Effective use of column histograms

Paul Baumgartel
Charles Island Consulting
NYOUG, March 15, 2007
What are histograms?

• Data dictionary objects
• Document the distribution of values in a column
• Created by DBMS STATS
• If used, must be kept up to date
Why histograms?

• Oracle’s optimizer relies heavily on *cardinality* in deciding on execution plans

• Cardinality is derived from *selectivity*

• Selectivity is the fraction of rows the optimizer expects to find, based on SQL predicate
Why histograms? continued

- Cardinality is computed as the number of input rows multiplied by selectivity.
- Another way to express cardinality is the number of rows expected to result from a given predicate applied to a given row source.
Why histograms? continued

• DBMS_STATS by default calculates
  – The number of rows in a table
  – The number of distinct values in a column

• Without additional information, optimizer assumes that values are evenly distributed

• But if they’re not evenly distributed, inefficient execution plans can result
How the optimizer uses histograms

• It uses information about value distribution in creating execution plan

• Histograms can be useful for both indexed and non-indexed columns

• The two most common uses are
  – Determining whether to use an indexed access path
  – Determining join order
Without a histogram: example

- Consider a table of 1 million rows
- An indexed column contains 500K distinct values
- A query contains an equality predicate on the indexed column
- Values are not evenly distributed
  - 400K rows contain one value
  - The other 499,999 values occur in the other 600K rows
Without a histogram: example

- Selectivity = 1/500,000 = 0.000002
- Cardinality = 0.000002 * 1,000,000 = 2
- Number of rows expected = 2
- Optimizer will choose indexed access path
- Perfectly reasonable if the value in the predicate is not the “popular” value
Without a histogram: example

• Suppose the value in the predicate is the “popular” value
• Actual selectivity is \( \frac{400,000}{1,000,000} = 0.4 \)
• Actual cardinality is \( 0.4 \times 1,000,000 = 400,000 \)
• When retrieving 400,000 out of 1,000,000 rows, a full table scan is less expensive than indexed
• But optimizer doesn’t know this, and chooses to use index regardless of value in predicate
With a histogram: example

- The optimizer can see that one value is extremely popular
- If this value is used in the predicate, a full table scan will occur
- If any other value is used, an index lookup will occur
Histograms on non-indexed columns

- Despite emphasis on using histograms for indexed columns, they can also be useful for non-indexed columns.

- Optimizer uses them in this case to help determine join order.

- A row source that contains 400k rows is much less likely to appear early in a join order than one containing two rows.
Histograms can be great, but...

- They’re not useful when bind variables are employed in code. Why?
  - They require knowledge of the actual value in the predicate, but binds don’t provide it
  - Oracle does “bind variable peeking”: at parse time, it inspects the value bound to the variable and feeds that to the optimizer for all executions of the SQL
Cases for which histograms might be contraindicated

- On tables for which most SQL uses binds
- If you have reduced hard parsing by setting CURSOR_SHARING to FORCE or SIMILAR
- We all know that binds are preferable to literals, so in cases where histograms might be helpful you have to take great care and test, test, test.
Identifying histogram candidate columns

• First and foremost, it does no good, and increases overhead, to create histograms on columns whose data distribution is not skewed

• So don’t create histograms indiscriminately!
Identifying histogram candidate columns, continued

• DBMS_STATS provides a “SKEWONLY” option
• It is supposed to create histograms only on columns with skewed data distributions
• Unfortunately, it doesn’t work very well
• Much better to determine data skew directly to your own satisfaction
Identifying histogram candidate columns, continued

- There are various ways to do this
- My preference is to find a query that someone else has written and use that
- Who better to steal from than Tom Kyte?
- The query on asktom.oracle.com produces a nice graphic representation of skew
Identifying histogram candidate columns, continued

```sql
select wb, cnt,
    to_char(round( 100*cnt/(max(cnt) over ()),2),
    '999.00') rat,
    rpad( '*', 40*cnt/(max(cnt) over ()), '*' ) hist
from (select wb, count(*) cnt
    from (select width_bucket( r, 0,
        (select   count(distinct &cname)
            from &tname)+1,255) wb
    from (select dense_rank() over (order by &cname) r
        from &tname)
    group by wb)
order by wb
/
```

Substitution variables tname and cname are table and column names.
## Candidate query results

<table>
<thead>
<tr>
<th>WB</th>
<th>CNT</th>
<th>RAT</th>
<th>HIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3033</td>
<td>.01</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>49</td>
<td>.00</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>23</td>
<td>.00</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>27</td>
<td>.00</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>515</td>
<td>.00</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>684415</td>
<td>1.37</td>
<td>*</td>
</tr>
<tr>
<td>7</td>
<td>15270</td>
<td>.03</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>15757</td>
<td>.03</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>43919</td>
<td>.09</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>419413</td>
<td>.84</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>9512</td>
<td>.02</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>9479</td>
<td>.02</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>4498</td>
<td>.01</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>166645</td>
<td>.33</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>2809</td>
<td>.01</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>19315</td>
<td>.04</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>9786</td>
<td>.02</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>65466</td>
<td>.13</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1775111</td>
<td>3.55</td>
<td>*</td>
</tr>
<tr>
<td>20</td>
<td>598581</td>
<td>1.20</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>31389</td>
<td>.06</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>41223543</td>
<td>82.51</td>
<td>*</td>
</tr>
<tr>
<td>23</td>
<td>37406226</td>
<td>74.87</td>
<td>*</td>
</tr>
<tr>
<td>24</td>
<td>57905</td>
<td>.12</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>14030</td>
<td>.03</td>
<td></td>
</tr>
</tbody>
</table>
Candidate query results

- The previous slide shows that a couple of values are quite popular, making the column a good candidate for a histogram.
- Remember to test!
Creating histograms

• Generate for a single column

Creates a histogram

Owner

Table name

Column name

Number of buckets

Where to store current statistics

```
dbms_stats.gather_table_stats('XXX', 'XXX_EXECUTION',
method_opt=>'FOR COLUMNS CUST_ACCT_ID SIZE 254',
stattab=>'MYSTATS',
statown=>user, statid=>'NoHisto');
```
Buckets and histogram types

- A histogram can have from 1 to 254 buckets.
- The default level of column statistics is equivalent to a histogram with 1 bucket.
- If the number of distinct values is \( \leq 254 \), Oracle creates a frequency histogram.
- Each “bucket” consists of a column value and the number of times it occurs.
Buckets and histogram types continued

- If number of distinct values > 254, Oracle creates a *height-balanced histogram*
- Each bucket represents \( \text{ceil}(\text{row_count}/\text{num_buckets}) \) rows, and the high value (end point) for that bucket is stored
- Popular values are identified by the fact that they appear as end points in more than one bucket
- Unpopular values can be completely obscured
Loading and unloading statistics

• For testing, important to be able to switch between using and not using histograms

• Histograms are expensive to calculate, so we save them in a user stats table

```
dbms_stats.export_column_stats('XXX','XXX_EXECUTION',-
    colname=>'CUST_ACCT_ID', stattab=>'MYSTATS',-
    statid=>'Histo', statown=>user)
```
Loading and unloading statistics

• To prepare for test using alternate set of statistics:

```sql
dbms_stats.delete_column_stats('XXX','XXX_EXECUTION',
colname=>'CUST_ACCT_ID')
```

```sql
dbms_stats.import_column_stats('XXX','XXX_EXECUTION',
colname=>'CUST_ACCT_ID', stattab=>'MYSTATS',
statid=>'NoHisto',statown=>user)
```
Examining optimizer decisions

• Create a trace file showing optimizer plan consideration

alter session set events
'10053 trace name context forever, level 1';