 00:00:04 |
-------------------------------------------------------------------------------
```

Optimizer doesn't know impact of either function, but additional predicate reduces rows..

Functions friends or foe?

Expected index range scan but got fast full index scan

Query – How many packages of bounce did we sell?

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

  COUNT(*)
----------
     11547
```

- Sales 2 has 400,000 rows
- Sales2 has a b-tree index on the `prod_id` column

Functions friends or foe?

Let's check Cardinality Estimate

- Use `GATHER_PLAN_STATISTICS` hint to capture execution statistics
- Use format parameter of `DBMS_XPLAN` to show estimated and execution statistics in plan
- Additional column added to the plan to show `Actual Rows` produce by each operation

```
SQL> SELECT /*+ gather_plan_statistics */ count(*)
  2  FROM   SALES2
  3  WHERE  to_char(prod_id)='139';

  COUNT(*)
----------
     11574

SQL>
SQL> select * from table(dbms_xplan.display_cursor(FORMAT=> 'ALLSTATS LAST'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
-----
SQL_ID  dutkuphf7z6j2, child number 1
-------------------------------------
SELECT /*+ gather_plan_statistics */ count(*) FROM   SALES2 WHERE
to_char(prod_id)='139'

Plan hash value: 4080465399

------------------------------------------------------------------------------------------
| Id  | Operation             | Name        | Starts | E-Rows | A-Rows |   A-Time   |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |             |      1 |        |      1 |00:00:00.24 |
|   1 |  SORT AGGREGATE       |             |      1 |      1 |      1 |00:00:00.24 |
|*  2 |   INDEX FAST FULL SCAN| MY_PROD_IND |      1 |  12762 |  11574 |00:00:00.22 |
------------------------------------------------------------------------------------------

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

   2 - filter(TO_CHAR("PROD_ID")='139')
```

Functions friends or foe?

Expected index range scan but got fast full index scan

- Cardinality estimate is in the right ballpark so not a problem with statistics
  - But why is an equality predicate being evaluated as a filter and not an access predicate?
  - Could it have something to do with the `TO_CHAR` function?

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

Plan hash value: 4080465399

-------------------------------------------------------------------------------
| Id  | Operation             | Name         | Starts | E-Rows | A-Rows |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |              |      1 |        |      1 |
|   1 |  SORT AGGREGATE       |              |      1 |      1 |      1 |
|*  2 |   INDEX FAST FULL SCAN| MY_PROD_IND  |      1 |  12762 |  11574 |
-------------------------------------------------------------------------------

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

   2 - filter(TO_CHAR("PROD_ID")='139')
```

Functions friends or foe?

Expected index range scan but got fast full index scan

What data type is the prod_id column ?

Name	Null?	Type
PROD_ID	NOT NULL	NUMBER
CUST_ID	NOT NULL	NUMBER
TIME_ID	NOT NULL	DATE
CHANNEL_ID	NOT NULL	NUMBER
PROMO_ID	NOT NULL	NUMBER
QUANTITY_SOLD	NOT NULL	NUMBER(10,2)
AMOUNT_SOLD	NOT NULL	NUMBER(10,2)

But literal value is a character string ‘139’

Better to apply inverse function on other side of predicate

Functions friends or foe?

Expected index range scan but got fast full index scan

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

  COUNT(*)
----------
     11547
```

```
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)
```

Functions friends or foe?

Expected query to access only 4 partitions but its accesses all

Query – calculate total amount sold for one year

```
SELECT sum(amount_sold)
FROM     sh.sales s
WHERE TO_CHAR(s.time_id,'YYYYMMDD') BETWEEN
      '19990101' AND '19991231';
```

- Sales table is partitioned on the time_id column
- Sales table has quarterly partitions for historical data

Functions friends or foe?

Expected query to access only 4 partitions but its accesses all

```
SELECT sum(amount_sold) FROM    sh.sales s WHERE
TO_CHAR(s.time_id,'YYYYMMDD') between '19990101' and '20000101'

Plan hash value: 3519235612

-------------------------------------------------------------------------------------------
| Id  | Operation             | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |       |       |       |   552 (100)|          |       |       |
|   1 |  SORT AGGREGATE       |       |     1 |    13 |            |          |       |       |
|   2 |   PARTITION RANGE ALL |       |  2297 | 29861 |   552   (7)| 00:00:01 |     1 |    28 |
|*  3 |    TABLE ACCESS FULL  | SALES |  2297 | 29861 |   552   (7)| 00:00:01 |     1 |    28 |
-------------------------------------------------------------------------------------------

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

   3 - filter(TO_CHAR(INTERNAL_FUNCTION("S"."TIME_ID"),'YYYYMMDD')>='19990101' AND
              TO_CHAR(INTERNAL_FUNCTION("S"."TIME_ID"),'YYYYMMDD')<='20000101'))
```

Why has an additional INTERNAL_FUNCTION been added to our predicate?

Functions friends or foe?

Expected query to access only 4 partitions but its accesses all

- `INTERNAL_FUNCTION` typically means a data type conversion has occurred
- Predicate is `TO_CHAR(s.time_id,'YYYYMMDD')`
- Optimizer has no idea how `TO_CHAR` function will effect the values in the `time_id` column
- Optimizer can't determine which partitions will be accessed now
  - Needs to scan them all just in case

Functions friends or foe?

Solution – Use inverse function on other side of predicate

```
SELECT sum(amount_sold) FROM    sh.sales s WHERE   s.time_id between
TO_DATE('19990101','YYYYMMDD') and TO_DATE('19991231','YYYYMMDD')

Plan hash value: 1500327972

-----------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |       |       |       |    86 (100)|          |       |       |
|   1 |  SORT AGGREGATE              |       |     1 |    13 |            |          |       |       |
|   2 |   PARTITION RANGE ITERATOR   |       |   246K|  3127K|    86  (14)| 00:00:01 |     9 |    12 |
|*  3 |    TABLE ACCESS STORAGE FULL| SALES |   246K|  3127K|    86  (14)| 00:00:01 |     9 |    12 |
-----------------------------------------------------------------------------------------------------

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

   3 - storage("S"."TIME_ID"<=TO_DATE(' 1999-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
       filter("S"."TIME_ID"<=TO_DATE(' 1999-12-31 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
```

Functions friends or foe?

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

Agenda

1. Use the right Tools
2. Functions Friends or Foe
3. Data Type Dilemmas
4. Influencing an execution without adding hints

Data type dilemmas

Expected index range scan but got fast full index scan

Query – Simple IAS part of an ETL process

```
INSERT /*+ APPEND gather_plan_statistics */
INTO t1(row_id, modification_num, operation, last_upd)
SELECT row_id, 1 , 'I', last_upd
FROM    t2
WHERE   t2.last_upd > systimestamp;
```

- T2 has 823,926 rows
- T2 has a b-tree index on the last_upd column

Data type dilemmas

Expected index range scan but got fast full index scan

- Cardinality Estimate is seriously wrong
- Only 1 non-equality access predicate
- So why is our access predicate applied as a filter?
- What does the `INTERNAL_FUNCTION` mean?

```
SQL> INSERT /*+ APPEND GATHER_PLAN_STATISTICS */ into t1
  2         (ROW_ID, MODIFICATION_NUM, OPERATION, LAST_UPD)
  3  SELECT ROW_ID      ,1   ,'I'         ,LAST_UPD
  4  FROM   t2
  5  WHERE  t2.last_upd > systimestamp;

0 rows created.

SQL>
SQL> select * from table(dbms_xplan.display_cursor(FORMAT=>'ALLSTATS LAST'));

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------------------------------
SQL_ID  044zjxxnq4t0q, child number 1
-------------------------------------
INSERT /*+ APPEND GATHER_PLAN_STATISTICS */ into t1        (ROW_ID,
MODIFICATION_NUM, OPERATION, LAST_UPD) SELECT ROW_ID   ,1   ,'I'
,LAST_UPD FROM   t2 WHERE  t2.last_upd > systimestamp

Plan hash value: 1931392233

-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation             | Name   | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT      |        |      1 |        |      0 |00:00:00.36 |    2546 |       |       |          |
|   1 |  LOAD AS SELECT       |        |      1 |        |      0 |00:00:00.36 |    2546 |  1024 |  1024 |          |
|*  2 |   INDEX FAST FULL SCAN| IND_T2 |      1 |  41165 |      0 |00:00:00.36 |    2545 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------

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

   2 - filter(SYS_EXTRACT_UTC(INTERNAL_FUNCTION("T2"."LAST_UPD"))>SYS_EXTRACT_UTC(SYSTIMESTAMP(6)))
```

Data type dilemmas

Expected index range scan but got fast full index scan

- `INTERNAL_FUNCTION` typically means a data type conversion has occurred
- Predicate is “t2.last_upd > systimestamp”
- What data type is the last_upd column

Name	Null?	Type
ROW_ID	NOT NULL	NUMBER
DESCRIPTION		CHAR(2000)
MODIFICATION_NUM		NUMBER
OPERATION		NUMBER
LAST_UPD	NOT NULL	DATE

Data type dilemmas

Why is the cardinality estimate wrong?

- Presence of the `INTERNAL_FUNCTION` cause the Optimizer to guess the cardinality estimate
- Optimizer has no way of knowing how function effects data in `LAST_UPD` column
- Without a function-based index or extended statistics the Optimizer must guess
- Guess is 5% of the rows in the tables
    - 5% of 823296 is 41,164.8 or 41,165

Data type dilemmas

Solution - correct data type mismatch

```
INSERT /*+ APPEND gather_plan_statistics */
INTO t1(row_id, modification_num, operation, last_upd)
SELECT row_id, 1 , 'I', last_upd
FROM     t2
WHERE    t2.last_upd > sysdate;
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation          | Name   | Starts | E-Rows | Cost (%CPU)| A-Rows |   A-Time   |
-------------------------------------------------------------------------------------------
|   0 | INSERT STATEMENT   |        |      1 |        |   241 (100)|      0 |00:00:00.01 |
|   1 |  LOAD AS SELECT    |        |      1 |        |            |      0 |00:00:00.01 |
|*  2 |   INDEX RANGE SCAN| IND_T2 |      1 |  73310 |   241   (6)|      0 |00:00:00.01 |
-------------------------------------------------------------------------------------------

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

   2 - access("T2"."LAST_UPD">SYSDATE@!)
```

Data type dilemmas

Expected to get partition pruning via a join but didn't

Query – calculate total amount sold that was return same day

```
SELECT    sum(amount_sold)
  FROM     sh.sales s, sh.sales_returns sr
  WHERE    s.time_id = sr.time_id
  AND      sr.time_id='31-DEC-19';
```

- Sales table is range partitioned on `time_id`
- Sales table has 4 years of data in quarterly partitions

Data type dilemmas

Expected to get partition pruning via a join but didn't

```
SELECT count(s.amount_sold) FROM sales s, sales_return sr WHERE
s.time_id =sr.time_id AND sr.time_id = '31-DEC-01'

Plan hash value: 890024704

-----------------------------------------------------------------------------------------------------
| Id  | Operation             | Name         | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |              |       |       |  1290 (100)|          |       |       |
|   1 |  SORT AGGREGATE       |              |     1 |    19 |            |          |       |       |
|*  2 |   HASH JOIN           |              |  3961 | 75259 |  1290   (5)| 00:00:01 |       |       |
|*  3 |    TABLE ACCESS FULL  | SALES_RETURN |   629 |  6919 |   738   (3)| 00:00:01 |       |       |
|   4 |    PARTITION RANGE ALL|              |  9188 | 73504 |   551   (6)| 00:00:01 |     1 |    28 |
|*  5 |     TABLE ACCESS FULL | SALES        |  9188 | 73504 |   551   (6)| 00:00:01 |     1 |    28 |
-----------------------------------------------------------------------------------------------------

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

   2 - access("SR"."TIME_ID"=INTERNAL_FUNCTION("S"."TIME_ID"))
   3 - filter("SR"."TIME_ID"=TO_TIMESTAMP('31-DEC-01'))
   5 - filter(INTERNAL_FUNCTION("S"."TIME_ID")=TO_TIMESTAMP('31-DEC-01'))
```

Getting transitive predicate but `INTERNAL_FUNCTION` on partitioned column prevents pruning

Function needed because the join columns have different data types

Data type dilemmas

Solution – ensure join columns have the same data type

```
SELECT count(s.amount_sold) FROM sales s, sales_return sr WHERE
s.time_id =sr.time_id AND sr.time_id = '31-DEC-01'

Plan hash value: 462494559

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |       |       |   783 (100)|          |       |       |
|   1 |  SORT AGGREGATE              |              |     1 |    16 |            |          |       |       |
|*  2 |   HASH JOIN                  |              |   494K|  7724K|   783   (4)| 00:00:01 |       |       |
|   3 |    PART JOIN FILTER CREATE   | :BF0000      |   629 |  5032 |   738   (3)| 00:00:01 |       |       |
|*  4 |     TABLE ACCESS FULL        | SALES_RETURN |   629 |  5032 |   738   (3)| 00:00:01 |       |       |
|   5 |    PARTITION RANGE SINGLE    |              |   786 |  6288 |    42   (5)| 00:00:01 |KEY(AP)|KEY(AP)|
|*  6 |     TABLE ACCESS FULL        | SALES        |   786 |  6288 |    42   (5)| 00:00:01 |KEY(AP)|KEY(AP)|
-----------------------------------------------------------------------------------------------------------

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

   2 - access("S"."TIME_ID"="SR"."TIME_ID")
   4 - filter("SR"."TIME_ID"='31-DEC-01')
   6 - filter("S"."TIME_ID"='31-DEC-01')
```

KEY means dynamic pruning at execution time AP means And Pruning, caused by bloom filter

Now get transitive predicate without data type conversion hence pruning

Agenda

1. Use the right Tools
2. Functions Friends or Foe
3. Data Type Dilemmas
4. Influencing an execution without adding hints

Different ways to influence the Optimizer

Statistics	Use the auto job or DBMS_STATS package Think about using extended statistics and histograms Don’t forget to include all constraints too
Stored Outline	Provides plan stability by freezing an execution plan No way to evolve a plan over time Currently deprecated
SQL Plan Management	Provides plan stability but can delay adopt of new plans as they need to be verification. Can't be shared across stmts.
SQL Profile	Requires Diagnostics Pack and may go stale. Can be shared by multiple SQL stmts with force matching
SQL Patch	A SQL manageability object that can be generated by the SQL Repair Advisor, in order to circumvent a plan which causes a failure
Hints	Only use as a last resort and only as a complete set of hints. Remember if you can hint it you can baseline or patch it!

If you can hint it, baseline it

Alternative approach to hints

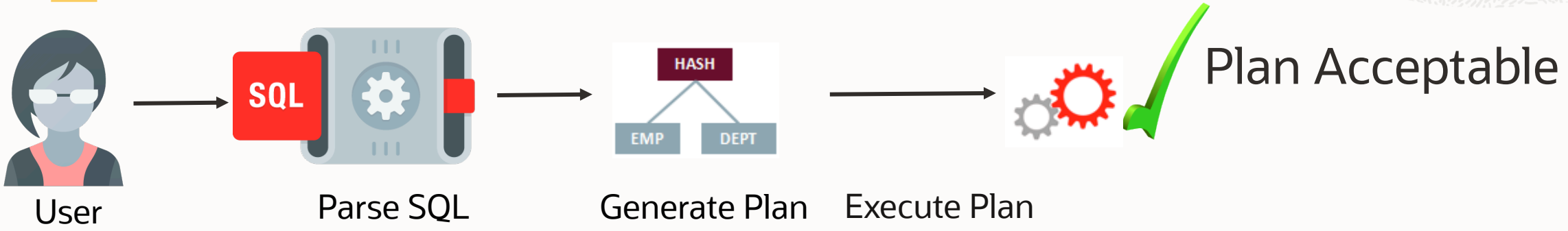
- It is not always possible to add hints to third party applications
- Hints can be extremely difficult to manage over time
- Once added never removed

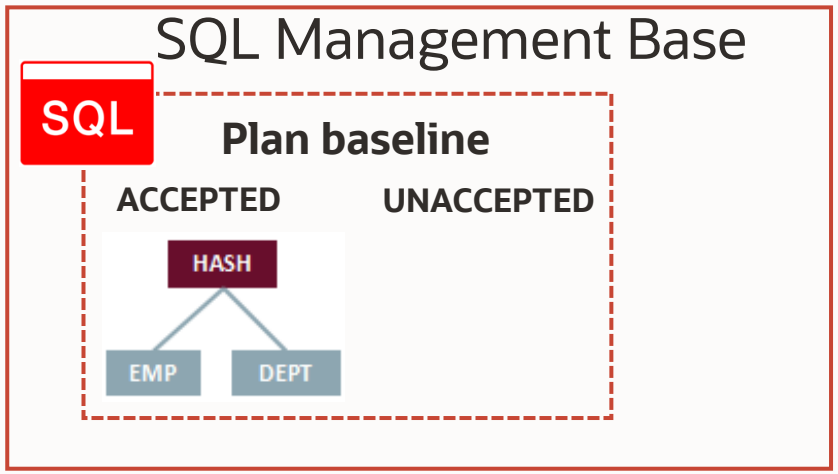
Solution

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

If you can hint it, baseline it

SQL Plan Management

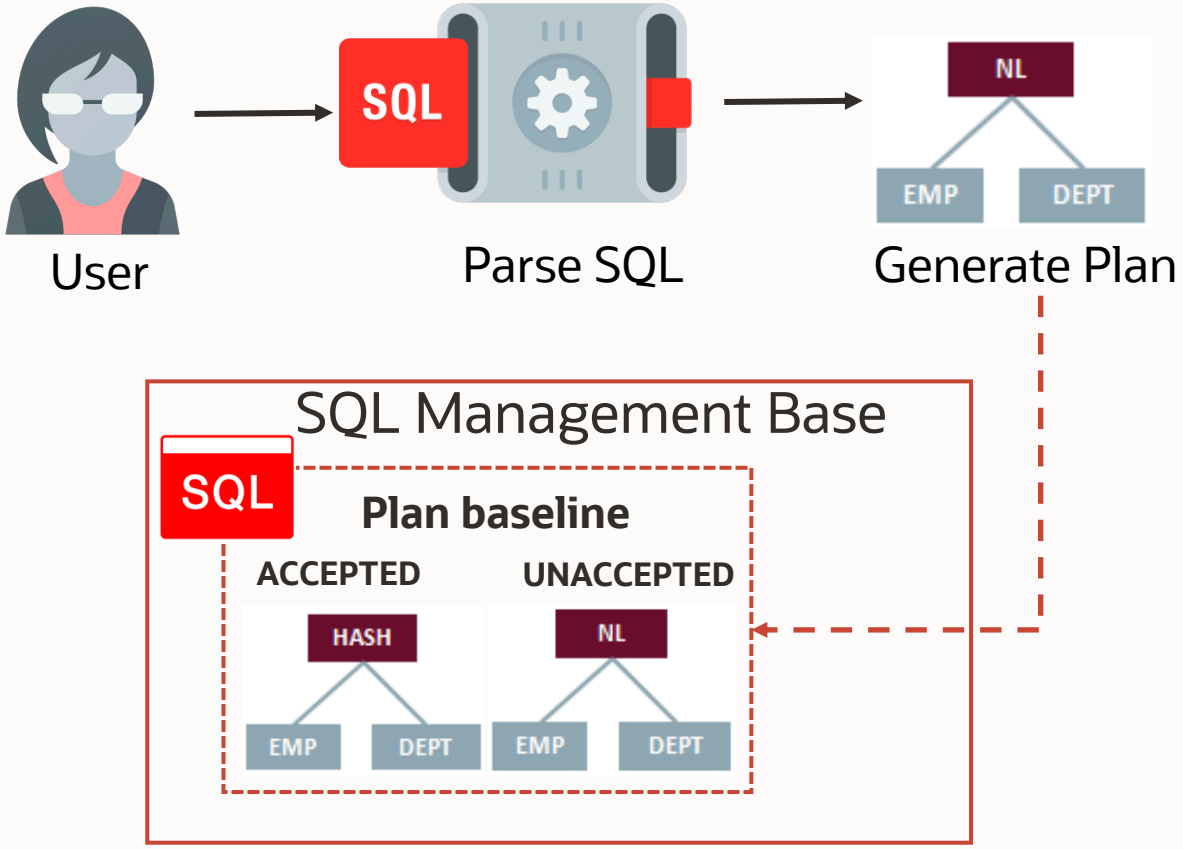




NOTE:: Actual execution plans stored in SQL plan baseline in Oracle Database 12c

If you can hint it, baseline it

SQL Plan Management

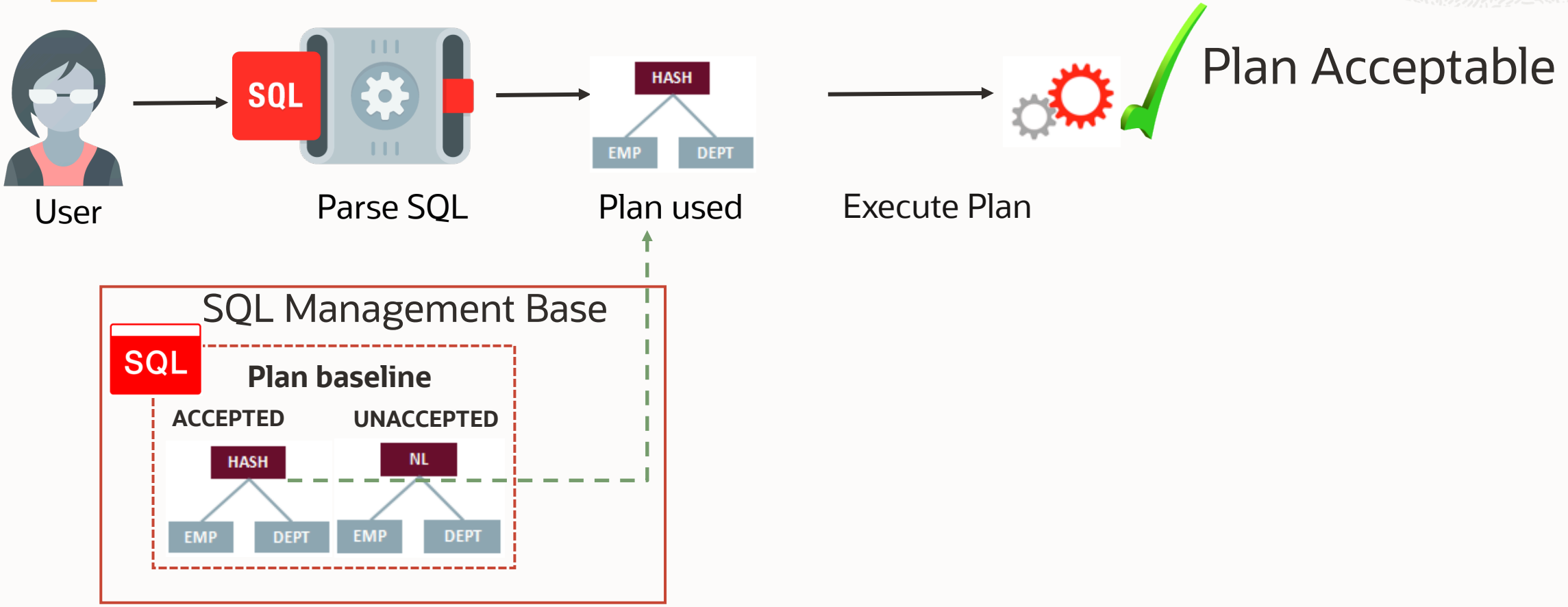


NOTE:: You **do not** need to be in auto-capture mode to have a new plan added to an existing SQL plan baseline

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

If you can hint it, baseline it

SQL Plan Management

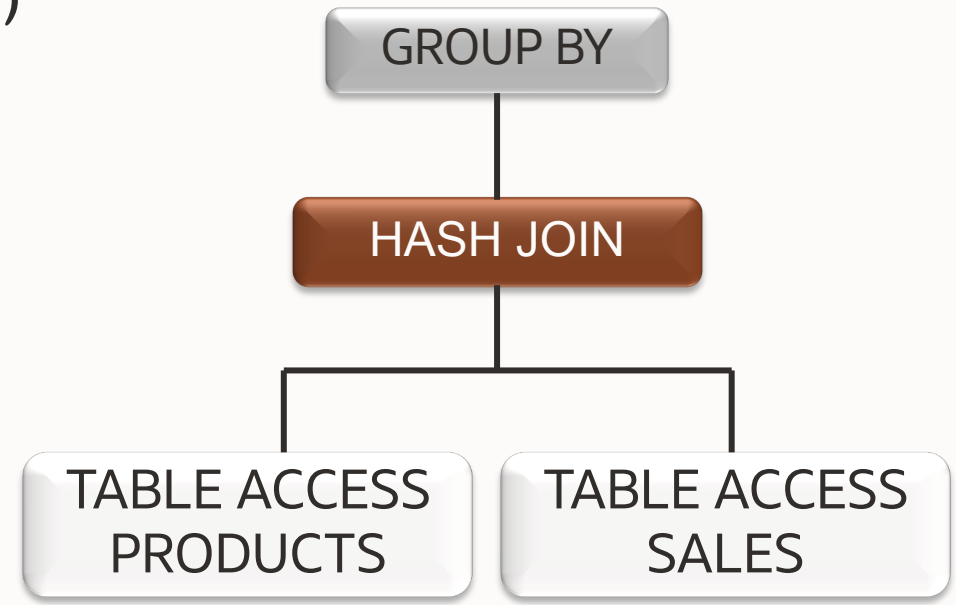


Influence execution plan without adding hints

Example Overview

Simple two table join between the SALES and PRODUCTS tables

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



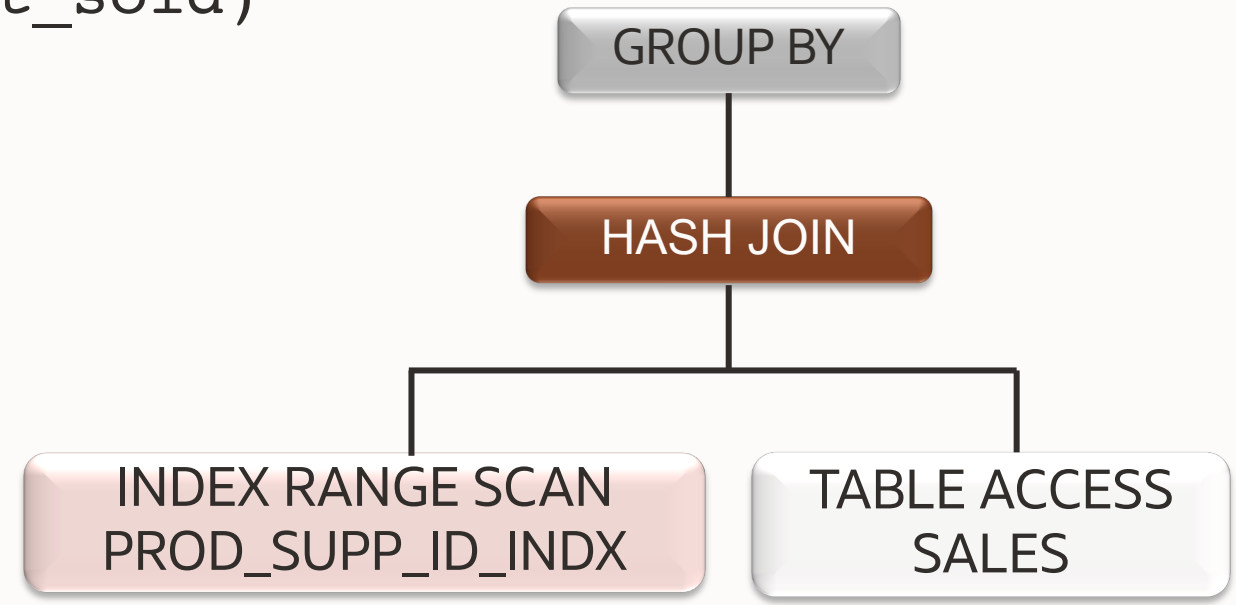
Current Plan

Influence execution plan without adding hints

Example Overview

Simple two table join between the SALES and PRODUCTS tables

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



Desired Plan

Influence execution plan without adding hints

Step 1. Execute the non-hinted SQL statement

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

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

Influence execution plan without adding hints

Default plan uses full table scans followed by a hash join

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

Plan hash value: 3535171836

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

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

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

Influence execution plan without adding hints

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

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

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

Influence execution plan without adding hints

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

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

PL/SQL PROCEDURE successfully completed.
```

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

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

Influence execution plan without adding hints

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

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

PL/SQL PROCEDURE successfully completed.

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

Influence execution plan without adding hints

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

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

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

Influence execution plan without adding hints

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

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

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

Influence execution plan without adding hints

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

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

PL/SQL PROCEDURE successfully completed.
```

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

Influence execution plan without adding hints

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

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

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

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

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

Influence execution plan without adding hints

Step 9. Confirm hinted plan is being used

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

Plan hash value: 2567686925

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

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

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

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

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

Note section also confirms SQL plan baseline used for statement

How to Create a SQL Patch to influence execution plans

Setup

```
CREATE table t (n number NOT NULL)
AS
SELECT object_id
FROM all_objects;

Table created.

CREATE INDEX ind_t_n ON t(n);

Index created.
```

How to Create a SQL Patch to influence execution plans

Step 1 What is the current plan

```
EXPLAIN PLAN FOR
SELECT *
FROM t
WHERE n > 0;

Explained.

SELECT * FROM table(dbms_xplan.display());
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       | 69478 |   882K|     42  (29)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL|  T    | 69478 |   882K|     42  (29)| 00:00:01 |
-------------------------------------------------------------------------------
```

But what if we needed the plan to use the index?

How to Create a SQL Patch to influence execution plans

Step 2 Create a SQL Patch with the hint to force an index plan

```
DECLARE
  patch_name varchar2(100);
BEGIN
 patch_name := sys.dbms_sqldiag.create_sql_patch(-
               sql_text=>'select * from t where n > 0', -
               hint_text=>'INDEX(@"SEL$1" "T")', -
               name=>'TEST_PATCH');
END;
/

PL/SQL procedure successfully completed.
```

How to Create a SQL Patch to influence execution plans

Step 3 Check patch took affect

```
EXPLAIN PLAN FOR
SELECT *
FROM t
WHERE n > 0;

Explained.

SELECT * FROM table(dbms_xplan.display());
------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          | 70187 |   891K|   166   (8)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN| IND_T_N1 | 70187 |   891K|   166   (8)| 00:00:01 |
------------------------------------------------------------------------------
Note
-----
 - SQL patch "TEST_PATCH" used for this statement
```

SQLMaria®

Join the Conversation

https://twitter.com/SQLMaria

https://blogs.oracle.com/optimizer/

https://sqlmaria.com

https://www.facebook.com/SQLMaria

ORACLE