Optimized Analytical Processing New Features with 11g R2

Hüsnü Şensoy

Global Maksimum Data & Information Technologies

Founder, VLDB Expert

husnu.sensoy@globalmaksimum.com

ORACLE
ACE Director

Agenda

- Introduction
  - Understanding Optimized Analytical Processing Capabilities
- New Capabilities by 11g Release 2
  - Multi-Predicate Partition Pruning
  - Intelligent Multi-Branch Execution
  - NULL Aware ANTI JOIN
  - Hash-Based DISTINCT Aggregation
- Conclusion

Who am I ?

- Data & Information expert on VLDB environments
  - DWH
  - Data Mining
  - Inference Systems
  - Data Archiving Solutions
  - Niche Storage Technologies
  - Recovery Strategies & Solutions
  - HA Systems
- Oracle ACED on BI field
  - Only one in Turkey
  - Still the youngest one all over the community.
- DBA of the Year 2009
  - 7th and still the youngest all over the world.
  - Only one in Turkey
- Blogger @ http://husnusensoy.wordpress.com
- Member of Oracle CAB for 12g DWH development
- Worldwide presenter of Oracle conferences and user group events

Optimized Analytical Processing New Features with 11g R2

Introduction

Optimized Analytical Processing Capabilities

- *Optimized Analytical Processing Capabilities* are those features implemented by Oracle on CBO, SQL execution, and expression manager that transparently improve SQL performance for your data crunching processes.
- The keyword is transparency. In many circumstances, you don’t need to change any configuration to enable those capabilities.
- Oracle keeps saying *«SQL is X times faster in this release»* mainly due to those features.
- It is usually very hard to hear about them until the product is mature or some of them cause problems in your production.

Why should I care about them ?

- In 10g one of the most important headaches for large DWH customers was related to new hash group by optimizations. Many customers have disabled them with the guidance of support (`_gby_hash_aggregation_enabled`). So being familiar with new SQL engine will let you a better understanding of product and give you the chance to take remedial actions.
- Most people are annoyed with SQL plan changes with each release. They usually choose to freeze them using various techniques. Understanding those new features will let you to understand the reasons behind some plan changes in new release.
- Just to appreciate the effort made by those developers optimizing our lives.

Optimized Analytical Processing New Features with 11g R2

Multi-Predicate Partition Pruning

husnu.sensoy@globalmaksimum.com

Partition Pruning

- In one of recent surveys, Oracle partitioning seems to be the *Top 1* feature used by large DWH sites.
- Range partitioning not only helps ILM in DWH but also improves query performance by partition pruning most of the time.
- Until 11gR2 Oracle is *biased* on using *static partition* pruning rather than *dynamic* one.
- *Multi-predicate pruning* is the idea of utilizing each and every possible pruning opportunity to reduce the amount of data to be read from disk or buffer cache.

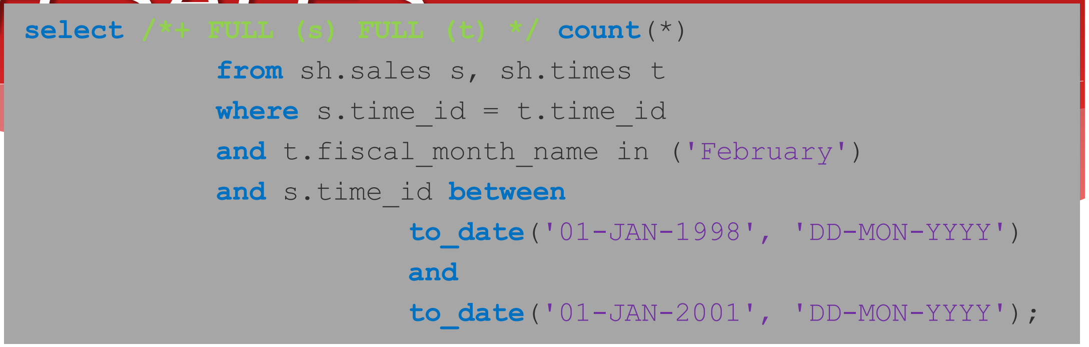
Partitioning Scheme for *SH.SALES*

SH.SALES	1995-1996	One partition per year
	1997	One partition for each half of a year
	1998-2003	One partition for each quarter of a year

A Simple Query on *SH.SALES*

```
select /*+ FULL (s) FULL (t) */ count(*)
      from sh.sales s, sh.times t
      where s.time_id = t.time_id
      and t.fiscal_month_name in ('February')
      and s.time_id between
            to_date('01-JAN-1998', 'DD-MON-YYYY')
            and
            to_date('01-JAN-2001', 'DD-MON-YYYY');
```

Partition Pruning Opportunities on SH.SALES



```
select /*+ FULL (s) FULL (t) */ count(*)
           from sh.sales s, sh.times t
           where s.time_id = t.time_id
           and t.fiscal_month_name in ('February')
           and s.time_id between
                      to_date('01-JAN-1998', 'DD-MON-YYYY')
                      and
                      to_date('01-JAN-2001', 'DD-MON-YYYY');
```

Pruning Idea	Description	Partitions to be scanned on SH.SALES
No Pruning		Scan all 28 partitions.
Static Pruning	Use predicate on *time_id* column of *SH.SALES*	Scan only 13 partitions.
Dynamic Pruning	Build a filter list for month *February* on *SH.TIMES* table then access to *SH.SALES* table.	Scan only 5 partitions
Static + Dynamic Pruning (Multi-predicate Pruning)	Use static & dynamic pruning together.	Scan only 3 partitions

In 10.2.0.4

```
Execution Plan

Plan hash value: 68236240

Id  Operation                     Name   Rows   Bytes  Cost (%CPU)  Time      Pstart  Pstop
 0  SELECT STATEMENT                     1      24       217  (11)  00:00:03
 1     SORT AGGREGATE                    1      24
*2        HASH JOIN                      41164  964K     217  (11)  00:00:03
*3           TABLE ACCESS FULL    TIMES  84     1344       9   (0)  00:00:01
 4           PARTITION RANGE ITERATOR    684K   5344K    202   (9)  00:00:03  5       17
*5              TABLE ACCESS FULL SALES  684K   5344K    202   (9)  00:00:03  5       17
```

A Simple Query on *SH.SALES* with Tracing Add-ons

```
set autot trace exp stat
alter sesssion set traacefile_identifier = ‘multiPredicatePruning’;
alter session set events ‘10128 trace name context forever, level 2’;

select /*+ FULL (s) FULL (t) */ count(*)
         from sh.sales s, sh.times t
         where s.time_id = t.time_id
         and t.fiscal_month_name in ('February')
         and s.time_id between
                 to_date('01-JAN-1998', 'DD-MON-YYYY')
                 and
                 to_date('01-JAN-2001', 'DD-MON-YYYY');

alter session set sql_trace = false;
```

Execution Plan in 11.2.0.1

```
Execution Plan

Plan hash value: 3278936322

Id   Operation                    Name      Rows    Bytes   Cost (%CPU)    Time        Pstart    Pstop

 0   SELECT STATEMENT                       1       24         322     (8)  00:00:05

 1     SORT AGGREGATE                       1       24

*2       HASH JOIN                          43252   1013K      322     (8)  00:00:05

 3         PART JOIN FILTER CREATE  :BF0000 91      1456        13     (0)  00:00:01

*4           TABLE ACCESS FULL      TIMES   91      1456        13     (0)  00:00:01

 5         PARTITION RANGE AND              690K    5393K      303     (7)  00:00:05  KEY(AP)   KEY(AP)

*6           TABLE ACCESS FULL      SALES   690K    5393K      303     (7)  00:00:05  KEY(AP)   KEY(AP)
```

10128 Trace Content in 11.2.0.1

```
...
Partition Iterator Information:
partition level = PARTITION
call time = RUN
order = ASCENDING
Partition iterator for level 1:
iterator = ARRAY [count= 3, max = 28] = 4 8 12
...
```

Remarks

- Partitioning is and will be *Number One* trick of very large data management and processing.
- *Multi-predicate Partition Pruning* boosts Oracle's pruning opportunities for cases where several predicates can result in pruning.

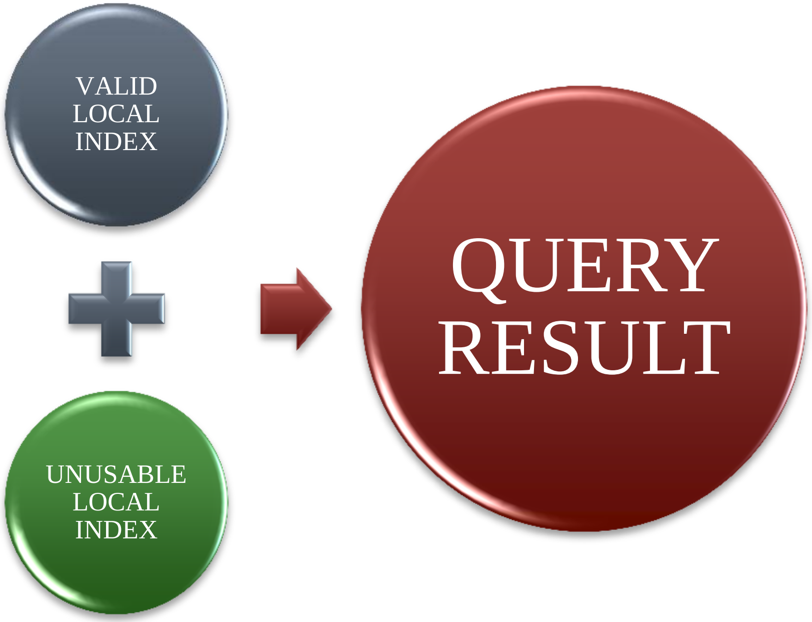
Optimized Analytical Processing New Features with 11g R2

Intelligent Multi-Branch Execution

husnu.sensoy@globalmaksimum.com

Horizontal Partial Indexing

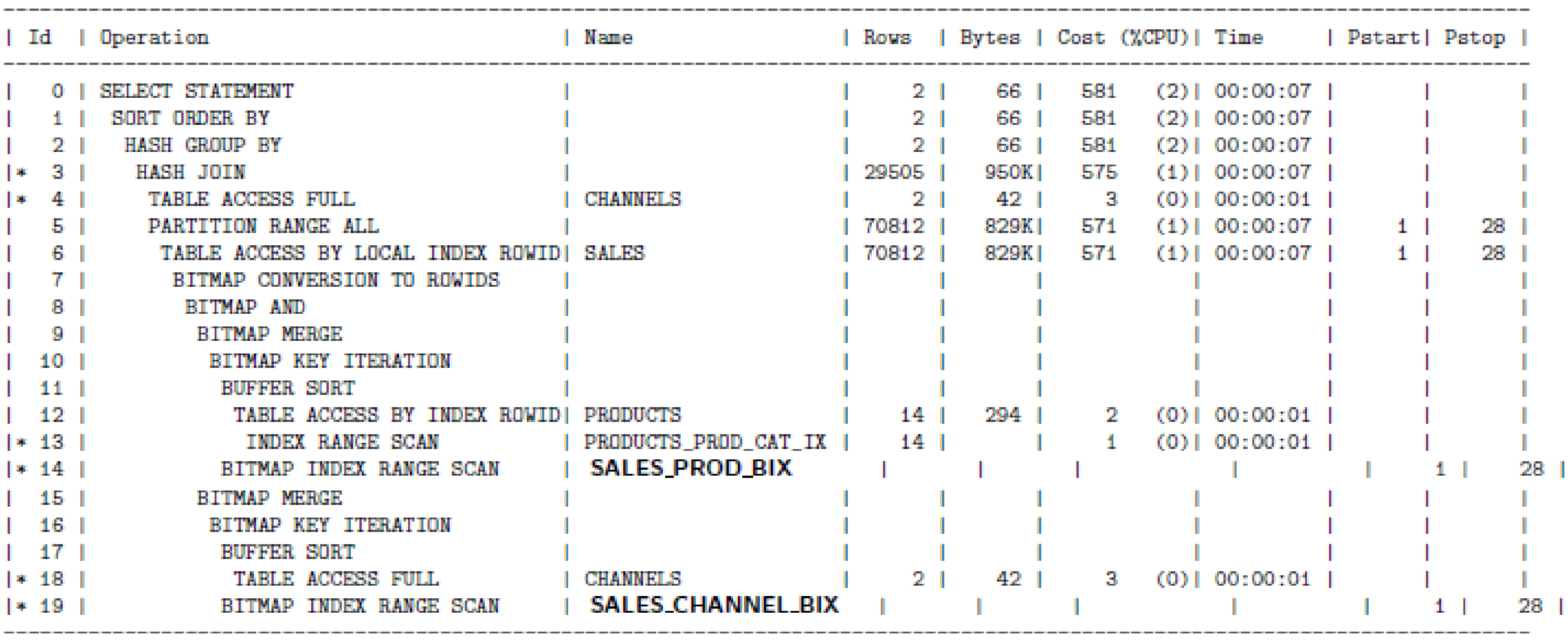
- As you may all know, Oracle allows *UNUSABLE* index partitions starting from early releases of partitioning technology.
- Many data warehouses wish to disable some old index partitions to reveal the burden of maintaining them during ELT.
- Intelligent Multi-Branch Execution is a query rewrite technique to split a single SQL statement based on a partitioned table having LOCAL indices.



Another Simple Query on *SH.SALES*

```
select channels.channel_desc,
      sum(sales.amount_sold) as total_amount
      from sh.sales, sh.products, sh.channels
      where channels.channel_id = sales.channel_id
      and products.prod_id = sales.prod_id
      and channels.channel_class = 'Direct'
      and products.prod_categorY = 'Photo'
      group by channels.channel_desc
      order by 2 desc;
```

Execution Plan

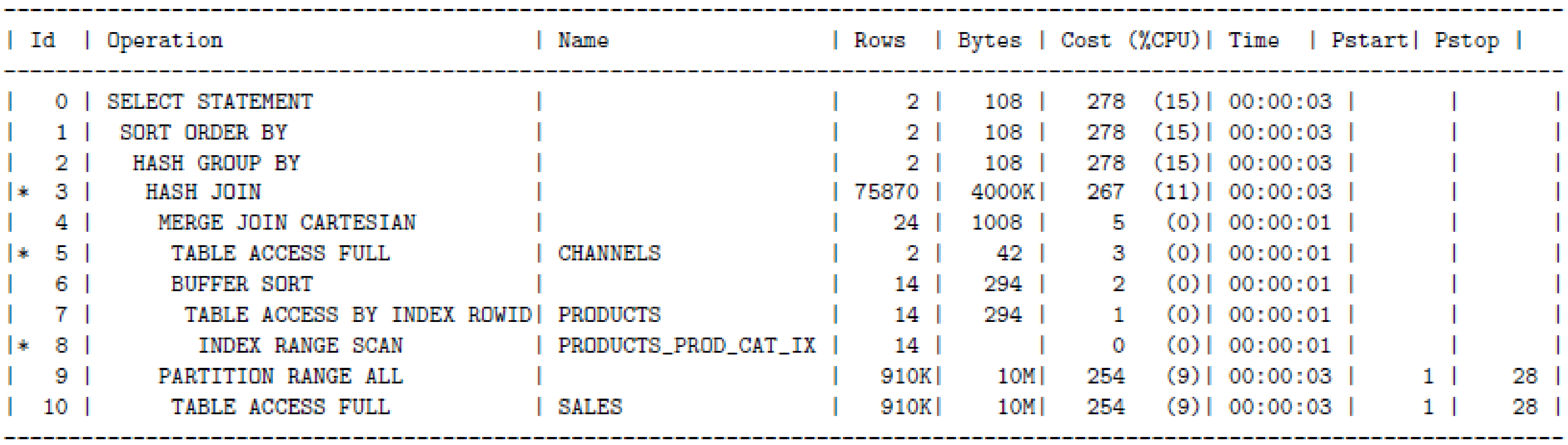


```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name                 | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |                      |     2 |    66 |   581   (2)| 00:00:07 |       |       |
|   1 |  SORT ORDER BY                        |                      |     2 |    66 |   581   (2)| 00:00:07 |       |       |
|   2 |   HASH GROUP BY                       |                      |     2 |    66 |   581   (2)| 00:00:07 |       |       |
|*  3 |    HASH JOIN                          |                      | 29505 |   950K|   575   (1)| 00:00:07 |       |       |
|*  4 |     TABLE ACCESS FULL                 | CHANNELS             |     2 |    42 |     3   (0)| 00:00:01 |       |       |
|   5 |     PARTITION RANGE ALL               |                      | 70812 |   829K|   571   (1)| 00:00:07 |     1 |    28 |
|   6 |      TABLE ACCESS BY LOCAL INDEX ROWID| SALES                | 70812 |   829K|   571   (1)| 00:00:07 |     1 |    28 |
|   7 |       BITMAP CONVERSION TO ROWIDS     |                      |       |       |            |          |       |       |
|   8 |        BITMAP AND                     |                      |       |       |            |          |       |       |
|   9 |         BITMAP MERGE                  |                      |       |       |            |          |       |       |
|  10 |          BITMAP KEY ITERATION         |                      |       |       |            |          |       |       |
|  11 |           BUFFER SORT                 |                      |       |       |            |          |       |       |
|  12 |            TABLE ACCESS BY INDEX ROWID| PRODUCTS             |    14 |   294 |     2   (0)| 00:00:01 |       |       |
|* 13 |             INDEX RANGE SCAN          | PRODUCTS_PROD_CAT_IX |    14 |       |     1   (0)| 00:00:01 |       |       |
|* 14 |           BITMAP INDEX RANGE SCAN     | SALES_PROD_BIX       |       |       |            |          |     1 |    28 |
|  15 |         BITMAP MERGE                  |                      |       |       |            |          |       |       |
|  16 |          BITMAP KEY ITERATION         |                      |       |       |            |          |       |       |
|  17 |           BUFFER SORT                 |                      |       |       |            |          |       |       |
|* 18 |            TABLE ACCESS FULL          | CHANNELS             |     2 |    42 |     3   (0)| 00:00:01 |       |       |
|* 19 |           BITMAP INDEX RANGE SCAN     | SALES_CHANNEL_BIX    |       |       |            |          |     1 |    28 |
-------------------------------------------------------------------------------------------------------------------------------
```

Disable A Few LOCAL Index Partitions

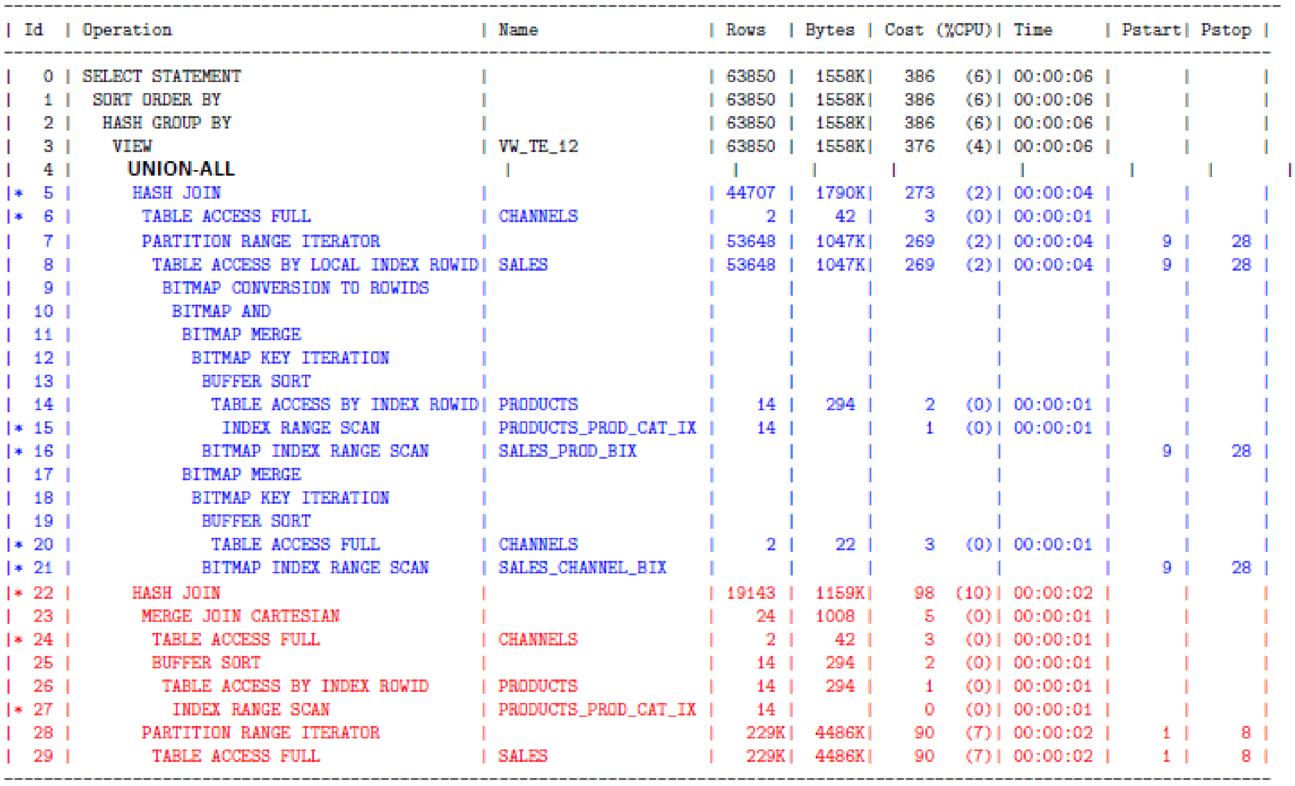
```
ALTER INDEX SH. SALES_CHANNEL_BIX MODIFY PARTITION SALES_1995 UNUSABLE;
ALTER INDEX SH. SALES_CHANNEL_BIX MODIFY PARTITION SALES_1996 UNUSABLE;
ALTER INDEX SH. SALES_CHANNEL_BIX MODIFY PARTITION SALES_H1_1997 UNUSABLE;
ALTER INDEX SH. SALES_CHANNEL_BIX MODIFY PARTITION SALES_H2_1997 UNUSABLE;
ALTER INDEX SH. SALES_CHANNEL_BIX MODIFY PARTITION SALES_Q1_1998 UNUSABLE;
ALTER INDEX SH. SALES_CHANNEL_BIX MODIFY PARTITION SALES_Q2_1998 UNUSABLE;
ALTER INDEX SH. SALES_CHANNEL_BIX MODIFY PARTITION SALES_Q3_1998 UNUSABLE;
ALTER INDEX SH. SALES_CHANNEL_BIX MODIFY PARTITION SALES_Q4_1998 UNUSABLE;
```

Execution Plan in 10.2.0.4



```
-------------------------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                 | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                      |     2 |   108 |   278  (15)| 00:00:03 |       |       |
|   1 |  SORT ORDER BY                 |                      |     2 |   108 |   278  (15)| 00:00:03 |       |       |
|   2 |   HASH GROUP BY                |                      |     2 |   108 |   278  (15)| 00:00:03 |       |       |
|*  3 |    HASH JOIN                   |                      | 75870 |  4000K|   267  (11)| 00:00:03 |       |       |
|   4 |     MERGE JOIN CARTESIAN       |                      |    24 |  1008 |     5   (0)| 00:00:01 |       |       |
|*  5 |      TABLE ACCESS FULL         | CHANNELS             |     2 |    42 |     3   (0)| 00:00:01 |       |       |
|   6 |      BUFFER SORT               |                      |    14 |   294 |     2   (0)| 00:00:01 |       |       |
|   7 |       TABLE ACCESS BY INDEX ROWID| PRODUCTS             |    14 |   294 |     1   (0)| 00:00:01 |       |       |
|*  8 |        INDEX RANGE SCAN        | PRODUCTS_PROD_CAT_IX |    14 |       |     0   (0)| 00:00:01 |       |       |
|   9 |     PARTITION RANGE ALL        |                      |   910K|    10M|   254   (9)| 00:00:03 |     1 |    28 |
|  10 |      TABLE ACCESS FULL         | SALES                |   910K|    10M|   254   (9)| 00:00:03 |     1 |    28 |
-------------------------------------------------------------------------------------------------------------------
```

Execution Plan in 11.2.0.1



```
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                 | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                      | 63850 |  1558K|   386   (6)| 00:00:06 |       |       |
|   1 |  SORT ORDER BY                           |                      | 63850 |  1558K|   386   (6)| 00:00:06 |       |       |
|   2 |   HASH GROUP BY                          |                      | 63850 |  1558K|   386   (6)| 00:00:06 |       |       |
|   3 |    VIEW                                  | VW_TE_12             | 63850 |  1558K|   376   (4)| 00:00:06 |       |       |
|   4 |     UNION-ALL                              |                      |       |       |            |          |       |       |
|*  5 |      HASH JOIN                           |                      | 44707 |  1790K|   273   (2)| 00:00:04 |       |       |
|*  6 |       TABLE ACCESS FULL                  | CHANNELS             |     2 |    42 |     3   (0)| 00:00:01 |       |       |
|   7 |       PARTITION RANGE ITERATOR           |                      | 53648 |  1047K|   269   (2)| 00:00:04 |     9 |    28 |
|   8 |        TABLE ACCESS BY LOCAL INDEX ROWID| SALES                | 53648 |  1047K|   269   (2)| 00:00:04 |     9 |    28 |
|   9 |         BITMAP CONVERSION TO ROWIDS      |                      |       |       |            |          |       |       |
|  10 |          BITMAP AND                      |                      |       |       |            |          |       |       |
|  11 |           BITMAP MERGE                   |                      |       |       |            |          |       |       |
|  12 |            BITMAP KEY ITERATION          |                      |       |       |            |          |       |       |
|  13 |             BUFFER SORT                  |                      |       |       |            |          |       |       |
|  14 |              TABLE ACCESS BY INDEX ROWID| PRODUCTS             |    14 |   294 |     2   (0)| 00:00:01 |       |       |
|* 15 |               INDEX RANGE SCAN           | PRODUCTS_PROD_CAT_IX |    14 |       |     1   (0)| 00:00:01 |       |       |
|* 16 |             BITMAP INDEX RANGE SCAN      | SALES_PROD_BIX       |       |       |            |          |     9 |    28 |
|  17 |           BITMAP MERGE                   |                      |       |       |            |          |       |       |
|  18 |            BITMAP KEY ITERATION          |                      |       |       |            |          |       |       |
|  19 |             BUFFER SORT                  |                      |       |       |            |          |       |       |
|* 20 |              TABLE ACCESS FULL           | CHANNELS             |     2 |    22 |     3   (0)| 00:00:01 |       |       |
|* 21 |             BITMAP INDEX RANGE SCAN      | SALES_CHANNEL_BIX    |       |       |            |          |     9 |    28 |
|* 22 |      HASH JOIN                           |                      | 19143 |  1159K|    98  (10)| 00:00:02 |       |       |
|  23 |       MERGE JOIN CARTESIAN               |                      |    24 |  1008 |     5   (0)| 00:00:01 |       |       |
|* 24 |        TABLE ACCESS FULL                 | CHANNELS             |     2 |    42 |     3   (0)| 00:00:01 |       |       |
|  25 |        BUFFER SORT                       |                      |    14 |   294 |     2   (0)| 00:00:01 |       |       |
|  26 |         TABLE ACCESS BY INDEX ROWID      | PRODUCTS             |    14 |   294 |     1   (0)| 00:00:01 |       |       |
|* 27 |          INDEX RANGE SCAN                | PRODUCTS_PROD_CAT_IX |    14 |       |     0   (0)| 00:00:01 |       |       |
|  28 |       PARTITION RANGE ITERATOR           |                      |   229K|  4486K|    90   (7)| 00:00:02 |     1 |     8 |
|  29 |        TABLE ACCESS FULL                 | SALES                |   229K|  4486K|    90   (7)| 00:00:02 |     1 |     8 |
-------------------------------------------------------------------------------------------------------------------------------------
```

Remarks

- *Intelligent Multi-Branch Execution* is an invaluable new optimization for sites using LOCAL indexes.
- Keep in mind in order to use this optimization `SKIP_UNUSABLE_INDEXES` parameter should set to be `TRUE`.
- This option can be disabled by setting `_OPTIMIZER_TABLE_EXPANSION` parameter to `FALSE`.

Optimized Analytical Processing New Features with 11g R2

NULL Aware ANTI-JOIN

husnu.sensoy@globalmaksimum.com

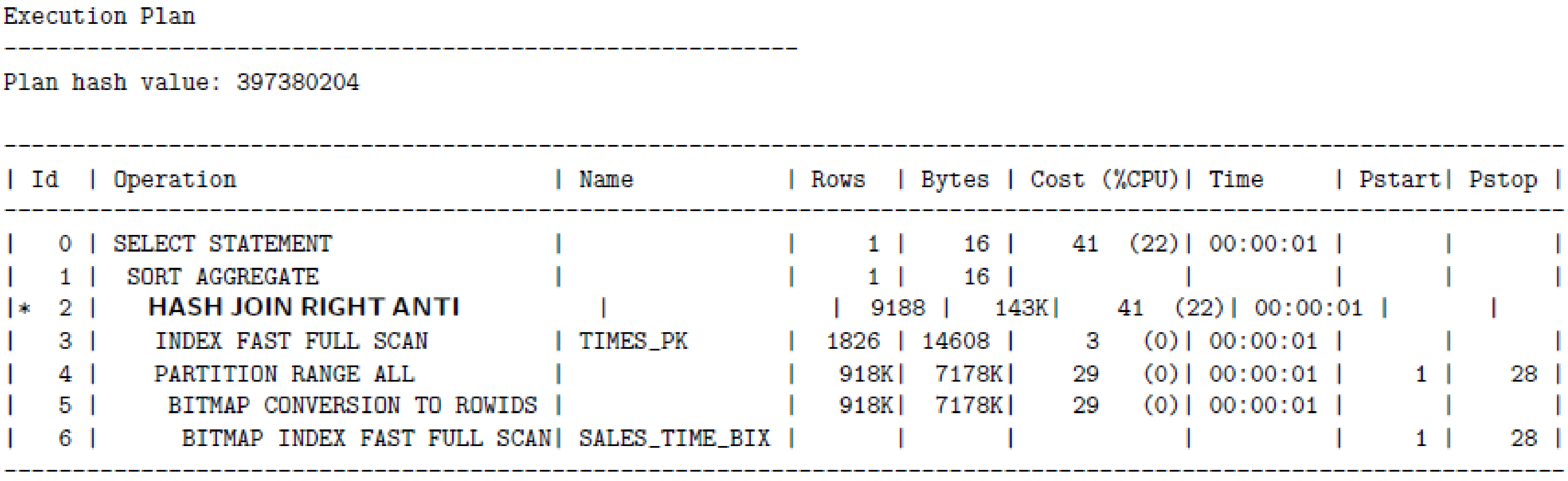
ANTI JOIN

- Oracle can use ANTI JOIN execution plan (with Nested Loop, Hash, or Merge join options) in case that a SQL statement contains NOT IN or NOT EXISTS clauses (or something rewritten to this).
- Hash Anti-Join is known to be an optimal execution plan for many large data warehouse queries containing above clauses.
- One major problem about classical anti-join is that due to some design errors like constraint ignorance, Oracle will reject using anti-join (not to generate erroneous result sets) and put a FILTER step instead (Refer one of my earlier blog posts).
- FILTER is usually CPU consuming and never-ending step for the join of large datasets.

Yet Another Simple Query on *SH.SALES*

```
select count(*) from sh.sales
                where time_id not in (select time_id
                                    from sh.times);
```

Anti Join

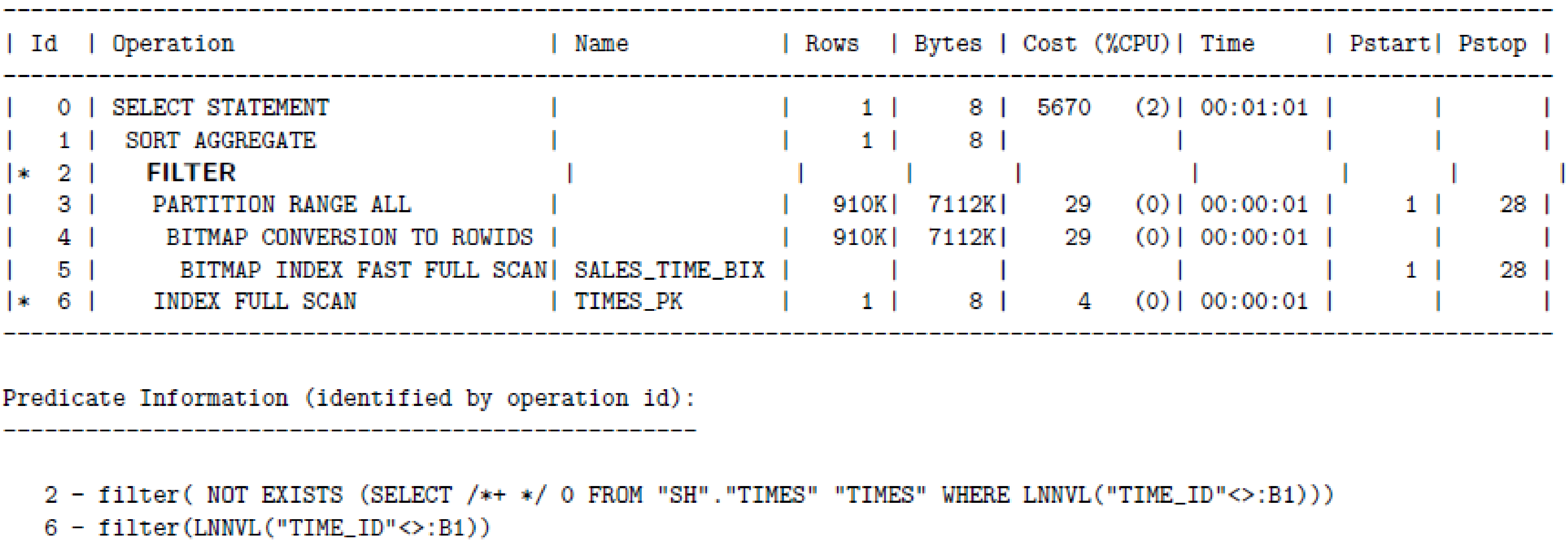


```
Execution Plan
----------------------------------------------------------
Plan hash value: 397380204

----------------------------------------------------------------------------------------------------------------
| Id  | Operation                        | Name             | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                 |                  |     1 |    16 |    41  (22)| 00:00:01 |       |       |
|   1 |  SORT AGGREGATE                  |                  |     1 |    16 |            |          |       |       |
|*  2 |   HASH JOIN RIGHT ANTI      |                  |  9188 |   143K|    41  (22)| 00:00:01 |       |
|   3 |    INDEX FAST FULL SCAN          | TIMES_PK         |  1826 | 14608 |     3   (0)| 00:00:01 |       |       |
|   4 |    PARTITION RANGE ALL           |                  |   918K|  7178K|    29   (0)| 00:00:01 |     1 |    28 |
|   5 |     BITMAP CONVERSION TO ROWIDS  |                  |   918K|  7178K|    29   (0)| 00:00:01 |       |       |
|   6 |      BITMAP INDEX FAST FULL SCAN| SALES_TIME_BIX   |       |       |            |          |     1 |    28 |
----------------------------------------------------------------------------------------------------------------
```

Release NOT NULL Constraint on *SH.SALES*

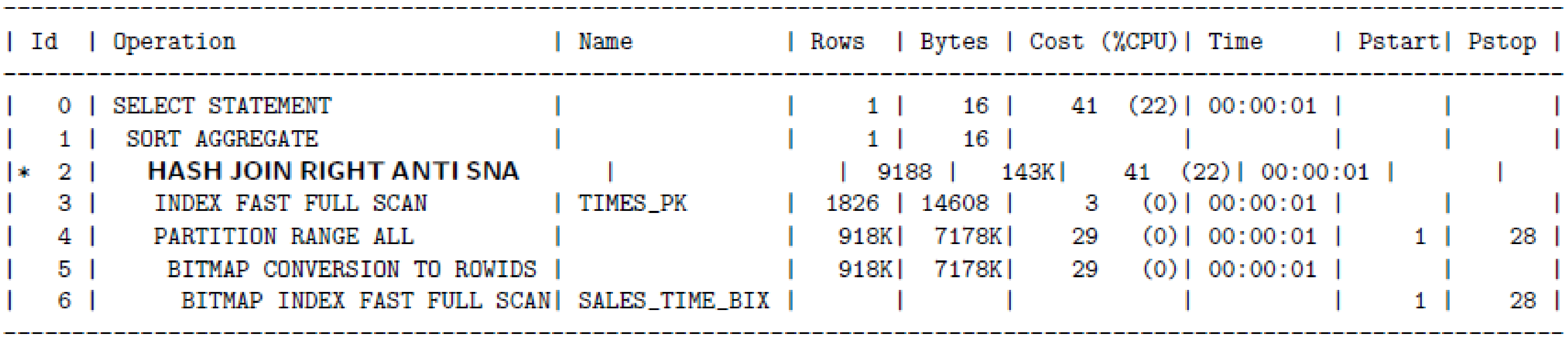
```
alter table SH.SALES modify TIME_ID NULL;
```

Execution Plan in 10.2.0.4



```
--------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name           | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
--------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |     1 |     8 |  5670   (2)| 00:01:01 |       |       |
|   1 |  SORT AGGREGATE              |                |     1 |     8 |            |          |       |       |
|*  2 |   FILTER                     |                |       |       |            |          |       |       |
|   3 |    PARTITION RANGE ALL       |                |  910K|  7112K|    29   (0)| 00:00:01 |     1 |    28 |
|   4 |     BITMAP CONVERSION TO ROWIDS |             |  910K|  7112K|    29   (0)| 00:00:01 |       |       |
|   5 |      BITMAP INDEX FAST FULL SCAN| SALES_TIME_BIX |       |       |            |          |     1 |    28 |
|*  6 |    INDEX FULL SCAN           | TIMES_PK       |     1 |     8 |     4   (0)| 00:00:01 |       |       |
--------------------------------------------------------------------------------------------------------

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

   2 - filter( NOT EXISTS (SELECT /*+ */ 0 FROM "SH"."TIMES" "TIMES" WHERE LNNVL("TIME_ID"<>:B1)))
   6 - filter(LNNVL("TIME_ID"<>:B1))
```

Execution Plan in 11.1.0.6+



```
-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                       | Name            | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                |                 |     1 |    16 |    41  (22)| 00:00:01 |       |       |
|   1 |  SORT AGGREGATE                 |                 |     1 |    16 |            |          |       |       |
|*  2 |   HASH JOIN RIGHT ANTI SNA    |                 |  9188 |   143K|    41  (22)| 00:00:01 |       |
|   3 |    INDEX FAST FULL SCAN         | TIMES_PK        |  1826 | 14608 |     3   (0)| 00:00:01 |       |       |
|   4 |    PARTITION RANGE ALL          |                 |   918K|  7178K|    29   (0)| 00:00:01 |     1 |    28 |
|   5 |     BITMAP CONVERSION TO ROWIDS |                 |   918K|  7178K|    29   (0)| 00:00:01 |       |       |
|   6 |      BITMAP INDEX FAST FULL SCAN| SALES_TIME_BIX  |       |       |            |          |     1 |    28 |
-----------------------------------------------------------------------------------------------------------------
```

Create *SH.SALES_NEW* & *SH.TIMES_NEW*

```
create table sh.sales_new nologging
tablespace users as
select     s.*,
           to_char(time_id,'DD') day_id,
           to_char(time_id,'MM') month_id,
           to_char(time_id,'YYYY') year_id
from sh.sales s;

create table sh.times_new
nologging tablespace users as
select     t.*,
           to_char(time_id,'DD') day_id,
           to_char(time_id,'MM') month_id,
           to_char(time_id,'YYYY') year_id
from sh.times t;
```

```
alter table sh.sales_new modify day_id not null;
alter table sh.sales_new modify month_id not null;
alter table sh.sales_new modify year_id not null;

alter table sh.times_new modify day_id not null;
alter table sh.times_new modify month_id not null;
alter table sh.times_new modify year_id not null;
```

How about this Query ?

```
select count(*)
from sh.sales_new
where (day_id,
       month_id,
       year_id) not in (select day_id,
                                month_id,
                                year_id
                          from sh.times_new);
```

Simple Execution Plan

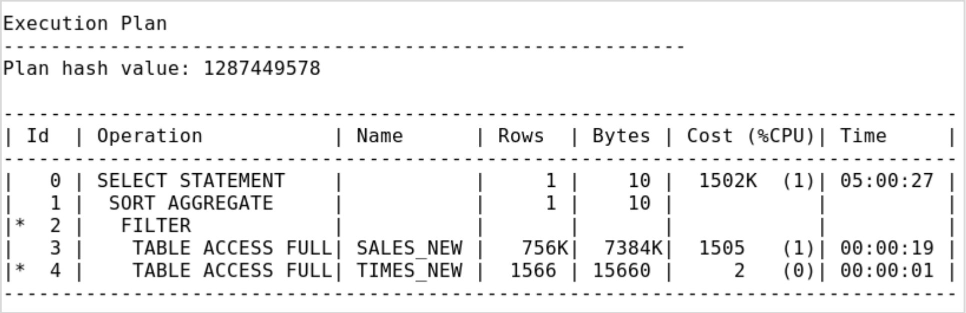
```
Execution Plan
----------------------------------------------------------
Plan hash value: 3458658284

-----------------------------------------------------------------------------------
| Id  | Operation             | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |           |     1 |    22 |  1527   (1)| 00:00:19 |
|   1 |  SORT AGGREGATE       |           |     1 |    22 |            |          |
|*  2 |   HASH JOIN RIGHT ANTI|           |  9188 |   197K|  1527   (1)| 00:00:19 |
|   3 |    TABLE ACCESS FULL  | TIMES_NEW |  1826 | 20086 |    17   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL  | SALES_NEW |   918K|  9870K|  1507   (1)| 00:00:19 |
-----------------------------------------------------------------------------------
```

Remove Only One NULL Constraint on *SH.TIMES*

```
alter table sh.times_new modify year_id null;
```

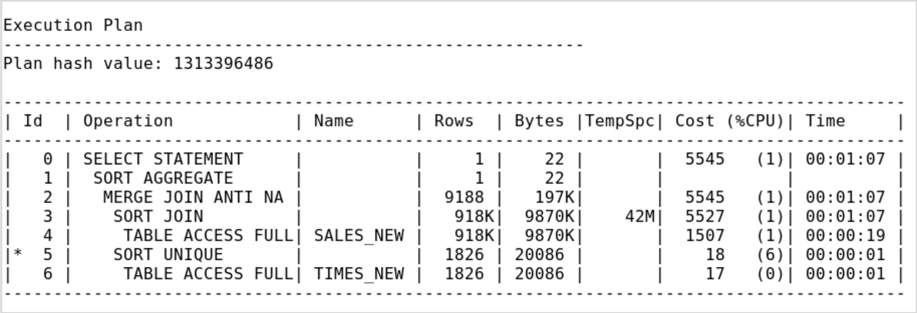
Execution Plan (NA)

Pre 11g Release 2



```
Execution Plan
----------------------------------------------------------
Plan hash value: 1287449578

--------------------------------------------------------------------------------
| Id  | Operation           | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |     1 |    10 |  1502K  (1)| 05:00:27 |
|   1 |  SORT AGGREGATE     |           |     1 |    10 |            |          |
|*  2 |   FILTER            |           |       |       |            |          |
|   3 |    TABLE ACCESS FULL| SALES_NEW |   756K|  7384K|  1505   (1)| 00:00:19 |
|*  4 |    TABLE ACCESS FULL| TIMES_NEW |  1566 | 15660 |     2   (0)| 00:00:01 |
--------------------------------------------------------------------------------
```

By 11g Release 2



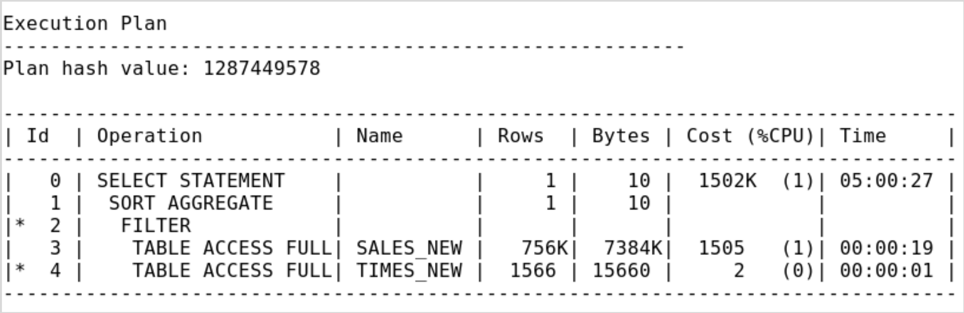
```
Execution Plan
----------------------------------------------------------
Plan hash value: 1313396486

----------------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |     1 |    22 |       |  5545   (1)| 00:01:07 |
|   1 |  SORT AGGREGATE      |           |     1 |    22 |       |            |          |
|   2 |   MERGE JOIN ANTI NA |           |  9188 |   197K|       |  5545   (1)| 00:01:07 |
|   3 |    SORT JOIN         |           |   918K|  9870K|    42M|  5527   (1)| 00:01:07 |
|   4 |     TABLE ACCESS FULL| SALES_NEW |   918K|  9870K|       |  1507   (1)| 00:00:19 |
|*  5 |    SORT UNIQUE       |           |  1826 | 20086 |       |    18   (6)| 00:00:01 |
|   6 |     TABLE ACCESS FULL| TIMES_NEW |  1826 | 20086 |       |    17   (0)| 00:00:01 |
----------------------------------------------------------------------------------------
```

Remove Only One NULL Constraint on *SH.SALES*

```
alter table sh.times_new modify year_id not null;
alter table sh.sales_new modify year_id null;
```

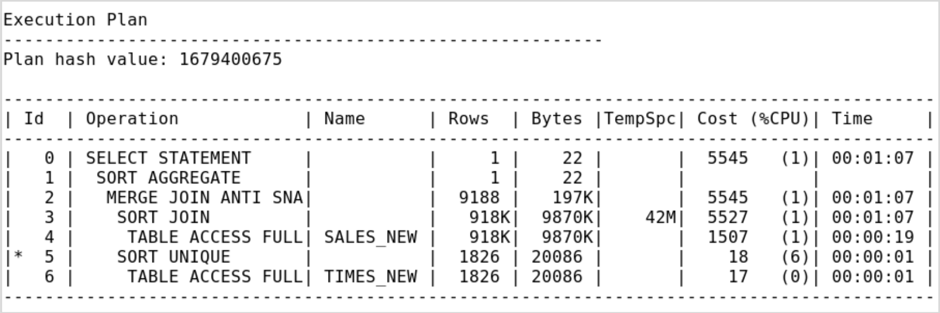
Execution Plan (SNA)

Pre 11g Release 2



```
Execution Plan
----------------------------------------------------------
Plan hash value: 1287449578

---------------------------------------------------------------------------------
| Id  | Operation           | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |            |     1 |    10 |  1502K  (1)| 05:00:27 |
|   1 |  SORT AGGREGATE     |            |     1 |    10 |            |          |
|*  2 |   FILTER            |            |       |       |            |          |
|   3 |    TABLE ACCESS FULL| SALES_NEW  |   756K|  7384K|  1505   (1)| 00:00:19 |
|*  4 |    TABLE ACCESS FULL| TIMES_NEW  |  1566 | 15660 |     2   (0)| 00:00:01 |
---------------------------------------------------------------------------------
```

By 11g Release 2



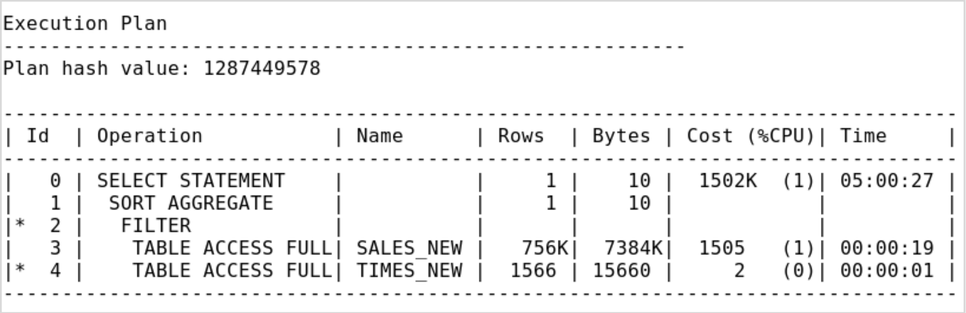
```
Execution Plan
----------------------------------------------------------
Plan hash value: 1679400675

-----------------------------------------------------------------------------------------
| Id  | Operation            | Name       | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |            |     1 |    22 |       |  5545   (1)| 00:01:07 |
|   1 |  SORT AGGREGATE      |            |     1 |    22 |       |            |          |
|   2 |   MERGE JOIN ANTI SNA|            |  9188 |   197K|       |  5545   (1)| 00:01:07 |
|   3 |    SORT JOIN         |            |   918K|  9870K|    42M|  5527   (1)| 00:01:07 |
|   4 |     TABLE ACCESS FULL| SALES_NEW  |   918K|  9870K|       |  1507   (1)| 00:00:19 |
|*  5 |    SORT UNIQUE       |            |  1826 | 20086 |       |    18   (6)| 00:00:01 |
|   6 |     TABLE ACCESS FULL| TIMES_NEW  |  1826 | 20086 |       |    17   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------
```

Remove Only One NULL Constraint on *SH.SALES & SH.TIMES*

```
alter table sh.times_new modify year_id null;
alter table sh.sales_new modify year_id null;
```

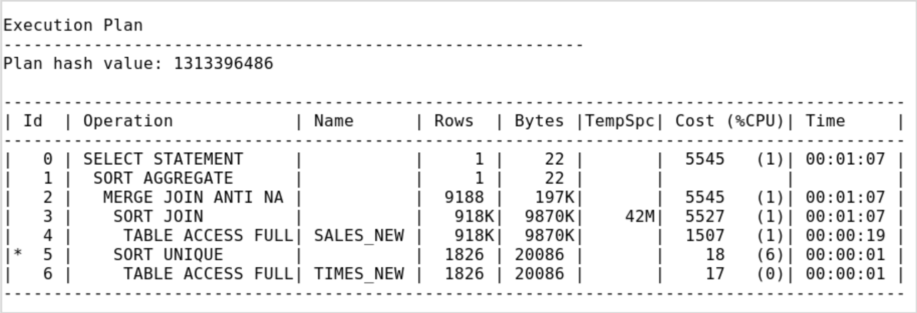
Execution Plan (NA)

Pre 11g Release 2



```
Execution Plan
----------------------------------------------------------
Plan hash value: 1287449578

---------------------------------------------------------------------------------
| Id  | Operation           | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |     1 |    10 |  1502K  (1)| 05:00:27 |
|   1 |  SORT AGGREGATE     |           |     1 |    10 |            |          |
|*  2 |   FILTER            |           |       |       |            |          |
|   3 |    TABLE ACCESS FULL| SALES_NEW |   756K|  7384K|  1505   (1)| 00:00:19 |
|*  4 |    TABLE ACCESS FULL| TIMES_NEW |  1566 | 15660 |     2   (0)| 00:00:01 |
---------------------------------------------------------------------------------
```

By 11g Release 2



```
Execution Plan
----------------------------------------------------------
Plan hash value: 1313396486

------------------------------------------------------------------------------------------
| Id  | Operation             | Name      | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |           |     1 |    22 |       |  5545   (1)| 00:01:07 |
|   1 |  SORT AGGREGATE       |           |     1 |    22 |       |            |          |
|   2 |   MERGE JOIN ANTI NA  |           |  9188 |   197K|       |  5545   (1)| 00:01:07 |
|   3 |    SORT JOIN          |           |   918K|  9870K|    42M|  5527   (1)| 00:01:07 |
|   4 |     TABLE ACCESS FULL| SALES_NEW |   918K|  9870K|       |  1507   (1)| 00:00:19 |
|*  5 |    SORT UNIQUE        |           |  1826 | 20086 |       |    18   (6)| 00:00:01 |
|   6 |     TABLE ACCESS FULL| TIMES_NEW |  1826 | 20086 |       |    17   (0)| 00:00:01 |
------------------------------------------------------------------------------------------
```

Remarks

- *NULL Aware ANTI JOIN* is a great enhancement for constraint ignorant databases.
- SNA is first introduced in 11g Release 1, but multi column support is now available by 11g Release 2
- SNA is not a way to cheat SQL design practices.
- This option can be disabled by setting `_optimizer_null_aware_antijoin` parameter to `FALSE`
- To learn more about *NULL Aware ANTI JOIN*, refer to great post by Greg Rahn.

Optimized Analytical Processing New Features with 11g R2

Hash-Based DISTINCT Aggregation

husnu.sensoy@globalmaksimum.com

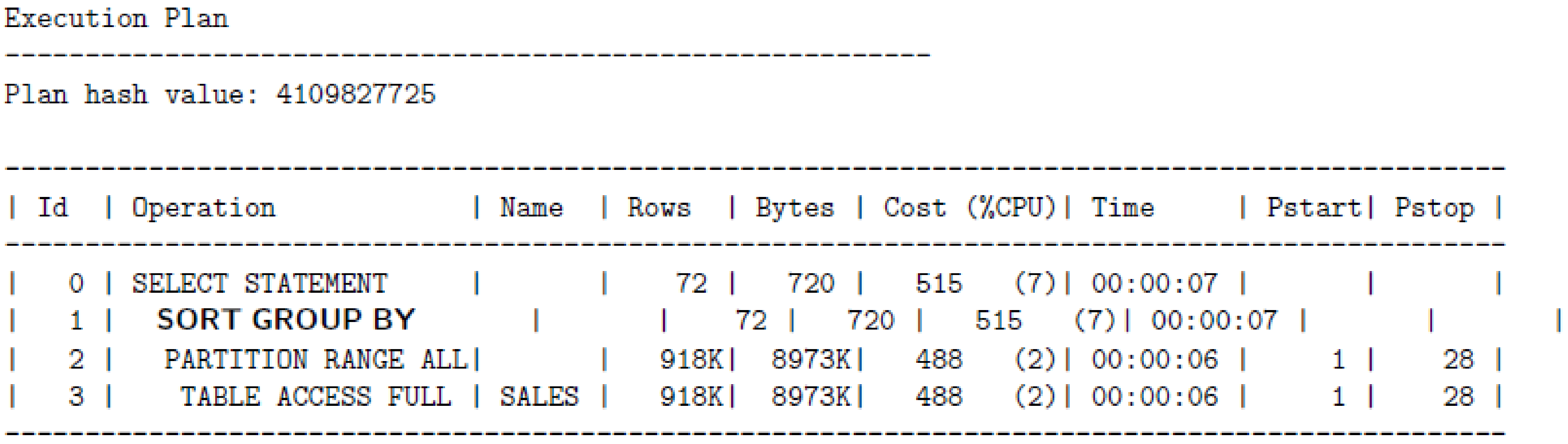
HASH GROUP BY

- After 10g Oracle starts to use HASH GROUP BY instead of SORT GROUP BY more extensively as it is appropriate.
- This is fundamentally related with hashing has a lower time complexity (*O(n)*) than sorting (*O(nlogn)*).
- DISTINCT clause inhibits Oracle from using HASH GROUP BY and force it to utilize SORT GROUP BY instead.
- And some unlucky Telco customers heavily utilizes DISTINCT COUNT clause in their queries (number of distinct subscribers).

Yet Another Simple Query on *SH.SALES*

```
select sum(QUANTITY_SOLD) total_sold ,
       count(distinct channel_id) ndiff_channel
from sh.sales
group by prod_id;
```

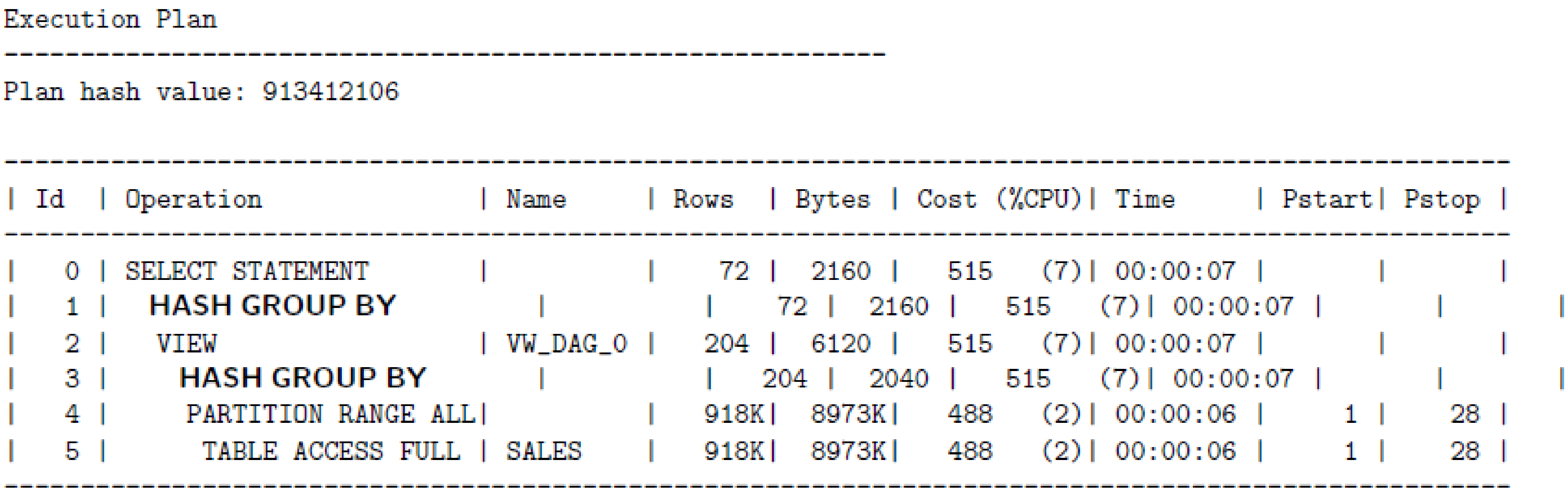
Pre 11.2.0.1 Execution Plan



```
Execution Plan
----------------------------------------------------------
Plan hash value: 4109827725

---------------------------------------------------------------------------------------------
| Id  | Operation            | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
---------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |       |    72 |   720 |   515   (7)| 00:00:07 |       |       |
|   1 |  SORT GROUP BY       |       |    72 |   720 |   515   (7)| 00:00:07 |       |       |
|   2 |   PARTITION RANGE ALL|       |  918K|  8973K|   488   (2)| 00:00:06 |     1 |    28 |
|   3 |    TABLE ACCESS FULL | SALES |  918K|  8973K|   488   (2)| 00:00:06 |     1 |    28 |
---------------------------------------------------------------------------------------------
```

husnu.sensoy@globalmaksimum.com

Execution Plan by 11.2.0.1



```
Execution Plan
----------------------------------------------------------
Plan hash value: 913412106

-----------------------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          |    72 |  2160 |   515   (7)| 00:00:07 |       |       |
|   1 |  HASH GROUP BY         |          |    72 |  2160 |   515   (7)| 00:00:07 |       |       |
|   2 |   VIEW                 | VW_DAG_0 |   204 |  6120 |   515   (7)| 00:00:07 |       |       |
|   3 |    HASH GROUP BY       |          |   204 |  2040 |   515   (7)| 00:00:07 |       |       |
|   4 |     PARTITION RANGE ALL|          |  918K|  8973K|   488   (2)| 00:00:06 |     1 |    28 |
|   5 |      TABLE ACCESS FULL | SALES    |  918K|  8973K|   488   (2)| 00:00:06 |     1 |    28 |
-----------------------------------------------------------------------------------------------------
```

Be Careful !!!

```
select sum(QUANTITY_SOLD) total_sold ,
        count(distinct channel_id) ndiff_channel,
        count(distinct time_id) ndiff_time
from sh.sales
group by prod_id;
```

Even in 11.2.0.1

```
Execution Plan
----------------------------------------------------------
Plan hash value: 4109827725

---------------------------------------------------------------------------------------------
| Id  | Operation            | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
---------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |       |    72 |  1296 |   515   (7)| 00:00:07 |       |       |
|   1 |  SORT GROUP BY       |       |    72 |  1296 |   515   (7)| 00:00:07 |       |       |
|   2 |   PARTITION RANGE ALL|       |   918K|    15M|   488   (2)| 00:00:06 |     1 |    28 |
|   3 |    TABLE ACCESS FULL | SALES |   918K|    15M|   488   (2)| 00:00:06 |     1 |    28 |
---------------------------------------------------------------------------------------------
```

Remarks

- I believe, this feature have no customer coverage as much as others but if you are one of those *distinct counters*, you will definitely benefit from it.
- Actually the part I have introduced is a part of all hash group by optimizations introduced with 11g Release 2. For appropriate use of all optimizations you might need to fix *Bug 9148171* in 11.2.0.1.
- More than one distinct count do not work.
- This option can be disabled by setting `_optimizer_distinct_agg_transform` parameter to `FALSE`.

Conclusion

- There are many more optimized analytical processing capabilities introduced in Oracle 11g Release 2.
- Those are all about fine tuning the existing features instead of introducing new fancy ones.
- And to be honest that’s what large customers want.

Q & A