Quick Wins in Optimized Analytical Processing Capabilities of 11g Release 2

Husnu Sensoy
husnu.sensoy@gmail.com

Global Maksimum Data & Information Technologies

May 18, 2010

Content

Who am I ?

- Recently formed up my own consulting company
- Previously  VLDB Expert
- Oracle ACE
- Oracle DBA of 2009
- Oracle Blogger in ▸ The great grandson of Husnu Sensoy
- Speaker in various meetings like Open World, User Groups, and Universities

What are they ?

- Optimized Analytical Processing Capabilities are those features implemented by different Oracle development teams like CBO,SQL execution, and expression management transparently improving SQL performance for your data crunching SQL statements.

What are they ?

- Optimized Analytical Processing Capabilities are those features implemented by different Oracle development teams like CBO,SQL execution, and expression management transparently improving SQL performance for your data crunching SQL statements.
- The keyword is *transparency*. By default you do not need to change any configuration to enable those capabilities.

What are they ?

- Optimized Analytical Processing Capabilities are those features implemented by different Oracle development teams like CBO,SQL execution, and expression management transparently improving SQL performance for your data crunching SQL statements.
- The keyword is *transparency*. By default you do not need to change any configuration to enable those capabilities.
- Oracle keeps saying SQL is X times faster in this release because of those features.

What are they ?

- Optimized Analytical Processing Capabilities are those features implemented by different Oracle development teams like CBO,SQL execution, and expression management transparently improving SQL performance for your data crunching SQL statements.
- The keyword is *transparency*. By default you do not need to change any configuration to enable those capabilities.
- Oracle keeps saying SQL is X times faster in this release because of those features.
- It is usually very hard to hear about them until the product is old enough or some of them cause problems in your production.

Why are they important ?

- In 10g one of the most important headaches for large DWH customers stem from 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.

Why are they important ?

- In 10g one of the most important headaches for large DWH customers stem from 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 some plan changes in new release.

Why are they important ?

- In 10g one of the most important headaches for large DWH customers stem from 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 some plan changes in new release.
- Just to appreciate the effort made by those developers optimizing our lives.

Partition Pruning

- In one of recent surveys, Oracle partitioning seems to be the Top 1 feature used by large DWH sites.

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 lets ILM for our data warehouses but also improves query performance by partition pruning most of the time.

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 lets ILM for our data warehouses 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 if both are possible.

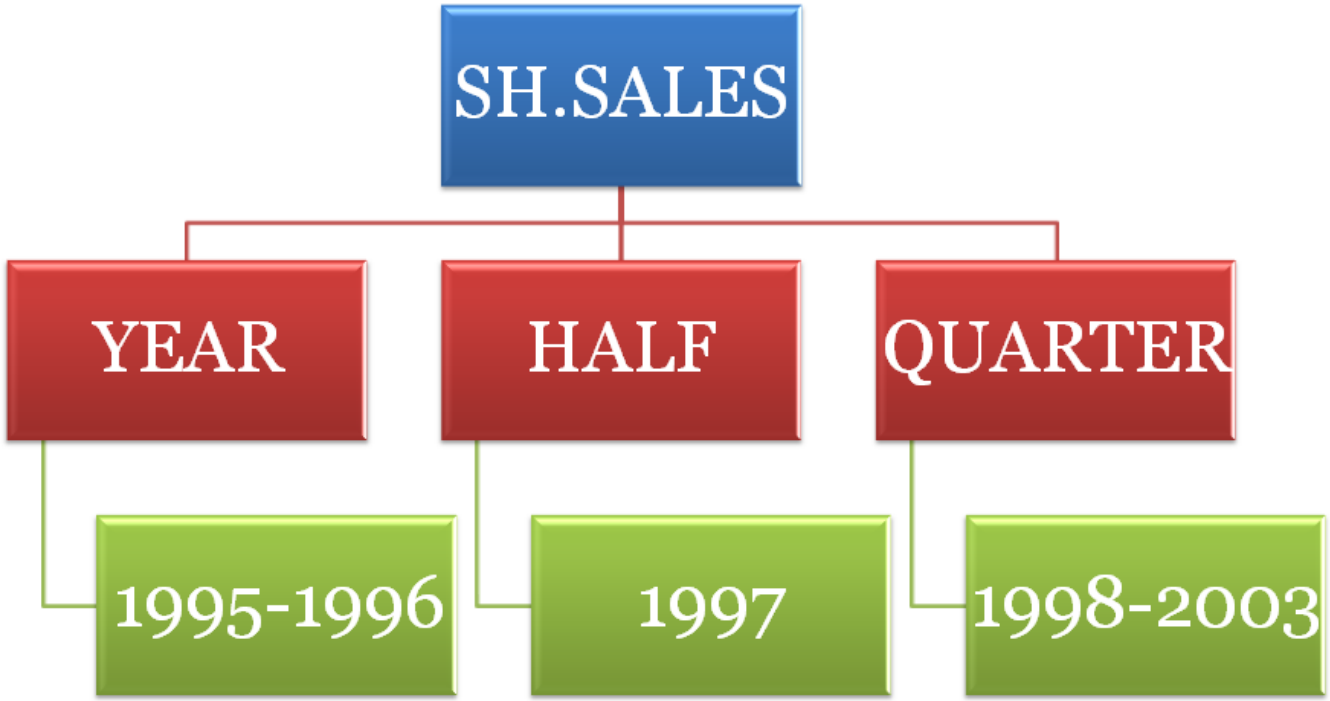
Partition Pruning

- In one of recent surveys, Oracle partitioning seems to be the Top 1 feature used by large DWH sites.
- Range partitioning not only lets ILM for our data warehouses 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 if both are possible.
- Multi-predicate pruning is based on boosting each and every pruning opportunity to reduce the amount of data to be read from disk or buffer cache.

A Data Warehouse Query...

```
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');
```

Partitioning Scheme for *SH.SALES*



Possible Partition Pruning Opportunities

No Pruning Scan all 28 partitions

Possible Partition Pruning Opportunities

No Pruning Scan all 28 partitions

Static Pruning Prune over `time_id` column down to 13 partitions

Possible Partition Pruning Opportunities

No Pruning Scan all 28 partitions

Static Pruning Prune over `time_id` column down to 13 partitions

Dynamic Pruning Build a filter list for *February* on *times* table then access to *sales* which results in accessing only 5 partitions

Possible Partition Pruning Opportunities

No Pruning Scan all 28 partitions

Static Pruning Prune over `time_id` column down to 13 partitions

Dynamic Pruning Build a filter list for *February* on *times* table then access to *sales* which results in accessing only 5 partitions

Both Static & Dynamic Pruning Using static and dynamic pruning together will yield a better result as expected. Oracle will filter out 25 paritions and access only 1998_Q1,1999_Q1,and 2000_Q1.

Multi-predicate Partition Pruning

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 |
-------------------------------------------------------------------------------------------------

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

   2 - access("S"."TIME_ID"="T"."TIME_ID")
   3 - filter("T"."FISCAL_MONTH_NAME"='February' AND "T"."TIME_ID"<=TO_DATE(' 2001-01-01
              00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "T"."TIME_ID">=TO_DATE(' 1998-01-01 00:00:00',
              'syyyy-mm-dd hh24:mi:ss'))
   5 - filter("S"."TIME_ID"<=TO_DATE(' 2001-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
        764  consistent gets
          0  physical reads
          0  redo size
        229  bytes sent via SQL*Net to client
        248  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

In 11.2.0.1

```
set autot trace exp stat
alter session set tracefile_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;
```

Multi-predicate Partition Pruning

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)|
-------------------------------------------------------------------------------------------------------------

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

   2 - access("S"."TIME_ID"="T"."TIME_ID")
   4 - filter("T"."FISCAL_MONTH_NAME"='February' AND "T"."TIME_ID"<=TO_DATE(' 2001-01-01
              00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND "T"."TIME_ID">=TO_DATE(' 1998-01-01 00:00:00',
              'syyyy-mm-dd hh24:mi:ss'))
   6 - filter("S"."TIME_ID"<=TO_DATE(' 2001-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
        200  consistent gets
          0  physical reads
          0  redo size
        344  bytes sent via SQL*Net to client
        411  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

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 the top 1 idea of very large data management

Remarks

- Partitioning is and will be the top 1 idea of very large data management
- Multi-predicate partition pruning really boosts Oracle's pruning opportunities for cases where several predicates can result in pruning.

Partially Indexing

- As you may all know, Oracle allows `UNUSABLE` index partitions from early releases of partitioning technology

Partially Indexing

- As you may all know, Oracle allows `UNUSABLE` index partitions from early releases of partitioning technology
- Many data warehouses wish to disable some old index partitions to reveal the burden of maintaning them durint ELT processes.

Partially Indexing

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

Partially Indexing

- As you may all know, Oracle allows `UNUSABLE` index partitions from early releases of partitioning technology
- Many data warehouses wish to disable some old index partitions to reveal the burden of maintaning them durint ELT processes.
- Intelligent Multi-Branch Execution is a query rewrite technique to split a single SQL statement based on a partitioned table having LOCAL index into two
  1. `USABLE` index partitions

Partially Indexing

- As you may all know, Oracle allows `UNUSABLE` index partitions from early releases of partitioning technology
- Many data warehouses wish to disable some old index partitions to reveal the burden of maintaning them durint ELT processes.
- Intelligent Multi-Branch Execution is a query rewrite technique to split a single SQL statement based on a partitioned table having LOCAL index into two
  1. `USABLE` index partitions
  2. `UNUSABLE` index partitions

Another Data Warehouse Query...

```
select  channels.channel_desc,
        sum(sales.amount_sold) total_amount
  from  sales, products,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

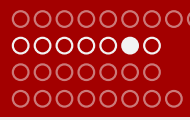
```
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name                 | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
----------------------------------------------------------------------------------------------------------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

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

   3 - access("CHANNELS"."CHANNEL_ID"="SALES"."CHANNEL_ID")
   4 - filter("CHANNELS"."CHANNEL_CLASS"='Direct')
  13 - access("PRODUCTS"."PROD_CATEGORY"='Photo')
  14 - access("SALES"."PROD_ID"="PRODUCTS"."PROD_ID")
  18 - filter("CHANNELS"."CHANNEL_CLASS"='Direct')
  19 - access("SALES"."CHANNEL_ID"="CHANNELS"."CHANNEL_ID")
```

Alter LOCAL Index Partitions UNUSABLE

```
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;
```

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



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.

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`

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`.

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).

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 data warehouse queries containing above clauses.

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 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 resultsets) and put a `FILTER` step insted (▸ Refer one of my earlier blog posts).

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 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 resultsets) and put a `FILTER` step insted (▸ Refer one of my earlier blog posts).
- `FILTER` is usually CPU consuming and never-ending option for the join of large datasets.

Another Data Warehouse Query...

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

Anti Join Execution Plan

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

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

   2 - access("TIME_ID"="TIME_ID")
```

Disable NULL Constraint in 10.2.0.4

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

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

Disable NULL Constraint in 11.1.0.6+

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

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

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

   2 - access("TIME_ID"="TIME_ID")
```

How about this ?

```
select count(*)
from call_records
where (day_id,
       month_id,
       year_id) not in (select day_id,
                               month_id,
                               year_id
                        from times);
```

Remarks

- NULL aware anti join is a great enhancement for constraint ignorant databases.

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

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
- As far as I have tested classical anti join is still faster for large queries.

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
- As far as I have tested classical anti join is still faster for large queries.
- So SNA is a remedial solution for erroneous design. Not a way to cheat SQL design best practices.

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
- As far as I have tested classical anti join is still faster for large queries.
- So SNA is a remedial solution for erroneous design. Not a way to cheat SQL design best practices.
- This option can be disabled by setting `_optimizer_null_aware_antijoin` parameter to `FALSE` (It seems to be not functional for 11g Release 2).

Hash Group by

- After 10g Oracle starts to use `HASH GROUP BY` instead of `SORT GROUP BY` more extensively as it is appropriate.

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 complexity ($O(n)$) than sorting ($O(nlogn)$)
  - Modern machines have more and more memory to satify memory needs of hashing.

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 complexity ($O(n)$) than sorting ($O(nlogn)$)
  - Modern machines have more and more memory to satify memory needs of hashing.
- `DISTINCT` clause inhibits Oracle from using `HASH GROUP BY` and force it to utilize `SORT GROUP BY` instead.

Hash Group by

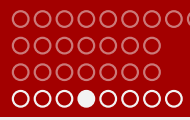
- After 10g Oracle starts to use `HASH GROUP BY` instead of `SORT GROUP BY` more extensively as it is appropriate.
- This is fundamentally related with
  - Hashing has a lower complexity (*O(n)*) than sorting (*O(nlogn)*)
  - Modern machines have more and more memory to satify memory needs of hashing.
- `DISTINCT` clause inhibits Oracle from using `HASH GROUP BY` and force it to utilize `SORT GROUP BY` instead.
- And some unlucky customers like  heavility utilizes `DISTINCT COUNT` clause in their queries (number of distinct subscribers doing something).

Another Data Warehouse Query...

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



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

On My VirtualBox Instance

```
select executions,cpu_time,elapsed_time,sorts,
       RUNTIME_MEM,SHARABLE_MEM,PERSISTENT_MEM
from v\$sql
where sql_id = :sqlid;
```

On My VirtualBox Instance

```
select executions,cpu_time,elapsed_time,sorts,
       RUNTIME_MEM,SHARABLE_MEM,PERSISTENT_MEM
from v\$sql
where sql_id = :sqlid;
```

SORT GROUP BY

EXECUTION	CPU TIME	ELAPSED TIME	SORTS	RUNTIME MEMORY	DISK READS	BUFFER GETS
1	1423784	2142388	1	3280	1793	5643

On My VirtualBox Instance

```
select executions,cpu_time,elapsed_time,sorts,
       RUNTIME_MEM,SHARABLE_MEM,PERSISTENT_MEM
from v\$sql
where sql_id = :sqlid;
```

SORT GROUP BY

EXECUTION	CPU TIME	ELAPSED TIME	SORTS	RUNTIME MEMORY	DISK READS	BUFFER GETS
1	1423784	2142388	1	3280	1793	5643

HASH GROUP BY

EXECUTION	CPU TIME	ELAPSED TIME	SORTS	RUNTIME MEMORY	DISK READS	BUFFER GETS
1	843872	1113372	0	3884	1832	5643

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

- 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.

Remarks

- 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.

Remarks

- 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.
- More than one distinct count does not work.

Remarks

- 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.
- More than one distinct count does not work.
- This option can be disabled by setting `_optimizer_distinct_agg_transform` parameter to `FALSE`.

To sum up...

- There are many more optimized analytical processing capabilities introduced in Oracle 11g Release 2.

To sum up...

- There are many more optimized analytical processing capabilities introduced in Oracle 11g Release 2.
- Those are all about tweaking the existing features instead of introducing new fancy ones.

To sum up...

- There are many more optimized analytical processing capabilities introduced in Oracle 11g Release 2.
- Those are all about tweaking the existing features instead of introducing new fancy ones.
- And to be honest thats what large customers want. They do not want to see *exceptional cases*.

To sum up...

- There are many more optimized analytical processing capabilities introduced in Oracle 11g Release 2.
- Those are all about tweaking the existing features instead of introducing new fancy ones.
- And to be honest thats what large customers want. They do not want to see *exceptional cases*.
- Keep in mind that Oracle is a software meaning that there might be bugs. Never take those features for granted.

Tack