Inside the Oracle Database 11g Optimizer
Removing the black magic

Hermann Bär
Data Warehousing Product Management, Server Technologies
Goals of this session

We will

• Provide a common understanding of what is important for the optimizer and how (s)he thinks
• Introduce “new” functionality to and how to leverage it
• Give you a set of simple steps for success

We will not

• Provide a sudden enlightenment and make everybody an expert
• Cover all functionality of the Optimizer
• Be able to solve all your problems
Agenda

• What drives the Optimizer?
• Use new functionality to your advantage
• Simple rules for success
• Q&A
## A typical execution plan

- Lots of information

---

Plan hash value: 1873588300

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>E-Bytes</th>
<th>Cost (CPU)</th>
<th>E-Time</th>
<th>Pstart</th>
<th>Pstop</th>
<th>TQ</th>
<th>TIN-OUT</th>
<th>PQ Distrib</th>
<th>A-Rows</th>
<th>A-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>213 (100)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>71</td>
</tr>
<tr>
<td>1</td>
<td>PX COORDINATOR</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>71</td>
</tr>
<tr>
<td>2</td>
<td>PX SEND QC (RANDOM)</td>
<td>:TQ10004</td>
<td>1</td>
<td>71</td>
<td>3537</td>
<td>213</td>
<td>(8)</td>
<td>0:00:00:03</td>
<td></td>
<td>1</td>
<td>31.04</td>
<td>T-&gt;P</td>
<td>QC (RAND)</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>HASH GROUP BY</td>
<td></td>
<td>4</td>
<td>71</td>
<td>3337</td>
<td>213</td>
<td>(8)</td>
<td>0:00:00:08</td>
<td></td>
<td>1</td>
<td>31.04</td>
<td>PCWF</td>
<td></td>
<td>71</td>
</tr>
<tr>
<td>0/01</td>
<td>PX RECEIVE</td>
<td></td>
<td>4</td>
<td>72</td>
<td>3394</td>
<td>212</td>
<td>(8)</td>
<td>0:00:00:03</td>
<td></td>
<td>1</td>
<td>31.04</td>
<td>PCWF</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>1</td>
<td>PX SEND HASH</td>
<td>:TQ10003</td>
<td>0</td>
<td>72</td>
<td>3394</td>
<td>212</td>
<td>(8)</td>
<td>0:00:00:03</td>
<td></td>
<td>1</td>
<td>31.03</td>
<td>P-&gt;P</td>
<td>HASH</td>
<td>0</td>
</tr>
<tr>
<td>* 6</td>
<td>HASH JOIN</td>
<td></td>
<td>4</td>
<td>72</td>
<td>3394</td>
<td>212</td>
<td>(8)</td>
<td>0:00:00:02</td>
<td></td>
<td>1</td>
<td>31.03</td>
<td>PCWF</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>0/01</td>
<td>PX RECEIVE</td>
<td></td>
<td>4</td>
<td>72</td>
<td>1224</td>
<td>208</td>
<td>(8)</td>
<td>0:00:00:03</td>
<td></td>
<td>1</td>
<td>31.03</td>
<td>PCWF</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>1</td>
<td>PX SEND HASH</td>
<td>:TQ10002</td>
<td>0</td>
<td>72</td>
<td>1224</td>
<td>208</td>
<td>(8)</td>
<td>0:00:00:03</td>
<td></td>
<td>1</td>
<td>31.02</td>
<td>P-&gt;P</td>
<td>HASH</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>VIEW</td>
<td>VM_UNC.5</td>
<td>4</td>
<td>72</td>
<td>1224</td>
<td>208</td>
<td>(8)</td>
<td>0:00:00:03</td>
<td></td>
<td>1</td>
<td>31.02</td>
<td>PCWF</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>0/01</td>
<td>PX RECEIVE</td>
<td></td>
<td>4</td>
<td>72</td>
<td>504</td>
<td>208</td>
<td>(8)</td>
<td>0:00:00:03</td>
<td></td>
<td>1</td>
<td>31.02</td>
<td>PCWF</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>10</td>
<td>HASH GROUP BY</td>
<td></td>
<td>4</td>
<td>72</td>
<td>504</td>
<td>208</td>
<td>(8)</td>
<td>0:00:00:03</td>
<td></td>
<td>1</td>
<td>31.02</td>
<td>PCWF</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>11</td>
<td>PX SEND HASH</td>
<td>:TQ10001</td>
<td>0</td>
<td>72</td>
<td>504</td>
<td>208</td>
<td>(8)</td>
<td>0:00:00:03</td>
<td></td>
<td>1</td>
<td>31.01</td>
<td>P-&gt;P</td>
<td>HASH</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>HASH GROUP BY</td>
<td></td>
<td>4</td>
<td>72</td>
<td>504</td>
<td>208</td>
<td>(8)</td>
<td>0:00:00:03</td>
<td></td>
<td>1</td>
<td>31.01</td>
<td>PCWF</td>
<td></td>
<td>287</td>
</tr>
<tr>
<td>0/01</td>
<td>PX BLOCK ITERATOR</td>
<td></td>
<td>4</td>
<td>1043K</td>
<td>7172K</td>
<td>196</td>
<td>(2)</td>
<td>0:00:00:03</td>
<td>1</td>
<td>29</td>
<td>31.01</td>
<td>PCUC</td>
<td></td>
<td>1043K</td>
</tr>
<tr>
<td>* 15</td>
<td>TABLE ACCESS FULL SALES</td>
<td>45</td>
<td>1043K</td>
<td>7172K</td>
<td>196</td>
<td>(2)</td>
<td>0:00:00:03</td>
<td>1</td>
<td>29</td>
<td>31.01</td>
<td>PCWF</td>
<td></td>
<td>1043K</td>
<td>100:00:01:87</td>
</tr>
<tr>
<td>16</td>
<td>BUFFER SORT</td>
<td></td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0/01</td>
<td>PX RECEIVE</td>
<td></td>
<td>4</td>
<td>72</td>
<td>2160</td>
<td>3</td>
<td>(8)</td>
<td>0:00:00:01</td>
<td></td>
<td>1</td>
<td>31.03</td>
<td>PCWF</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>17</td>
<td>PX SEND HASH</td>
<td>:TQ10006</td>
<td>0</td>
<td>72</td>
<td>2160</td>
<td>3</td>
<td>(8)</td>
<td>0:00:00:01</td>
<td></td>
<td>1</td>
<td>104.03</td>
<td>S-&gt;P</td>
<td>HASH</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCTS</td>
<td>1</td>
<td>72</td>
<td>2160</td>
<td>3</td>
<td>(8)</td>
<td>0:00:00:01</td>
<td></td>
<td>1</td>
<td>104.03</td>
<td>PCWF</td>
<td></td>
<td>72</td>
</tr>
</tbody>
</table>
A typical execution plan

- Lots of information
- But looking at ONE aspect alone tells you A LOT

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>Bytesi Cost (XCPU)</th>
<th>E-Time</th>
<th>Pstart</th>
<th>Pstop</th>
<th>TQ</th>
<th>IN-OUT</th>
<th>PO Distri</th>
<th>A-Rows</th>
<th>A-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>213</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PX COORDINATOR</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PX SEND QC (RANDOM)</td>
<td>TQ10004</td>
<td>0</td>
<td>71</td>
<td>5557</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>HASH GROUP BY</td>
<td></td>
<td>4</td>
<td>71</td>
<td>5537</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PK RECEIVE</td>
<td></td>
<td>4</td>
<td>72</td>
<td>3894</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PX SEND HASH</td>
<td>TQ10003</td>
<td>0</td>
<td>72</td>
<td>3504</td>
<td>212</td>
<td>00:00:03</td>
<td></td>
<td></td>
<td>01.03</td>
<td>P-&gt;P</td>
<td>HASH</td>
<td>72</td>
</tr>
<tr>
<td>6</td>
<td>HASH JOIN</td>
<td></td>
<td>4</td>
<td>72</td>
<td>3204</td>
<td>212</td>
<td>00:00:03</td>
<td></td>
<td></td>
<td>01.03</td>
<td>PCWF</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>PK RECEIVE</td>
<td></td>
<td>4</td>
<td>72</td>
<td>1224</td>
<td>208</td>
<td>00:00:03</td>
<td></td>
<td></td>
<td>01.03</td>
<td>PCWF</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>PX SEND HASH</td>
<td>TQ10002</td>
<td>0</td>
<td>72</td>
<td>1224</td>
<td>208</td>
<td>00:00:03</td>
<td></td>
<td></td>
<td>01.02</td>
<td>P-&gt;P</td>
<td>HASH</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>VIEW</td>
<td></td>
<td>4</td>
<td>72</td>
<td>1224</td>
<td>208</td>
<td>00:00:03</td>
<td></td>
<td></td>
<td>01.03</td>
<td>PCWF</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>HASH GROUP BY</td>
<td></td>
<td>4</td>
<td>72</td>
<td>504</td>
<td>208</td>
<td>00:00:03</td>
<td></td>
<td></td>
<td>01.03</td>
<td>PCWF</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>PK RECEIVE</td>
<td></td>
<td>4</td>
<td>72</td>
<td>504</td>
<td>208</td>
<td>00:00:03</td>
<td></td>
<td></td>
<td>01.03</td>
<td>PCWF</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>PX SEND HASH</td>
<td>TQ10001</td>
<td>0</td>
<td>72</td>
<td>504</td>
<td>208</td>
<td>00:00:03</td>
<td></td>
<td></td>
<td>01.03</td>
<td>PCWF</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>HASH GROUP BY</td>
<td></td>
<td>4</td>
<td>72</td>
<td>504</td>
<td>208</td>
<td>00:00:03</td>
<td></td>
<td></td>
<td>01.03</td>
<td>PCWF</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>PK BLOCK ITERATOR</td>
<td></td>
<td>4</td>
<td>1049K</td>
<td>71724</td>
<td>196</td>
<td>22</td>
<td>00:00:03</td>
<td></td>
<td></td>
<td>01.03</td>
<td>PCWF</td>
<td>1049K</td>
</tr>
<tr>
<td>15</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>45</td>
<td>1049K</td>
<td>71724</td>
<td>196</td>
<td>(2)</td>
<td>00:00:03</td>
<td></td>
<td></td>
<td>01.01</td>
<td>PCWF</td>
<td>1049K</td>
</tr>
<tr>
<td>16</td>
<td>BUFFER SORT</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>PK RECEIVE</td>
<td></td>
<td>4</td>
<td>72</td>
<td>2160</td>
<td>3</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td>01.03</td>
<td>PCWF</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>PX SEND HASH</td>
<td>TQ10000</td>
<td>0</td>
<td>72</td>
<td>2160</td>
<td>3</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td>01.03</td>
<td>PCWF</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCTS</td>
<td>1</td>
<td>72</td>
<td>2160</td>
<td>3</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How many rows does the optimizer expect?
How many rows will actually be returned?
What drives the Optimizer?

• A lot of things have an influence on the Optimizer
• Cardinality is the most common “input” that is off

• Incorrect cardinality leads to
  • Wrong access methods
  • Wrong join types
  • Wrong join orders
Agenda

• What drives the Optimizer?
• Use new functionality to your advantage
• Simple rules for success
• Q&A
### Causes for incorrect cardinality estimates

<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stale or missing statistics</td>
<td></td>
</tr>
<tr>
<td>Data Skew</td>
<td></td>
</tr>
<tr>
<td>Multiple single column predicates on a table</td>
<td></td>
</tr>
<tr>
<td>Multiple columns used in a join</td>
<td></td>
</tr>
<tr>
<td>Function wrapped column</td>
<td></td>
</tr>
<tr>
<td>Complicated expression containing columns from multiple tables</td>
<td></td>
</tr>
</tbody>
</table>
## Causes for incorrect cardinality estimates

<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stale or missing statistics</td>
<td>Collect statistics</td>
</tr>
<tr>
<td>Data Skew</td>
<td></td>
</tr>
<tr>
<td>Multiple single column predicates on a table</td>
<td></td>
</tr>
<tr>
<td>Multiple columns used in a join</td>
<td></td>
</tr>
<tr>
<td>Function wrapped column</td>
<td></td>
</tr>
<tr>
<td>Complicated expression containing columns from multiple tables</td>
<td></td>
</tr>
</tbody>
</table>
Improved Efficiency and Quality

New statistics gathering algorithm

Business problem

- ".. Compute statistics gives accurate results but takes too long .."
- ".. Sampling is fast but not always accurate .."
- ".. AUTO SAMPLE SIZE does not always work with data skew .."

Solution

- New groundbreaking implementation
  - Faster than sampling
  - Accuracy comparable to compute statistics
- Used by default with AUTO_SAMPLE_SIZE value
- No need to use manual sampling anymore

FASTER AND BETTER

Speed of sampling with the accuracy of compute
Improved Efficiency and Quality
Incremental Statistics Maintenance

Business Requirement
- Gathering statistics on one partition (e.g. after a bulk load) causes a full scan of all partitions to gather global table statistics
- Extremely time consuming

Solution
- Gather statistics for touched partition(s) ONLY
- Table (global) statistics are built from partition statistics

Refeshed WITHOUT scanning the NON touched partitions
Incremental Global Statistics

1. Partition level stats are gathered & synopsis created

   May 18th 2008 \(\rightarrow\) S1
   May 19th 2008 \(\rightarrow\) S2
   May 20th 2008 \(\rightarrow\) S3
   May 21st 2008 \(\rightarrow\) S4
   May 22nd 2008 \(\rightarrow\) S5
   May 23rd 2008 \(\rightarrow\) S6

2. Global stats generated by aggregating partition level statistics and synopsis

   Global Statistic

   Sysaux Tablespace
Incremental Global Statistics Cont’d

3. A new partition is added to the table & Data is Loaded

5. Retrieve synopsis for each of the other partitions from Sysaux

6. Global stats generated by aggregating the original partition synopsis with the new one
Automatic concurrent Statistics gathering

10g Limitations

- Statistics are gathered one object at a time
- Resources not fully utilized during maintenance window
- Window ends before all stale statistics have been re-gathered

11g Solution

- Gather statistics on multiple objects at the same time
- Allows you to control the number of concurrent gather operations
- Each gather operation still operates in parallel
No concurrent Statistics gathering in 10g

Gather partition statistics

Each individual partition will have statistics gathered one after the other.

The statistics gather procedure on each individual partition can be operating in parallel BUT the statistics gathering procedures won’t happen concurrently.

Exec DBMS_STATS.GATHER_TABLE_STATS(null, 'SALES');
Concurrent Statistics gathering in 11.2.0.2

Sales Table

May 18th 2010
May 19th 2010
May 20th 2010
May 21st 2010
May 22nd 2010
May 23rd 2010
May 24th 2010
May 25th 2010

Gather partition statistics
Gather partition statistics
Gather partition statistics
Gather partition statistics

The number of concurrent gathers is controlled by the parameter job_queue_processes.

In this example it is set to 4.

Remember each concurrent gather can be operating in parallel.

Exec DBMS_STATS.GATHER_TABLE_STATS(null, 'SALES');

With Concurrent - only one partitioned table is allowed to be gathered at one time.
Concurrent Statistics gathering in 11.2.0.2
Non-partitioned tables

Exec DBMS_STATS.GATHER_TABLE_SCHEMA('SH');

The number of concurrent gathers is controlled by the parameter job_queue_processes. In this example it is set to 4

Sales Table
Cust Table
Prod Table
Promo Table

Gather statistics
Gather statistics
Gather statistics
Gather statistics

Remember each concurrent gather can be operating in parallel

Only one partitioned table is allowed to be gathered at one time
## Causes for incorrect cardinality estimates

<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stale or missing statistics</td>
<td>Collect statistics</td>
</tr>
<tr>
<td>Data Skew</td>
<td>Collect histograms</td>
</tr>
<tr>
<td>Multiple single column predicates on a table</td>
<td></td>
</tr>
<tr>
<td>Multiple columns used in a join</td>
<td></td>
</tr>
<tr>
<td>Function wrapped column</td>
<td></td>
</tr>
<tr>
<td>Complicated expression containing columns from multiple tables</td>
<td></td>
</tr>
<tr>
<td>Cause</td>
<td>Solution</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Stale or missing statistics</td>
<td>Collect statistics</td>
</tr>
<tr>
<td>Data Skew</td>
<td>Collect histograms</td>
</tr>
<tr>
<td>Multiple single column predicates on a table</td>
<td>Create extended statistics</td>
</tr>
<tr>
<td>Multiple columns used in a join</td>
<td>Create extended statistics</td>
</tr>
<tr>
<td>Function wrapped column</td>
<td>Create extended statistics</td>
</tr>
<tr>
<td>Complicated expression containing columns from multiple tables</td>
<td>Create extended statistics</td>
</tr>
</tbody>
</table>
Extended Optimizer Statistics

Business problem - Correlated Columns

- Real data often shows correlations between various attributes
  - e.g. job title influences salary, car model influences make, seasons affect the amount of sold goods (e.g. snow shoes in winter)
- Optimizer has to estimate the correct cardinality
  - “Does an additional filter reduce the result set or not?”

Solution

- Extended Optimizer Statistics provides a mechanism to collect statistics on a group of columns
- Full integration into existing statistics framework
  - Automatically maintained with column statistics
  - Instantaneous and transparent benefit for any migrated application

Improved cardinality leads to improved plans
Example: Single column Query

```
SELECT ......FROM..
WHERE model = '530xi'
```

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>RED</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>BLACK</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>SILVER</td>
</tr>
</tbody>
</table>

• Cardinality calculation #ROWS/NDV
  => 12/4 = 3
  • Single column statistics are accurate
Example: Multiple non-correlated Columns

```
SELECT ......FROM..
WHERE model = '530xi'
AND color = 'RED';
```

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>RED</td>
</tr>
</tbody>
</table>

- One record selected
- Cardinality = \( \text{#ROWS} \times \frac{1}{\text{NDV c1}} \times \frac{1}{\text{NDV c2}} \)
  - Cardinality = \( 12 \times \frac{1}{4} \times \frac{1}{3} = 1 \)

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>RED</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>BLACK</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>SILVER</td>
</tr>
<tr>
<td>PORSCHE</td>
<td>911</td>
<td>RED</td>
</tr>
<tr>
<td>MERC</td>
<td>SLK</td>
<td>RED</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>A-Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>VEHICLES</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
### Example: Multiple correlated Columns

```sql
SELECT ... FROM..
WHERE model = '530xi'
AND make = 'BMW';
```

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>RED</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>BLACK</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>SILVER</td>
</tr>
</tbody>
</table>

- Three records selected
- Cardinality: \( \text{#ROWS} \times \frac{1}{\text{NDV c1}} \times \frac{1}{\text{NDV c2}} \)
- Cardinality = \( 12 \times \frac{1}{4} \times \frac{1}{3} = 1 \)

### Vehicles table

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>RED</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>BLACK</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>SILVER</td>
</tr>
<tr>
<td>PORSCHE</td>
<td>911</td>
<td>RED</td>
</tr>
<tr>
<td>MERC</td>
<td>SLK</td>
<td>RED</td>
</tr>
<tr>
<td>MERC</td>
<td>C320</td>
<td>SLIVER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>A-Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>VEHICLES</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Solution

- Create extended statistics on the Model & Make columns

```sql
Select dbms_stats.create_extended_stats(
    Null, ' Vehicles', '(model,make)') From dual;
Exec dbms_stats.gather_table_stats( Null, ' Vehicles');

Select column_name, num_distinct, histogram
From user_tab_col_statistics
Where table_name = ' Vehicles';
```

NOTE: Column Group statistics only works with equality predicates & in-lists
Solution: Multiple correlated columns

SELECT ......FROM..
WHERE model = '530xi'
AND make = 'BMW';

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>RED</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>BLACK</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>SILVER</td>
</tr>
</tbody>
</table>

• Three records selected.
• Cardinality is correctly estimated from multi-column

Vehicles table

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>RED</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>BLACK</td>
</tr>
<tr>
<td>BMW</td>
<td>530xi</td>
<td>SILVER</td>
</tr>
<tr>
<td>PORSCHE</td>
<td>911</td>
<td>RED</td>
</tr>
<tr>
<td>MERCEDES</td>
<td>SLK</td>
<td>BLACK</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>A-Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>VEHICLES</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Multiple Join Columns

SELECT count(*)
FROM Sales s, Sales2 s2
WHERE s.cust_id = s2.cust_id
AND s.prod_id = s2.prod_id;

Cardinality calculation

#Rows T1 \times #Rows T2 \times \text{Selectivity of P1} \times \text{Selectivity of P2} \ldots

\#\text{Rows} \ T1 \times \#\text{Rows} \ T2 \times 1 \times 1
\quad \text{max}(\text{NDV} \ T1.c1, \text{NDV} \ T2.c1) \times \text{max}(\text{NDV} \ T1.c2, \text{NDV} \ T2.c2)

=> 918843 \times 918843 \times \frac{1}{7059} \times \frac{1}{7059} \times 1 \times 1
\quad \text{Max}(7059, 7059) \times \text{Max}(72, 72)

=> 918843 \times 918843 \times \frac{1}{7059} \times \frac{1}{7059} = 1,661,143 \text{ rows}
### Example of multiple column joins

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Starts</th>
<th>E-Rows</th>
<th>A-Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>* 2</td>
<td>HASH JOIN</td>
<td></td>
<td>1</td>
<td><strong>1661K</strong></td>
<td><strong>4422K</strong></td>
</tr>
<tr>
<td>3</td>
<td>PARTITION RANGE ALL</td>
<td></td>
<td>1</td>
<td>918K</td>
<td>918K</td>
</tr>
<tr>
<td>4</td>
<td>TABLE ACCESS FULL</td>
<td>SALES</td>
<td>28</td>
<td>918K</td>
<td>918K</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS FULL</td>
<td>SALES2</td>
<td>1</td>
<td>918K</td>
<td>918K</td>
</tr>
</tbody>
</table>

- Optimizer assumes each join condition will eliminate rows
  - That’s not the case here so estimate is 4 X off

- Solution
  - Create extended statistics on the join columns on each table
  - Remember column group statistics only work with equality predicates and in lists
Function wrapped Column

SELECT *  
FROM Customers  
WHERE UPPER(CUST_LAST_NAME) = 'SMITH';

- Optimizer doesn’t know how function affects values in the column so it guesses the cardinality to be 1% of rows

```
SELECT count(*) FROM customers;
COUNT(*)
55500
```

Cardinality estimate is 1% of the rows → 555 rows

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>405 (100)</td>
</tr>
<tr>
<td></td>
<td>SORT AGGREGATE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* 2 TABLE ACCESS FULL</td>
<td>CUSTOMERS</td>
<td>555</td>
<td>4440</td>
<td>405 (1)</td>
</tr>
</tbody>
</table>
```
Extended Statistics – in detail

- Use `dbms_stats` package
  - `Create_extended_stats`
    - Manually specify the group of columns
  - `Show_extended_stats_name`
    - Displays the system generated name for the column group
  - `Drop_extended_stats`
    - Drop a column group and all the statistics associated with it
- Monitor
  - New dictionary table `user_stat_extensions`
    - Shows sys generated name & actual column group desc
  - Look at dictionary table `user_tab_col_statistics`
    - New row with sys generated name will be add for each column group
# Causes for incorrect cardinality estimates

<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stale or missing statistics</td>
<td>Collect statistics</td>
</tr>
<tr>
<td>Data Skew</td>
<td>Collect histograms</td>
</tr>
<tr>
<td>Multiple single column predicates on a table</td>
<td>Create extended statistics</td>
</tr>
<tr>
<td>Multiple columns used in a join</td>
<td>Create extended statistics</td>
</tr>
<tr>
<td>Function wrapped column</td>
<td>Create extended statistics</td>
</tr>
<tr>
<td>Complicated expression containing columns from multiple tables</td>
<td>Use dynamic sampling level 4 or higher And SQL Plan Management</td>
</tr>
</tbody>
</table>
Complex Expressions

SELECT count(*)
FROM sales
WHERE cust_id < 2222
AND prod_id > 5;

Optimizer won’t use the column group statistics in this case because predicates are non-equalities

Cardinality estimate used instead is

#Rows T1 X Selectivity of P1 X Selectivity of P2……

\[
104920 \times \frac{V1 - \min(c1)}{\max(c2) - V2} \times \frac{\max(c1) - \Min(c1)}{\Max(c1) - \Min(c1)}
\]

\[
= 104920 \times \frac{2222 - 2}{148 - 5} \times \frac{148 - 13}{10100 - 2}
\]

\[
= 104920 \times \frac{2220}{143}
\]

\[
= 23,062
\]
Solution

- Dynamic sampling is only solution for complex expressions

Alter session set optimizer_dynamic_sampling = 4;

Predicate Information (identified by operation id):

3 - filter("CUST_ID"<2222 AND "PROD_ID">5))

Note
- dynamic sampling used for this statement (level=4)

Estimate is 10X off
Dynamic Sampling automatic for Parallel execution

• New in Oracle Database 11g Release 2
• Optimizer automatically decide if dynamic sampling will be useful & what level for Parallel SQL statements
  • Decision is based on size of the tables in the statement
  • The complexity of the predicates
• If OPTIMIZER_DYNAMIC_SAMPLING is explicitly set then that specified value will be honored
• You can tell if dynamic sampling kicks in by looking in the note section of the execution plan
SQL Plan Management

Business Requirement
• Unpredictable changes in execution plans can happen
  • New Statistics
  • Changes in the Environment
  • Software upgrades
• Today you have to ‘freeze’ critical plans or statistics

Solution
• Optimizer automatically manages ‘execution plans’
  • Only known and verified plans are used
• Plan changes are automatically verified
  • Only comparable or better plans are used going forward

SQL Plan Management is controlled plan evolution
SQL Plan Management

- SQL statement is parsed for the first time and a plan is generated
- Check the log to see if this is a repeatable SQL statement
- Add SQL statement signature to the log and execute it
- Plan performance is still “verified by execution”
SQL Plan Management

- SQL statement is parsed again and a plan is generated
- Check log to see if this is a repeatable SQL statement
- Create a Plan history and use current plan as SQL plan baseline
- Plan performance is “verified by execution”
SQL Plan Management

- Something changes in the environment
- SQL statement is parsed again and a **new plan is generated**
- New plan is not the same as the baseline – **new plan is not executed** but marked for verification
SQL Plan Management

• Something changes in the environment
• SQL statement is parsed again and a **new plan is generated**
• New plan is not the same as the baseline – **new plan is not executed** but marked for verification
• Execute known plan baseline - plan performance is “verify by history”
Verifying the new plan

- Non-baseline plans will not be used until verified
- DBA can verify plan at any time

 invoke or schedule verification

Optimizer checks if new plan is as good as or better than old plan

Plans which perform as good as or better than original plan are added to the plan baseline

Plans which don’t perform as good as the original plan stay in the plan history and are marked unaccepted
SQL Plan Management – the details

• Controlled by two init.ora parameter
  • `optimizer_capture_sql_plan_baselines`
    • Controls auto-capture of SQL plan baselines for repeatable stmts
    • Set to false by default in 11gR1
  • `optimizer_use_sql_plan_baselines`
    • Controls the use of existing SQL plan baselines by the optimizer
    • Set to true by default in 11gR1

• Monitoring SPM
  • Dictionary view `DBA_SQL_PLAN_BASELINES`
  • Via the SQL Plan Control in EM DBControl

• Managing SPM
  • PL/SQL package `DBMS_SPM`
  • Via the SQL Plan Control in EM DBControl
SPM Plan Capture – Bulk

• From SQL Tuning Set (STS)
  • Captures plan details for a (critical) set of SQL Statement
  • Load these plans into SPM as baseline plans
  • Next time statements are executed baseline plans will be used

• From Cursor Cache
  • Load plans from the cursor cache into SPM as baseline plans
    • Filters can be specified (SQL_ID, Module name, schema)
    • Next time statements are executed baseline plans will be used

• From staging table
  • SQL plan baselines can be captured on another system
  • Exported via a table (similar to statistics) and imported locally
  • Plan are “unpacked” from the table and loaded into SPM

• From Stored Outlines
  • Migrate previously created Stored Outlines to SQL plan baselines
Capturing plan from the Cursor Cache

2. Run all SQL in the Application and auto load SQL Plan Baselines with 10g plan

4. After plans are loaded change OFE to 11

3. Auto Capture 10g plans

5. 11g plan queue for verification

- Seeding the SQL Plan Baselines with 10g plans No plan change on upgrade
- After all SQL Plan Baselines are populated switch Optimizer_Features_Enable to 11g
  - new 11g plans will only be used after they have been verified
Capturing Plans using Stored outlines

1. Begin with CREATE_STORED_OUTLINES=true

2. Run all SQL in the Application and auto create a Stored Outline for each one

3. After Store Outlines are captured CREATE_STORED_OUTLINES=false

4. Upgrade to 11g

5. Migrate Stored Outlines into SPM
Capturing Plans using SQL Tuning Set

1. Create STS for critical statements
2. Upgrade to 11g
3. Bulk load plans into SPM
Capturing plans using an 11g test environment

1. Create baselines from tuned stmts
2. Create staging table & pack baselines into it
3. Export staging table
4. Import staging table
5. Unpack baselines into SPM

Production Database 11g
- GB
- HJ
- No plan regressions

Development / Test Database 11g
- GB
- HJ
- Well tuned plan

Baseline
Plan History
Plan
Baseline
DBA

Baseline plans staging table
Export staging table
Import staging table
Unpack baselines into SPM
Create staging table & pack baselines into it
Create baselines from tuned stmts
Agenda

• What drives the Optimizer?
• Use new functionality to your advantage
• Simple rules for success
• Q&A
Simple rules for success

• Seed column usage
  • For automatic histogram creation
    • Execute all SQL statements against empty tables
    • Import from another environment `DBMS_STATS.SEED_COL_USAGE`
  • For automatic column group creation (new in 11.2.0.2)
    • Via STS or via workload monitoring

• Turn on incremental statistics
  • Automatically creates global stats from partition level stats

• Turn on concurrency (new 11.2.0.2)
  • `DBMS_STATS.SET_*_PREFS('CONCURRENT', 'TRUE');`

• Gather statistics using default value for all parameters
  • `Exec dbms_stats.gather_schema_stats('<schema>');`
Automatic Column Group Creation

• Check we have column usage information for the table

SQL> select dbms_stats.report_col_usage(user, 'customers_test') from dual;

COLUMN USAGE REPORT FOR SH.CUSTOMERS_TEST
1. COUNTRY_ID : EQ
2. CUST_CITY : EQ
3. CUST_STATE_PROVINCE : EQ
4. (CUST_CITY, CUST_STATE_PROVINCE, COUNTRY_ID) : FILTER
5. (CUST_STATE_PROVINCE, COUNTRY_ID) : GROUP_BY

Create extended stats for customers_test based on usage

SQL> select dbms_stats.create_extended_stats(user, 'customers_test') from dual;

EXTENSIONS FOR SH.CUSTOMERS_TEST
1. (CUST_CITY, CUST_STATE_PROVINCE, COUNTRY_ID):
   SYS_STUMZ$C3AIHLPBROI#SKA58H_N created
2. (CUST_STATE_PROVINCE, COUNTRY_ID):
   SYS_STU#S#WF25Z#QAHIE#MOFFMM created
Setting up for Success

• If db_file_multiblock_read_count is not set
  • _db_file_optimizer_read_count = 8 --used for costing only
  • _db_file_exec_read_count = db_cache_size/sessions
    • Used for execution
    • Maximum value 1MB

• If db_file_multiblock_read_count is explicitly set
  • _db_file_optimizer_read_count = specified value
  • _db_file_exec_read_count = specified value
    • Seriously effects the costing model

Do not set db_file_multiblock_read_count
Some other things to mention ..

• Important Optimizer topics not covered
  • Pending statistics (Oracle Database 11g Release 1)
  • Adaptive cursor sharing (Oracle Database 11g Release 1)

• Pointers to remember
  • Optimizer page on O.com
  • Oracle Optimizer blog
    • old blog link
  • Maria.Colgan@oracle.com
    • THE Optimizer PM
Q & A