One Pass Distinct Sampling

Amit Poddar

http://www.oraclegeek.net
# Object Statistics

## Table Statistics
- Number of rows (user_tab_statistics.num_rows)
- Blocks (user_tab_statistics.blocks)
- Average row length (user_tab_statistics.avg_row_len)

## Column Statistics
- Number of nulls (user_tab_col_statistics.num_nulls)
- Low/High value (user_tab_col_statistics.low/high_value)
- Number of distinct values (NDV) (user_tab_col_statistics.num_distinct)

## Index Statistics
- Leaf blocks (user_ind_statistics.leaf_blocks)
- Distinct keys (user_ind_statistics.distinct_keys)
- Clustering factor (user_ind_statistics.clustering_factor)
Inaccurate object statistics

- Non representative object statistics leads to
- Poor cardinality estimates which leads to
- Poor access path selection which leads to
- Poor join method selection which leads to
- Poor join order selection which leads to
- Poor SQL execution times which leads to
- Poor system performance
**NDV**

- Number of distinct values in a column (excluding nulls).
- Used to derive table and join cardinalities for equality predicates and equijoins (in absence of histograms).
- Probably the most critical statistic for the query optimizer.
- Deriving it accurately is a well researched but quite a challenging problem.
Gathering Table/Column Statistics

- Using procedures in dbms_stats package
- Table t1 has 285,888,512 rows with 68,000 distinct values in n1 and n2 each.
- dbms_stats.gather_table_stats
  
  (  ownname => USER,
      tabname  => 'T1',
      estimate_percent => 100,
      method_opt => 'for all columns size 1,
      cascade => false
  );
select  
  count(*) nr ,
  count(n1) n1nr ,
  count(different n1) n1ndv ,
  sum(sys_op_opnsize(n1)) n1sz ,
  substrb(dump(min(n1),16,0,32),1,120) n1low ,
  substrb(dump(max(n1),16,0,32),1, 120) n1high ,
  count(n2) n2nr ,
  count(different n2) n2ndv ,
  sum(sys_op_opnsize(n2)) n2sz ,
  substrb(dump(min(n2),16,0,32),1,120) n2low ,
  substrb(dump(max(n2),16,0,32),1, 120) n2high  
from t1 t

<table>
<thead>
<tr>
<th>num_rows</th>
<th>nr</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_nulls(n1,n2)</td>
<td>(nr – n1nr), (nr – n2nr)</td>
</tr>
<tr>
<td>ndv (n1,n2)</td>
<td>n1ndv, n2ndv</td>
</tr>
<tr>
<td>high_value(n1,n2)</td>
<td>n1high, n2high</td>
</tr>
<tr>
<td>low_value(n1,n2)</td>
<td>n1low, n2low</td>
</tr>
<tr>
<td>avg_col_len(n1,n2)</td>
<td>ceil(n1sz/n1nr) + 1, ceil(n2sz/n2nr) + 1</td>
</tr>
</tbody>
</table>
## Performance Statistics

### SQL without NDV aggregation

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL execute elapsed time</td>
<td>4.02 mins</td>
</tr>
<tr>
<td><strong>DB CPU</strong></td>
<td>2.33 mins</td>
</tr>
<tr>
<td>session logical reads</td>
<td>620,118</td>
</tr>
<tr>
<td>table scans (long)</td>
<td>1</td>
</tr>
<tr>
<td>session uga memory</td>
<td>2.01 MB</td>
</tr>
<tr>
<td>session uga memory(max)</td>
<td>2.01 MB</td>
</tr>
<tr>
<td>session pga memory</td>
<td>2.13 MB</td>
</tr>
<tr>
<td>session pga memory(max)</td>
<td>2.13 MB</td>
</tr>
<tr>
<td>sorts (rows)</td>
<td>0</td>
</tr>
<tr>
<td>workarea executions – optimal</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL with NDV aggregation

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL execute elapsed time</td>
<td>16.33 mins</td>
</tr>
<tr>
<td><strong>DB CPU</strong></td>
<td>14.25 mins</td>
</tr>
<tr>
<td>session logical reads</td>
<td>620,118</td>
</tr>
<tr>
<td>table scans (long)</td>
<td>1</td>
</tr>
<tr>
<td>session uga memory</td>
<td>5.74 MB</td>
</tr>
<tr>
<td>session uga memory(max)</td>
<td>5.74 MB</td>
</tr>
<tr>
<td>session pga memory</td>
<td>5.75 MB</td>
</tr>
<tr>
<td>session pga memory(max)</td>
<td>5.75 MB</td>
</tr>
<tr>
<td>sorts (rows)</td>
<td>571,768,832</td>
</tr>
<tr>
<td>workarea executions – optimal</td>
<td>1</td>
</tr>
</tbody>
</table>
Sampling

- Resource consumption by statistics gathering query increases exponentially with increase in table size.
- This increase primarily results from oracle having to sort increasing number of rows to derive NDV.
- To overcome this problem Oracle provides an option to gather statistics on smaller data set, and scale it up to represent the entire set.
- This smaller data set is obtained by statistically sampling the table.
Sampling

• **Row Sampling**
  - Row sampling reads rows without regard to their physical placement on disk. This provides the most random data for estimates, but it can result in reading more data than necessary. For example, a row sample might select one row from each block, requiring a full scan of the table or index.

• **Block Sampling**
  - Block sampling reads a random sample of blocks and uses all of the rows in those blocks for estimates. This reduces the amount of I/O activity for a given sample size, but it can reduce the randomness of the sample if rows are not randomly distributed on disk. This can significantly affect the quality of the estimate of number of distinct values.
Row Sampling

Each row has a probability $\rho$ of making into the sample where $\rho = \frac{\text{Sampling Percent}}{100}$.

Sample size is normally distributed with mean ($\mu$) = $N \times \rho$ and variance ($\sigma^2$) = $N \times \rho \times (1 - \rho)$.

Scans the entire table for all practical purposes.

Assumes uniform distribution of distinct values i.e. each distinct value has the same cardinality.

In a non-uniform distribution, values with higher cardinality have a higher probability of making into the sample which makes it much more difficult to get a representative sample of all the distinct values.

Accuracy of NDV derived is good with bounded variance for uniform and close to uniform distributions. But is no guarantee of accuracy of the derived NDV in cases of non-uniform distribution.

$\mu - 2\sigma < S_n < \mu + 2\sigma$ (95% of all values)

$f = 1 - (1 - p)^m$

$p =$ Sampling percent

$m =$ rows/block

$\sigma = \sqrt{N \times p \times (p - 1)}$

Fig 5.sql
Row sampling

```sql
select count(*) nr ,
count(n1) n1nr ,
count(distinct n1) n1ndv ,
sum(sys_op_opnsize(n1)) n1sz ,
substrb(dump(min(n1),16,0,32),1,120) n1low ,
substrb(dump(max(n1),16,0,32),1,120) n1high ,
count(n2) n2nr ,
count(distinct n2) n2ndv ,
sum(sys_op_opnsize(n2)) n2sz ,
substrb(dump(min(n2),16,0,32),1,120) n2low ,
substrb(dump(max(n2),16,0,32),1,120) n2high
from t1 t sample ( 1.00000000000 )
```
Block/Page Sampling

- Each block has a probability $\rho$ of making into the sample where $\rho = \text{Sampling Percent}/100$.
- Scans a fraction of the table.
- Assumes uniform distribution of rows amongst all blocks and uniform distribution of distinct values i.e. all blocks are statistically same and they have statistically similar data.
- For column distributions where the assumptions do not hold true, the accuracy rate of NDV derived is quite poor with unbounded variance.
- This is because block sampling deals with sets of rows, so anything that has potential to cause small errors in row sampling will result in larger errors in block sampling.
- Accuracy rate is very poor for most of the practical column distributions. Therefore oracle uses row sampling by default.
select count(*) nr , count(n1) n1nr , count(distinct n1) n1ndv , sum(sys_op_opnsize(n1)) n1sz , substrb(dump(min(n1),16,0,32),1,120) n1low , substrb(dump(max(n1),16,0,32),1,120) n1high , count(n2) n2nr , count(distinct n2) n2ndv , sum(sys_op_opnsize(n2)) n2sz , substrb(dump(min(n2),16,0,32),1,120) n2low , substrb(dump(max(n2),16,0,32),1,120) n2high
from t1 t sample block( 1.00000000000 )
Accuracy Numbers

Table t101 with 1000,000 rows

create table t101
as
with milli_row as ( 
    select /*+ materialize */
        rownum
    from all_objects
    where rownum <= 1000
)
select mod(rownum-1, 50000) scattered,
    trunc((rownum-1)/20) clustered,
    trunc(dbms_random.value(0,50000)) uniform,
    trunc(7000 * dbms_random.normal) normal,
    trunc(7000 * exp(dbms_random.normal)) lognormal
from
    milli_row m1,
    milli_row m2
where
    rownum <= 1000000
;
## Accuracy Numbers (t101 1000,000 rows)

Accuracy = 1 – (abs(Estimated NDV – Actual NDV) / Actual NDV )

<table>
<thead>
<tr>
<th>%</th>
<th>Scattered</th>
<th></th>
<th>Clumped</th>
<th></th>
<th>Uniform</th>
<th></th>
<th>Normal</th>
<th></th>
<th>Lognormal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Row</td>
<td>Block</td>
<td>Row</td>
<td>Block</td>
<td>Row</td>
<td>Block</td>
<td>Row</td>
<td>Block</td>
<td>Row</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.98</td>
<td>1</td>
<td>0.2</td>
<td>0.98</td>
<td>1</td>
<td>0.71</td>
<td>0.8</td>
<td>0.46</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>0.96</td>
<td>1</td>
<td>0.15</td>
<td>0.99</td>
<td>0.98</td>
<td>0.8</td>
<td>0.75</td>
<td>0.59</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.26</td>
<td>1</td>
<td>1</td>
<td>0.86</td>
<td>0.84</td>
<td>0.67</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.3</td>
<td>1</td>
<td>1</td>
<td>0.89</td>
<td>0.85</td>
<td>0.75</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.52</td>
<td>1</td>
<td>1</td>
<td>0.92</td>
<td>0.93</td>
<td>0.8</td>
</tr>
<tr>
<td>60</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.46</td>
<td>1</td>
<td>1</td>
<td>0.94</td>
<td>0.91</td>
<td>0.85</td>
</tr>
<tr>
<td>70</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.72</td>
<td>1</td>
<td>1</td>
<td>0.96</td>
<td>0.96</td>
<td>0.9</td>
</tr>
<tr>
<td>80</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.81</td>
<td>1</td>
<td>1</td>
<td>0.98</td>
<td>0.98</td>
<td>0.93</td>
</tr>
<tr>
<td>90</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.87</td>
<td>1</td>
<td>1</td>
<td>0.99</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Accuracy Numbers

Table t109 with 300,000,000 rows

create table t 109
as
with milli_row as (  
    select /*+ materialize */
        rownum
    from all_objects
    where rownum <= 1000
)
select mod(rownum-1, 50000000) scattered,
    trunc((rownum-1)/20) clustered,
    trunc(dbms_random.value(0,50000000)) uniform,
    trunc(7000 * dbms_random.normal) normal,
    trunc(7000 * exp(dbms_random.normal)) lognormal
from
    milli_row m1,
    milli_row m2,
    milli_row m3
where
    rownum <= 300000000
/

<table>
<thead>
<tr>
<th>%</th>
<th>Scattered</th>
<th></th>
<th>Clustered</th>
<th></th>
<th>Uniform</th>
<th></th>
<th>Normal</th>
<th></th>
<th>Lognormal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Row</td>
<td>Block</td>
<td>Row</td>
<td>Block</td>
<td>Row</td>
<td>Block</td>
<td>Row</td>
<td>Block</td>
<td>Row</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.77</td>
<td>1</td>
<td>0.01</td>
<td>0.86</td>
<td>0.85</td>
<td>0.74</td>
<td>0.74</td>
<td>0.33</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.89</td>
<td>1</td>
<td>0.02</td>
<td>0.87</td>
<td>0.88</td>
<td>0.79</td>
<td>0.79</td>
<td>0.4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.68</td>
<td>1</td>
<td>0.03</td>
<td>0.87</td>
<td>0.87</td>
<td>0.81</td>
<td>0.81</td>
<td>0.44</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.95</td>
<td>1</td>
<td>0.04</td>
<td>0.87</td>
<td>0.87</td>
<td>0.83</td>
<td>0.83</td>
<td>0.48</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.96</td>
<td>1</td>
<td>0.05</td>
<td>0.87</td>
<td>0.88</td>
<td>0.84</td>
<td>0.85</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0.98</td>
<td>1</td>
<td>0.06</td>
<td>0.87</td>
<td>0.88</td>
<td>0.85</td>
<td>0.86</td>
<td>0.53</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0.93</td>
<td>1</td>
<td>0.07</td>
<td>0.88</td>
<td>0.88</td>
<td>0.86</td>
<td>0.86</td>
<td>0.55</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0.95</td>
<td>1</td>
<td>0.08</td>
<td>0.88</td>
<td>0.87</td>
<td>0.87</td>
<td>0.87</td>
<td>0.56</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0.94</td>
<td>1</td>
<td>0.09</td>
<td>0.88</td>
<td>0.88</td>
<td>0.87</td>
<td>0.87</td>
<td>0.58</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.98</td>
<td>1</td>
<td>0.1</td>
<td>0.88</td>
<td>0.89</td>
<td>0.88</td>
<td>0.88</td>
<td>0.6</td>
</tr>
</tbody>
</table>
Row Sampling drawbacks

- Randomness in sampling results in randomness in sample sizes. This can result in over/under estimation of NDV. The error due to this problem is generally very low in row sampling, but can be significant in case of block sampling.
- Accuracy and variance of derived NDV is highly dependent on the data distribution in the column.
- Different kinds of data distribution may need different sample sizes. Maintaining different sample sizes for different tables (may be for different columns in the same table) can be a maintenance nightmare for the DBA.
- Row Sampling treats all the rows the same. That means every row has an equal chance of making it to the sample which is good for calculating number of rows in the table but it can be very inaccurate for calculating number of distinct values in a column.
Inaccurate NDV workarounds

- Hints
- Outlines (9i and 10g)
- SQL Profiles (10g)
- `dbms_stats.set_column_stats`
- Dynamic Sampling (9i and 10g)
- Adaptive sampling by `dbms_stats` (9i and 10g)
Distinct Sampling

- Adaptive sampling does improve the accuracy in some cases, but it also result in multiple scans of the table with larger sample sizes.
- Larger sample size would mean sorting larger number of rows, which sampling was suppose to avoid in the first place.
- Adaptive sampling still has the drawback of being non-deterministic i.e. multiple runs can result in different values for the NDV.
- Other workarounds mentioned, do not solve all the problems. They have to applied on a case by case basis.
- Ideally a new sampling algorithm is needed to calculate NDV, an algorithm that provides each distinct value equal probability for making into the sample, does not run a sort distinct operation over all the rows and also allows oracle to use the full data set to calculate other statistics.
Distinct Sampling

- Oracle 11g introduced a new algorithm, that derives NDV by sampling distinct values in a column without sorting the entire data set and uses the full data set in the process. This provides each distinct value equal probability of making into the sample. And since it uses the full data set, the other statistics are calculated precisely.

- This algorithm is termed as approximate NDV algorithm in Oracle’s SIGMOD presentations.

- This algorithm is also termed as synopsis based algorithm, since sample of each column is termed as a synopsis.
Column values are mapped to a 64 bit hash value in the domain shown above using a uniform hash function.

This mapping is done in such a way that, the values are distributed uniformly across the domain i.e. two equal sized portion of the domain contains same number of distinct values.

With this distribution oracle can count the number of distinct values in half of the domain and multiply the result by two to derive the NDV or count the NDV in one fourth of the domain and multiply the result by four to derive the NDV and so on.
Column synopsis

- At the start of the algorithm a memory area is allocated for each column in the table. This memory area is allocated in the PGA.
- While scanning the table each column value is mapped to a 64 bit hash value. This hash value is stored in the memory only if it does not exist already.
- At the end of the scan, the number of hash values in this memory area is the number of distinct values in the full selected domain which is same as number of distinct values in the column.
- This memory area is called column synopsis. The memory consumption by each synopsis is bounded at storing only $16384$ hash values. Every time the synopsis reaches its limit oracle splits the selected domain into two parts and discards all the values from lower half of the domain.
Domain splitting means that the algorithm will count the number of distinct hash values mapped to only half of the domain and then multiply the result by two to derive NDV.

Achieved by

- Discarding all the hash values in synopsis with first bit as zero since this is the first split.
- From here on only the hash values that belong to the domain (i.e. hash values with first bit as one) will be allowed in the synopsis.
First Split Example

`qesandvSplit`: split 0 to `lvl`: 1  `set`:16384

- pos: 0: 0000000000000000101010100110101001110000010011100100001101000110
+ pos: 1: 100000000000000000111011101010001011110100000100100101101111011
- pos: 2: 00000000000000001101101010110010101001100010010100110011010001101111
+ pos: 3: 1100000000000001001010000110011010101010110111001100110100110011000111
- pos: 4: 001000000000000011101011111111001010000110011001001011001100110000111
+ pos: 5: 10100000000000001011111111001001100010011010111110011011011000111
- pos: 6: 0110000000000000110110110100100000000110110110110000010011101000
- pos: 7: 0010000000000000111001001111011000000001101100100101011110000001
- pos: 8: 01100000000000000110101111111011110100100000001101101101001100110100100
+ pos: 9: 1001000000000000110100100111101011011011000000100100100
- pos: 10: 011010111111111101011001101111000000001101101011111111010100100001
+ pos: 11: 11010000000000000110010000001111001001100111100110110110100110001
- pos: 12: 00110000000000111011111101100100000001111001100100100010001001101011
- pos: 13: 00000111111111000110011010011110011001100001100110001111110001111000
Domain splitting

- If the synopsis reaches its maximum size again the domain is split again in two parts and lower half is discarded. This time values with second bit as zero are discarded. This process continues till all the column values are consumed.
- At the end of the process, the synopsis contains distinct hash values and number of times it was split (split level $d$). All the distinct hash values have their leading $d$ bits set to one.
- NDV for column $= (\text{Distinct hash values in synopsis}) \times 2^d$
Second Split Example

qesandvSplit: split 0 to lvl: 2 set: 16384

+ pos: 0: 11111111111111101011001100100100001001011110110000101100001101
- pos: 1: 10000000000000000111011110110100010111101000001001001011111101
- pos: 2: 1001000111111110001010011011010110101001111100001001011111101
+ pos: 3: 110000000000001001010000110011010101011100111010110001101
+ pos: 4: 110000000000001000110100011011010111111010001110100101111101
- pos: 5: 1010000000000001001000011001101010101011011100111001101011011
- pos: 6: 11000000000000100101000011100001010001010110001010010001101
+ pos: 7: 110000000000001000110010000011100011010001101101110110100011101
+ pos: 8: 1100000000000010010101100010101100000101001000110010001101
- pos: 9: 1001000000000001001000011001101010101011011100111001101011011
- pos: 10: 1001000000000001001000011001101010101111100001011001111010111
+ pos: 11: 11010000000000010010000111110001010110001110010111010100100100
- pos: 12: 1001000000000001111111110111111000101001001111001111010010001010110
- pos: 13: 101100000000000101001000011011101010110011000001011000010100100010
<table>
<thead>
<tr>
<th>Row</th>
<th>Hash Value</th>
<th>Synopsis</th>
<th>Splits</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>00000</td>
<td>{00000}</td>
<td>0</td>
<td>0-31 (32)</td>
</tr>
<tr>
<td>R2</td>
<td>00011</td>
<td>{00000,00011}</td>
<td>0</td>
<td>0-31(32)</td>
</tr>
<tr>
<td>R3</td>
<td>00110</td>
<td>{00000,00011,00110}</td>
<td>0</td>
<td>0-31(32)</td>
</tr>
<tr>
<td>R4</td>
<td>01001</td>
<td>{}</td>
<td>1</td>
<td>16-31(16)</td>
</tr>
<tr>
<td>R5</td>
<td>01001</td>
<td>{}</td>
<td>1</td>
<td>16-31(16)</td>
</tr>
<tr>
<td>R6</td>
<td>01100</td>
<td>{}</td>
<td>1</td>
<td>16-31(16)</td>
</tr>
<tr>
<td>R7</td>
<td>01111</td>
<td>{}</td>
<td>1</td>
<td>16-31(16)</td>
</tr>
<tr>
<td>R8</td>
<td>10010</td>
<td>{10010}</td>
<td>1</td>
<td>16-31(16)</td>
</tr>
<tr>
<td>R9</td>
<td>10101</td>
<td>{10010,10101}</td>
<td>1</td>
<td>16-31(16)</td>
</tr>
<tr>
<td>R10</td>
<td>11000</td>
<td>{10010,10101,11000}</td>
<td>1</td>
<td>16-31(16)</td>
</tr>
<tr>
<td>R11</td>
<td>11011</td>
<td>{11000,11011}</td>
<td>2</td>
<td>24-31(16)</td>
</tr>
<tr>
<td>R12</td>
<td>11110</td>
<td>{11000,11011,11110}</td>
<td>2</td>
<td>24-31(8)</td>
</tr>
</tbody>
</table>
Accuracy Numbers (t109 300,000,000 rows)
Accuracy = 1 – (abs(Estimated NDV – Actual NDV) / Actual NDV )

<table>
<thead>
<tr>
<th>%</th>
<th>Scattered</th>
<th>Clustered</th>
<th>Uniform</th>
<th>Normal</th>
<th>Lognormal</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0.88</td>
<td>0.88</td>
<td>0.6</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>1</td>
<td>0.91</td>
<td>0.92</td>
<td>0.7</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>1</td>
<td>0.93</td>
<td>0.94</td>
<td>0.77</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>1</td>
<td>0.95</td>
<td>0.95</td>
<td>0.82</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>1</td>
<td>0.97</td>
<td>0.96</td>
<td>0.86</td>
</tr>
<tr>
<td>60</td>
<td>1</td>
<td>1</td>
<td>0.98</td>
<td>0.98</td>
<td>0.9</td>
</tr>
<tr>
<td>70</td>
<td>1</td>
<td>1</td>
<td>0.99</td>
<td>0.98</td>
<td>0.93</td>
</tr>
<tr>
<td>80</td>
<td>1</td>
<td>1</td>
<td>0.99</td>
<td>0.99</td>
<td>0.95</td>
</tr>
<tr>
<td>90</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.99</td>
<td>0.98</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Appr. NDV</td>
<td>1</td>
<td>0.99</td>
<td>1</td>
<td>0.99</td>
<td>0.98</td>
</tr>
</tbody>
</table>
## Timing Details (t109 300,000,000 rows)

### Time taken (in minutes)

<table>
<thead>
<tr>
<th>%</th>
<th>Scattered</th>
<th>Clustered</th>
<th>Uniform</th>
<th>Normal</th>
<th>Lognormal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>CPU</td>
<td>Total</td>
<td>CPU</td>
<td>Total</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>9</td>
<td>16</td>
<td>9</td>
<td>16</td>
</tr>
<tr>
<td>20</td>
<td>26</td>
<td>17</td>
<td>26</td>
<td>17</td>
<td>26</td>
</tr>
<tr>
<td>30</td>
<td>37</td>
<td>25</td>
<td>37</td>
<td>25</td>
<td>37</td>
</tr>
<tr>
<td>40</td>
<td>48</td>
<td>33</td>
<td>48</td>
<td>33</td>
<td>48</td>
</tr>
<tr>
<td>50</td>
<td>59</td>
<td>41</td>
<td>59</td>
<td>41</td>
<td>59</td>
</tr>
<tr>
<td>60</td>
<td>90</td>
<td>55</td>
<td>90</td>
<td>55</td>
<td>90</td>
</tr>
<tr>
<td>70</td>
<td>79</td>
<td>57</td>
<td>79</td>
<td>57</td>
<td>79</td>
</tr>
<tr>
<td>80</td>
<td>90</td>
<td>65</td>
<td>90</td>
<td>65</td>
<td>90</td>
</tr>
<tr>
<td>90</td>
<td>101</td>
<td>73</td>
<td>101</td>
<td>73</td>
<td>101</td>
</tr>
<tr>
<td>100</td>
<td>108</td>
<td>79</td>
<td>108</td>
<td>79</td>
<td>108</td>
</tr>
<tr>
<td><strong>Approximate NDV</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Approximate NDV

<table>
<thead>
<tr>
<th></th>
<th>Scattered</th>
<th>Clustered</th>
<th>Uniform</th>
<th>Normal</th>
<th>Lognormal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>4</td>
<td>9</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>
## Performance characteristics

<table>
<thead>
<tr>
<th>Sort statistics</th>
<th>Sampling</th>
<th>Approximate NDV</th>
</tr>
</thead>
<tbody>
<tr>
<td>sorts (rows)</td>
<td>0.1%</td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td>1%</td>
<td>10%</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>1,503,069</td>
<td>15,010,448</td>
<td>149,991,577</td>
</tr>
<tr>
<td>15,000,044,577</td>
<td>4,890</td>
<td></td>
</tr>
<tr>
<td>session pga memory</td>
<td>638,332</td>
<td>507,260</td>
</tr>
<tr>
<td></td>
<td>572,796</td>
<td>638,332</td>
</tr>
<tr>
<td>4,980,736</td>
<td>4,980,736</td>
<td></td>
</tr>
<tr>
<td>session pga memory max</td>
<td>32,423,292</td>
<td>110,017,916</td>
</tr>
<tr>
<td></td>
<td>110,083,452</td>
<td>110,083,452</td>
</tr>
<tr>
<td>6,422,528</td>
<td>6,422,528</td>
<td></td>
</tr>
<tr>
<td>workarea executions - optimal</td>
<td>106</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>134</td>
<td></td>
<td></td>
</tr>
<tr>
<td>workarea executions – onepass</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Sampling</td>
<td>Approximate NDV</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>0.1%</td>
<td>1%</td>
</tr>
<tr>
<td>DB time</td>
<td>00:05:19</td>
<td>00:06:05</td>
</tr>
<tr>
<td>sql execute elapsed time</td>
<td>00:05:01</td>
<td>00:05:47</td>
</tr>
<tr>
<td>DB CPU</td>
<td>00:00:28</td>
<td>00:01:21</td>
</tr>
<tr>
<td><strong>Logical read statistics</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consistent gets direct</td>
<td>383,524</td>
<td>1,279,320</td>
</tr>
<tr>
<td>consistent gets from cache</td>
<td>6,985</td>
<td>6,954</td>
</tr>
<tr>
<td>consistent gets</td>
<td>390,509</td>
<td>1,286,274</td>
</tr>
<tr>
<td>db block gets</td>
<td>5,748</td>
<td>5,770</td>
</tr>
<tr>
<td>session logical reads</td>
<td>396,257</td>
<td>1,292,044</td>
</tr>
<tr>
<td>buffer is not pinned count</td>
<td>388,189</td>
<td>1,283,963</td>
</tr>
<tr>
<td>buffer is pinned count</td>
<td>2,187</td>
<td>2,187</td>
</tr>
<tr>
<td></td>
<td>Sampling</td>
<td>Approximate NDV</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>0.1%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td><strong>Physical read statistics</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>physical reads</td>
<td>1,322,416</td>
<td>1,322,697</td>
</tr>
<tr>
<td>physical reads direct</td>
<td>1,321,850</td>
<td>0</td>
</tr>
<tr>
<td>physical reads direct temporary tablespace</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>physical writes</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>physical writes direct</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>physical writes direct temporary tablespace</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Table scan statistics</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>table scans (long tables)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>table fetch by rowid</td>
<td>1,963</td>
<td>3,498</td>
</tr>
</tbody>
</table>
## Comparison

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Approximate NDV</th>
<th>Sampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution independence</td>
<td>✅</td>
<td>x</td>
</tr>
<tr>
<td>Low and bounded variance/Repeatability</td>
<td>✅</td>
<td>x</td>
</tr>
<tr>
<td>Low resource consumption</td>
<td>✅</td>
<td>x</td>
</tr>
<tr>
<td>Bounded and stable resource consumption</td>
<td>✅</td>
<td>x</td>
</tr>
<tr>
<td>Bounded and stable elapsed time</td>
<td>✅</td>
<td>x</td>
</tr>
</tbody>
</table>
Implementation

- Invoked by dbms_stats when two conditions are satisfied
  - Parameter APPROXIMATE_NDV is set to true (Default)
    - `exec dbms_stats.set_param('APPROXIMATE_NDV','TRUE');`
  - `DBMS_STATS.AUTO_SAMPLE_SIZE` is used as `estimate_percent` (Default)
    - `exec dbms_stats.set_param('ESTIMATE_PERCENT', 'DBMS_STATS.AUTO_SAMPLE_SIZE');`
Implementation

dbms_stats.gather_table_stats
    (ownname => 'AP349',
     tabname  => 'T101',
     estimate_percent=>dbms_stats.auto_sample_size,
     cascase  => false,
     method_opt=>’for all columns size 1’
    );
Implementation

- `dbms_sqltune_internal.gather_sql_stats`

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Argument Type</th>
<th>IN/OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_TEXT</td>
<td>CLOB</td>
<td>IN</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>VARCHAR2</td>
<td>IN</td>
</tr>
<tr>
<td>BIND_LIST</td>
<td>SQL_BINDS</td>
<td>IN</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>VARCHAR2</td>
<td>IN</td>
</tr>
<tr>
<td>REPT</td>
<td>XMLTYPE</td>
<td>IN/OUT</td>
</tr>
<tr>
<td>ERR_CODE</td>
<td>NUMBER</td>
<td>OUT</td>
</tr>
<tr>
<td>ERR_MESG</td>
<td>VARCHAR2</td>
<td>OUT</td>
</tr>
</tbody>
</table>
select to_char(count("SCATTERED")),
    to_char(substrb(dump(min("SCATTERED"),16,0,32),1,120)),
    to_char(substrb(dump(max("SCATTERED"),16,0,32),1,120)),
    to_char(count("CLUSTERED")),
    to_char(substrb(dump(min("CLUSTERED"),16,0,32),1,120)),
    to_char(substrb(dump(max("CLUSTERED"),16,0,32),1,120)),
    to_char(count("UNIFORM")),
    to_char(substrb(dump(min("UNIFORM"),16,0,32),1,120)),
    to_char(substrb(dump(max("UNIFORM"),16,0,32),1,120)),
    to_char(count("NORMAL")),
    to_char(substrb(dump(min("NORMAL"),16,0,32),1,120)),
    to_char(substrb(dump(max("NORMAL"),16,0,32),1,120)),
    to_char(count("LOGNORMAL")),
    to_char(substrb(dump(min("LOGNORMAL"),16,0,32),1,120)),
    to_char(substrb(dump(max("LOGNORMAL"),16,0,32),1,120))
from T101 t
USER_NAME AND BIND_LIST

- **USER_NAME**
  - Oracle user that should be used to run the query. For this example the user should be AP349, the owner of the table T109.

- **BIND_LIST**
  - `VARRAY(2000) OF ANYDATA`
  - Values for all bind values to be bound to the query.
  - For gathering statistics there is no predicate on the query, so this is supplied as NULL in this example.
  - Can be used to derive approximate NDV for queries that would otherwise would have to sort lot of rows.
Options

- Comma separated list of values. There is one value for each selection. In this example there are fifteen values in the option string.
  - NDV,NIL,NIL,NDV,NIL,NIL,NDV,NIL,NIL,NDV,NIL,NIL,NDV
  - NIL,NIL

- Each value decides what statistics are calculated for the corresponding column by dbms_sqltune_internal.gather_sql_stats.

- Observed values for different option values are NDV, NIL, ACL, SYN and NNV
Options (SYN)

- Following statistics are calculated for columns corresponding to option SYN
  - Value of the selection (count, min, max)
  - Number of distinct values using the APPROXIMATE NDV algorithm.
    - NDV for the column
  - Number of not null values
    - Used to calculate num_nulls, average column length and average row length
  - Total column size
    - Used to calculate average column length and average row length
  - Total number of rows if this is the first selection in the query.
Options (NDV)

- Following statistics are calculated for columns corresponding to option NDV
  - Value of the selection (count, max, min)
  - Number of distinct values using the APPROXIMATE NDV algorithm.
    - NDV for the column
  - Number of not null values
    - Used to calculate num_nulls, average column length and average row length.
  - Total column size
    - Used to calculate average_column_length and average_row_length
  - Total number of rows if this is the first selection in the query.
Options (ACL)

- Used when statistics is calculated for only some columns, but Oracle needs column lengths and number of not null values for all columns to derive average row length for the table.
- Following statistics are calculated for columns corresponding to option ACL
  - Value of the selection (count, max, min)
  - Number of not null values
    - Used to calculate average_row_length
  - Total column size
    - Used to calculate average_row_length
  - Total number of rows if this is the first selection in the query.
Options (NNV)

- Used when statistics is calculated for only some columns, but Oracle needs column lengths and number of not null values for all columns to derive average row length for the table.
- This is same as ACL but for data and timestamp columns Oracle already knows the average column length.
- Following statistics are calculated for columns corresponding to option NNV
  - Value of the selection (count, max, min)
  - Number of not null values
    - Used to calculate num_nulls for the column
  - Total number of rows if this is the first selection in the query.
Options (NIL)

- Used usually with min and max aggregates on columns that already has a NDV/SYN option associated with another aggregation.
- Only the value of the aggregation is calculated. No other statistics is calculated for column corresponding to this option.
- Used only with aggregations to find min and max value for a column.
REPT

- XMLTYPE IN/OUT variable
- XML report returned from `dbms_sqltune_internal.gather_sql_stats`.
- Contains all the statistics requested.
- Contains one XML fragment for every selection.
- Type of statistics in the XML fragment depends on the corresponding option value.
REPT (SYN)

<select_list_item>
  <pos>0</pos>
  <value>759954</value>
  <rowcnt>1000000</rowcnt>
  <split>2</split>
  <ndv>50536</ndv>
  <nonnulls>1000000</nonnulls>
  <rsize>3787920</rsize>
  <hash_val>…,…,…</hash_val>
</select_list_item>
REPT (NDV)

<select_list_item>
  <pos>0</pos>
  <value>759954</value>
  <rowcnt>1000000</rowcnt>
  <split>2</split>
  <ndv>50536</ndv>
  <nonnulls>1000000</nonnulls>
  <rsize>3787920</rsize>
</select_list_item>
<select_list_item>
  <pos>0</pos>
  <value>1000000</value>
  <rowcnt>1000000</rowcnt>
  <rsizer>3787920</rsizer>
  <nonnulls>1000000</nonnulls>
</select_list_item>
REPT (NNV)

<select_list_item>
    <pos>0</pos>
    <value>1000000</value>
    <rowcnt>1000000</rowcnt>
    <nonnulls>1000000</nonnulls>
</select_list_item>
REPT (NIL)

<select_list_item>
  <pos>0</pos>
  <value>1000000</value>
</select_list_item>
10832 trace file
Partition/Global Statistics

- Global statistics on partitioned tables necessary for queries that access multiple partitions.
- Global statistics on partitioned tables also necessary for queries where partition access information cannot be determined during parsing.
- All of the global statistics such as num_rows, num_nulls, avg_col_len, avg_row_len etc can be determined from partition statistics.
- Global NDV cannot be determined from partition statistics and require a separate scan of full table.
Partition/Global Statistics

- For table with m partitions
- Initial statistics gathering
  - `dbms_stats` has to scan 2m partitions to gather partition and global statistics.
- After significant change of data in n partitions
- Incremental maintenance
  - `Dbms_stats` has to scan n partitions to gather partition statistics and m partitions to gather global statistics.
- In both case oracle has to scan the full table for gathering global statistics, only because global NDV cannot be derived by partition statistics.
Synopses Aggregation

- Synopses generated for different parts of a table can be merged to generate a synopsis that statistically represents the entire table.

- This merge will be statistically correct when
  - All of the synopses were generated using the same hash function
  - They all represent the same part of the domain i.e. they all have been split the same number of times.

- Synopses representing a larger portion of the domain can be split multiple times so that they all represent the same portion of the domain.

- Partition synopses can be merged to derive global NDV.
1. Partition level stats are gathered & synopsis created

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

Sales Table

- May 18th 2008
- May 19th 2008
- May 20th 2008
- May 21st 2008
- May 22nd 2008
- May 23rd 2008

Global Statistic

Sysaux Tablespace
SELECT
  TO_CHAR(COUNT("L_ORDERKEY")),
  TO_CHAR(SUBSTRB(DUMP(MIN("L_ORDERKEY"),16,0,32),1,120)),
  TO_CHAR(SUBSTRB(DUMP(MAX("L_ORDERKEY"),16,0,32),1,120)),
  ...
  ...
  ...
  TO_CHAR(COUNT("L_COMMENT")),
  TO_CHAR(SUBSTRB(DUMP(MIN("L_COMMENT"),16,0,32),1,120)),
  TO_CHAR(SUBSTRB(DUMP(MAX("L_COMMENT"),16,0,32),1,120)),
  ...
  ...
  ...
  ...
WHERE TBL$OR$IDX$PART$NUM(LINEITEM,0,4,0,ROWID) = :objn
/* SYN,NIL,NIL .................................SYN,NIL,NIL*/
Partition synopsis

• Hash values are reversed and stored in decimal.

Hash value after one split:
10000000000000000011011011000000100100111001101101111011
=>11011110110100100100000101111010000101011101110000000000000001
=>16055967614137794561

• All the stored hash values of the synopsis with d splits will have trailing d bits as one.
• Hash values stored in two tables
  • WRI$_OPTSTAT_SYNOPSIS_HEAD$
  • WRI$_OPTSTAT_SYNOPSIS$
## Synopses storage

### $WRI\_OPTSTAT\_SYNOPSIS$

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPSIS#</td>
<td>Synopsis Number (Refers to $WRI_OPTSTAT_SYNOPSIS_HEAD$.SYNOPSIS#)</td>
</tr>
<tr>
<td>HASHVALUE</td>
<td>Hash values in the synopsis</td>
</tr>
</tbody>
</table>

### $WRI\_OPTSTAT\_SYNOPSIS\_HEAD$

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BO#</td>
<td>Object id for the partitioned table</td>
</tr>
<tr>
<td>GROUP#</td>
<td>2 * Object_id of the partition</td>
</tr>
<tr>
<td>INTCOL#</td>
<td>Column id for the column whose synopsis is represented by this row</td>
</tr>
<tr>
<td>SYNOPSIS#</td>
<td>Synopsis Id. The actual synopsis is store in $wri_optstat_synopsis$</td>
</tr>
<tr>
<td>SPLIT</td>
<td>Split level corresponding to this synopsis</td>
</tr>
<tr>
<td>ANALYZETIME</td>
<td>Time when this synopsis was created by dbms_stats</td>
</tr>
<tr>
<td>SPARE1, SPARE2</td>
<td>Unknown</td>
</tr>
</tbody>
</table>
Merging synopses

```sql
select sb.intcol#,
       count(discriminant(s.hashvalue)) *
       power(2, min(sb.maxsplit)) ndv
from sys.wri$_optstat_synopsis_head$ sh,
     ( select t.intcol#,
               max(split) maxsplit
        from sys.wri$_optstat_synopsis_head$ t
        where t.bo# = 76058
        group by t.intcol#
     ) sb,
     sys.wri$_optstat_synopsis$ s
where sh.bo# = 76058 and
     sh.synopsis# = s.synopsis# and
     sh.intcol# = sb.intcol# and
     ( sh.split = sb.maxsplit or
       mod(s.hashvalue + 1, power(2, sb.maxsplit)) = 0 )
group by sb.intcol#
```
Find maximum split level $d_{\text{max}}$ across all partitions for each column
Filter out all the hash values that do not belong to the selected range i.e. select only the hash values that would remain after $d_{\text{max}}$ splits. This makes sure that all the synopses represent same portion of the domain.
Merging synopses

```
select sb.intcol#,
  count(distinct(s.hashvalue)) * power(2,min(sb.maxsplit)) ndv
from sys.wri$_optstat_synopsis_head$ sh,
     ( select t.intcol#,
        max(split) maxsplit
     from sys.wri$_optstat_synopsis_head$ t
     where t.bo# = 76058
     group by t.intcol# ) sb,
     sys.wri$_optstat_synopsis$s
where sh.bo# = 76058 and
  sh.synopsis# = s.synopsis# and
  sh.intcol# = sb.intcol# and
  ( sh.split = sb.maxsplit or
    mod(s.hashvalue + 1, power(2, sb.maxsplit)) = 0 )
group by sb.intcol#
```

Count the number of distinct hash values in the selected range across all synopses and multiply it by $2^{d_{\text{max}}}$, thus deriving the global NDV for each column.
Incremental maintenance of global statistics

- Find partitions that have undergone significant changes
- Drop synopses for dropped partitions.
- Generate new synopses for new partitions.
- Drop and generate synopses for partitions identified in first step.
- Leave the synopses for unchanged partitions untouched.
- Merge all synopses to generate global statistics.
- Oracle in this case would only need to scan the changed partitions to derive global statistics.
Conclusion

- Finally dbms_stats.auto_sample_size can and should be used since it will provide the most accurate statistics with least amount of resources consumption.
- Any oracle database 11g and above should do the following
  - dbms_stats.set_param('APPROXIMATE_NDV', 'TRUE') (Default)
  - dbms_stats.set_param('ESTIMATE_PERCENT',
    dbms_stats.auto_sample_size) (Default)
- For partitioned tables
  - dbms_stats.set_table_prefs (ownname=>user,
    tabname=>'LINEITEM',
    pname => 'INCREMENTAL',
    pvalue => 'TRUE'
  ) (False by default)
Conclusion

- `dbms_stats.gather_table_stats`
  
  `(ownname=>user,
   tabname=>'LINEITEM',
   estimate_percent=>DBMS_STATS.AUTO_SAMPLE_SIZE,
   granularity => 'GLOBAL' (Only for partitioned tables)
   method_opt=>’for all columns size 1
  );`
Reference

- Oracle patent application 20080120275. Merging synopses to determine number of distinct values in large databases.
- Oracle patent 6732085, Method and system for sample size determination for database optimizers.
- Oracle presentation at SIGMOD 2008 Efficient and scalable statistics gathering for large databases in Oracle 11g.
- Greg Rahn’s blog posting about dbms_stats enhancements and incremental global statistics.
- Optimizer team’s blog posting on managing statistics on large partitioned tables and improvement of auto sampling in 11g.