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

1. Introduction

2. New Capabilities by 11g R2
   - Multi-predicate Partition Pruning
   - Intelligent Multi-Branch Execution
   - NULL Aware ANTI JOIN
   - Hash-based Distinct Aggregation

3. Conclusion
Who am I?

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

Husnu Sensoy husnu.sensoy@gmail.com

Global Maksimum Data & Information Technologies

Quick Wins in Optimized Analytical Processing Capabilities of 11g Release 2
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.
In one of recent surveys, Oracle partitioning seems to be the Top 1 feature used by large DWH sites.
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.
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.
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.
Multi-predicate Partition Pruning

A Data Warehouse Query...

```sql
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**

- **YEAR**
  - 1995-1996
- **HALF**
  - 1997
- **QUARTER**
  - 1998-2003
Multi-predicate Partition Pruning

Possible Partition Pruning Opportunities

No Pruning  Scan all 28 partitions
Multi-predicate Partition Pruning

Possible Partition Pruning Opportunities

No Pruning  Scan all 28 partitions
Static Pruning  Prune over time_id column down to 13 partitions
Multi-predicate Partition Pruning

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 partitions 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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>24</td>
<td>217 (11)</td>
<td>00:00:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>HASH JOIN</td>
<td></td>
<td>41164</td>
<td>964K</td>
<td>217 (11)</td>
<td>00:00:03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>TIMES</td>
<td>684K</td>
<td>5344K</td>
<td>202 (9)</td>
<td>00:00:03</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>PARTITION RANGE ITERATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>684K</td>
<td>5344K</td>
<td>202 (9)</td>
<td>00:00:03</td>
<td>5</td>
<td>17</td>
</tr>
</tbody>
</table>

**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

---

Husnu Sensoy husnu.sensoy@gmail.com

Quick Wins in Optimized Analytical Processing Capabilities of 11g Release 2
Multi-predicate Partition Pruning

**In 11.2.0.1**

```sql
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>24</td>
<td>322 (8)</td>
<td>00:00:05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SURT AGGREGATE</td>
<td></td>
<td>1</td>
<td>24</td>
<td>322 (8)</td>
<td>00:00:05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>HASH JOIN</td>
<td></td>
<td>43252</td>
<td>1013K</td>
<td>322 (8)</td>
<td>00:00:05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 3</td>
<td>PART JOIN FILTER CREATE</td>
<td>BF0000</td>
<td>91</td>
<td>1456</td>
<td>13 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 4</td>
<td>TABLE ACCESS FULL</td>
<td>TIMES</td>
<td>91</td>
<td>1456</td>
<td>13 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PARTITION RANGE AND</td>
<td></td>
<td>690K</td>
<td>5393K</td>
<td>303 (7)</td>
<td>00:00:05</td>
<td>KEY(AP)</td>
<td>KEY(AP)</td>
</tr>
<tr>
<td>* 6</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>690K</td>
<td>5393K</td>
<td>303 (7)</td>
<td>00:00:05</td>
<td>KEY(AP)</td>
<td>KEY(AP)</td>
</tr>
</tbody>
</table>

**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'))
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
Multi-predicate Partition Pruning

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
...
Multi-predicate Partition Pruning

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.
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 maintaining them during ELT processes.
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 maintaining them during 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
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 maintaining them during 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
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 maintaining them during 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...

```sql
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;
```
Intelligent Multi-Branch Execution

Execution Plan

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

Predicate Information (identified by operation id):

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

Husnu Sensoy husnu.sensoy@gmail.com

Quick Wins in Optimized Analytical Processing Capabilities of 11g Release 2
Intelligent Multi-Branch Execution

**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;
```
### Intelligent Multi-Branch Execution

In **10.2.0.4**

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

**In 11.2.0.1**

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

Husnu Sensoy husnu.sensoy@gmail.com

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

- Intelligent Multi-Branch Execution is an invaluable new optimization for sites using LOCAL indexes.
Intelligent Multi-Branch Execution

Remarks

- Intelligent Multi-Branch Execution is an invaluable new optimization for sites using LOCAL indexes.
- Keep in mind in order to use this optimization the `SKIP_UNUSABLE_INDEXES` parameter should set to be TRUE.
Intelligent Multi-Branch Execution

Remarks

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

```sql
select count(*)
from sh.sales
where time_id not in (select time_id
from sh.times);
```
NULL Aware ANTI JOIN

Anti Join Execution Plan

Execution Plan

Plan hash value: 397380204

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

Predicate Information (identified by operation id):

2 - access("TIME_ID"="TIME_ID")
NULL Aware ANTI JOIN

Disable NULL Constraint in 10.2.0.4

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

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

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))
NULL Aware ANTI JOIN

Disable NULL Constraint in 11.1.0.6+

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

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

Predicate Information (identified by operation id):

2 - access("TIME_ID"="TIME_ID")
How about this?

```sql
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.
NULL Aware ANTI JOIN

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.
NULL Aware ANTI JOIN

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.

Husnu Sensoy husnu.sensoy@gmail.com Global Maksimum Data & Information Technologies
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-based Distinct Aggregation

Hash Group by

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

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 \( O(n \log n) \)
  - Modern machines have more and more memory to satisfy memory needs of hashing.
Hash-based Distinct Aggregation

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 \( (O(n\log n)) \)
  - Modern machines have more and more memory to satisfy memory needs of hashing.
- DISTINCT clause inhibits Oracle from using HASH GROUP BY and force it to utilize SORT GROUP BY instead.
Hash-based Distinct Aggregation

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 ($O(n\log n)$)
  - Modern machines have more and more memory to satisfy 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 (⊙请选择) heavility utilizes `DISTINCT COUNT` clause in their queries (number of distinct subscribers doing something).
Another Data Warehouse Query...

```sql
SELECT SUM(QUANTITY_SOLD) total_sold,
       COUNT(DISTINCT channel_id) ndiff_channel
FROM sh.sales
GROUP BY prod_id;
```
Hash-based Distinct Aggregation

Pre 11.2.0.1 Execution Plan

Execution Plan

Plan hash value: 4109827725

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>72</td>
<td>720</td>
<td>515</td>
<td>00:00:07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT GROUP BY</td>
<td></td>
<td>72</td>
<td>720</td>
<td>515</td>
<td>00:00:07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>918K</td>
<td>8973K</td>
<td>488</td>
<td>00:00:06</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>918K</td>
<td>8973K</td>
<td>488</td>
<td>00:00:06</td>
<td>1</td>
<td>28</td>
</tr>
</tbody>
</table>
Hash-based Distinct Aggregation

By 11.2.0.1

Execution Plan

Plan hash value: 913412106

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>72</td>
<td>2160</td>
<td>515 (7)</td>
<td>00:00:07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HASH GROUP BY</td>
<td></td>
<td>72</td>
<td>2160</td>
<td>515 (7)</td>
<td>00:00:07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>VIEW</td>
<td>VW_DAG_0</td>
<td>204</td>
<td>6120</td>
<td>515 (7)</td>
<td>00:00:07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>HASH GROUP BY</td>
<td></td>
<td>204</td>
<td>2040</td>
<td>515 (7)</td>
<td>00:00:07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>918K</td>
<td>8973K</td>
<td>488 (2)</td>
<td>00:00:06</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>918K</td>
<td>8973K</td>
<td>488 (2)</td>
<td>00:00:06</td>
<td>1</td>
<td>28</td>
</tr>
</tbody>
</table>
Hash-based Distinct Aggregation

On My VirtualBox Instance

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

On My VirtualBox Instance

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

<table>
<thead>
<tr>
<th>EXECUTION</th>
<th>CPU_TIME</th>
<th>ELAPSED_TIME</th>
<th>SORTS</th>
<th>RUNTIME_MEMORY</th>
<th>DISK READS</th>
<th>BUFFER GETS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1423784</td>
<td>2142388</td>
<td>1</td>
<td>3280</td>
<td>1793</td>
<td>5643</td>
</tr>
</tbody>
</table>
**On My VirtualBox Instance**

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

<table>
<thead>
<tr>
<th>EXECUTION</th>
<th>CPU TIME</th>
<th>ELAPSED TIME</th>
<th>SORTS</th>
<th>RUNTIME MEMORY</th>
<th>DISK READS</th>
<th>BUFFER GETS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>142378</td>
<td>214238</td>
<td>1</td>
<td>3280</td>
<td>1793</td>
<td>5643</td>
</tr>
</tbody>
</table>

**Hash Group By**

<table>
<thead>
<tr>
<th>EXECUTION</th>
<th>CPU TIME</th>
<th>ELAPSED TIME</th>
<th>SORTS</th>
<th>RUNTIME MEMORY</th>
<th>DISK READS</th>
<th>BUFFER GETS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>843872</td>
<td>111337</td>
<td>0</td>
<td>3884</td>
<td>1832</td>
<td>5643</td>
</tr>
</tbody>
</table>
Hash-based Distinct Aggregation

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;
```
### Hash-based Distinct Aggregation

### Even in 11.2.0.1

#### Execution Plan

Plan hash value: 4109827725

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
<th>Pstart</th>
<th>Pstop</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>72</td>
<td>1296</td>
<td>515 (7)</td>
<td>00:00:07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT GROUP BY</td>
<td></td>
<td>72</td>
<td>1296</td>
<td>515 (7)</td>
<td>00:00:07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>918K</td>
<td>15M</td>
<td>488 (2)</td>
<td>00:00:06</td>
<td>1</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>918K</td>
<td>15M</td>
<td>488 (2)</td>
<td>00:00:06</td>
<td>1</td>
<td>28</td>
</tr>
</tbody>
</table>
Hash-based Distinct Aggregation

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 has 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 that's 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 that's 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