Interpreting Histogram Information (Doc ID 72539.1)

APPLIES TO:

Oracle Cloud Infrastructure - Database Service - Version N/A and later
Oracle Database Cloud Exadata Service - Version N/A and later
Oracle Database Exadata Express Cloud Service - Version N/A and later
Oracle Database Cloud Schema Service - Version N/A and later
Oracle Database Backup Service - Version N/A and later
Information in this document applies to any platform.

PURPOSE

To provide an understanding of how histogram information is stored and can be interpreted.

SCOPE

Other useful Histogram references:

Document 1445372.1 Histograms: An Overview (10g and Above)

DETAILS

NOTE: In the images and/or the document content below, the user information and data used represents fictitious data from the Oracle sample schema(s) or Public Documentation delivered with an Oracle database product. Any similarity to actual persons, living or dead, is purely coincidental and not intended in any manner.

Histograms are a mechanism for storing more detail information about column data. This data is used by the Cost based Optimizer (CBO) to determine the optimal access path for a query. Without histograms, all the optimizer has to go on is the high and low value for the column, the number of distinct values, the number of nulls and the number of records in the table. (In actual fact the 2nd lowest and highest values are stored in raw format (so they are not terribly helpful) but the other information can be selected from dictionary views.)

Without column statistics, the optimizer assumed a uniform data distribution and generates a column selectivity of 1/NDV (Number of Distinct Values) for equality predicates.

With Histograms you can access more information about the distribution of the row data.

Where there is a non-uniform distribution of data in a column (a high degree of skew in the column distribution), Oracle can store column histograms to lead to a better estimation of selectivity. This should produce plans that are more likely to be optimal than if the standard statistics are used (high and low values plus Number of Distinct Values).

In terms of implementation, we could choose to store every distinct value together with the number of rows for that value. For a small number of values this is efficient and 'width balanced' histograms are used.

As numbers of distinct values increase, the amount of data stored becomes prohibitive and we need to use a different method for storing histogram data. In this case chose to use height balanced histograms.

So using these 2 methods, column histograms provide an efficient and compact way to represent data distributions. When building histograms, the information stored is interpreted differently depending on whether the number of distinct values is less than or equal to the number of buckets request (default 75 maximum 254).

- If the number of distinct values is less than or equal to the number of histogram buckets specified (up to 254) then a Frequency Histogram is created.
- If the number of distinct values is greater than the number of histogram buckets specified, a Height Balanced Histogram is created.

Frequency Histograms

These use buckets to record the row count for each distinct value.

Height Balanced Histograms

These are implemented by dividing the data up in to different 'buckets' where
each bucket contains the same number of values. The highest value in each bucket (or END_POINT) is recorded together with the lowest value in the "zero" bucket.

Once the data is recorded in buckets we recognise 2 types of data value - Non-popular values and popular values.

- Non-popular values - are those that do not occur multiple times as end points.
- Popular values - occur multiple times as end points.

We can use Popular and Non-Popular Values to provide use with various statistics.Since we know how many values there are in a bucket we can use this information to estimate the number of rows in total that are covered by Popular and Non-Popular values.

- The selectivity for popular values can be obtained by calculation the proportion of bucket endpoints filled by that popular value.
- The selectivity for non popular values can now be calculated as 1/number non-popular bucket endpoints, so we can now be more accurate about selectivities than the original 1/NDV, because we have removed the popular values from the equation.

How histograms are used

Histograms are used to get better selectivity estimates for column predicates.

Where there are fewer distinct values than buckets, the selectivity is simply calculated as we have accurate row information for each value. For the case where we have more distinct values than buckets, the following outlines how these selectivities are obtained.

Equality Predicate Selectivity calculated from:

- Popular Value:
  Number of buckets for value / Total Number of buckets
- Non-Popular Value:
  Density see:

> Document 43041.1 Query Optimizer: What is Density?

Less than < (Same principle applies for > & >=)

- All Values:
  Buckets with endpoints < value / Total No. of buckets

Histogram Examples

```
 Table TAB1

SQL> desc tab1
 Name                            Null?    Type
 ------------------------------- -------- ----
 A                                        NUMBER(6)
 B                                        NUMBER(6)
```

Column A contains unique values from 1 to 10000.
Column B contains 10 distinct values.

The value '5' occurs 9991 times.
Values '1, 2, 3, 4, 9996, 9997, 9998, 9999, 10000' occur only once.

i.e.

```
select distinct B , count(*)
from HTAB1
group by B
order by B
;

         B   COUNT(*)
---------- ----------
         1          1
         2          1
         3          1
         4          1
         5       9991
      9996          1
      9997          1
      9998          1
      9999          1
     10000          1

10 rows selected.
```

There is an index on Column B.
Statistics are gathered without Histograms using:

```
exec DBMS_STATS.GATHER_TABLE_STATS (NULL,'HTAB1', method_opt => 'FOR ALL COLUMNS SIZE 1');
```

Setup:

```
drop table HTAB1;
create table HTAB1 (a number, b number);

  Insert into HTAB1 ( A,B) values ( 1,1);
  Insert into HTAB1 ( A,B) values ( 2,2);
  Insert into HTAB1 ( A,B) values ( 3,3);
  Insert into HTAB1 ( A,B) values ( 4,4);
  Insert into HTAB1 ( A,B) values ( 9996,9996);
  Insert into HTAB1 ( A,B) values ( 9997,9997);
  Insert into HTAB1 ( A,B) values ( 9998,9998);
  Insert into HTAB1 ( A,B) values ( 9999,9999);
  Insert into HTAB1 ( A,B) values ( 10000,10000);

commit;
begin
 for i in 5 .. 9995 loop
  Insert into HTAB1 ( A,B)
values ( i,5);
  if (mod(i,100) = 0) then
     commit;
  end if;
 end loop;
 commit;
end;
/
commit;


create index HTAB1_B on HTAB1(b);
exec DBMS_STATS.GATHER_TABLE_STATS (NULL,'HTAB1', method_opt => 'FOR ALL COLUMNS SIZE 1');

alter session set OPTIMIZER_DYNAMIC_SAMPLING = 0;
```

Function to convert raw data in to numeric data:

```
create or replace function raw_to_number(my_input raw)
return number
as
    my_output number;
begin
    dbms_stats.convert_raw_value(my_input,my_output);
    return my_output;
end;
/
```

This results in statistics as follows:

```
column COLUMN_NAME format a5 heading COL
column NUM_DISTINCT format 99990
column LOW_VALUE format 99990
column HIGH_VALUE format 99990
column DENSITY format 99990
column NUM_NULLS format 99990
column NUM_BUCKETS format 99990
column SAMPLE_SIZE format 99990
select COLUMN_NAME,NUM_DISTINCT,raw_to_number(LOW_VALUE) Low,raw_to_number(HIGH_VALUE) High,DENSITY,NUM_NULLS,
       NUM_BUCKETS,LAST_ANALYZED,SAMPLE_SIZE,HISTOGRAM
from user_tab_columns
where table_name = 'HTAB1';

COL   NUM_DISTINCT        LOW       HIGH DENSITY NUM_NULLS NUM_BUCKETS LAST_ANALYZED        SAMPLE_SIZE HISTOGRAM
----- ------------ ---------- ---------- ------- --------- ----------- -------------------- ----------- ---------
A            10000          1      10000       0         0           1 31-jan-2013 09:32:08       10000 NONE
B               10          1      10000       0         0           1 31-jan-2013 09:32:08       10000 NONE


select lpad(TABLE_NAME,10) TAB, lpad(COLUMN_NAME, 10) COL,
 ENDPOINT_NUMBER, ENDPOINT_VALUE
from user_histograms
where table_name='HTAB1'
order by COL, ENDPOINT_NUMBER;


TAB        COL        ENDPOINT_NUMBER ENDPOINT_VALUE
---------- ---------- --------------- --------------
     HTAB1          A               0              1
     HTAB1          A               1          10000
     HTAB1          B               0              1
     HTAB1          B               1          10000
```

In the above you can see that the statistics gathering has not created a histogram. There is a single bucket and high and a low ENDPOINT_NUMBER for each column value (you will always get 2 entries in USER_HISTOGRAMS for each column, for the high and low values respectively).

Test queries:

- ```
  select * from htab1 where b=5;
  ```
- ```
  select * from htab1 where b=3;
  ```

To replicate the tests you will need to disable OPTIMIZER_DYNAMIC_SAMPLING

```
alter session set OPTIMIZER_DYNAMIC_SAMPLING = 0;
```

See:

Document 336267.1 Optimizer Dynamic Sampling (OPTIMIZER_DYNAMIC_SAMPLING)

Without Histograms, both queries do an INDEX RANGE SCAN because the optimizer believes that the data is uniformly distributed in column B and that each predicate with return 1/10th of the values because there are 10 distinct values:

```
-------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Rows  | Bytes | Cost  (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |  1111 |  6666 |     5    (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| HTAB1   |  1111 |  6666 |     5    (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | HTAB1_B |  1111 |       |     3    (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

In fact it may be preferable to use a Full Table Scan for the select where b=5 and index lookups for the others.

Gathering Histogram Statistics

If we collect histogram statistics with the recommended settings:

```
exec DBMS_STATS.GATHER_TABLE_STATS (NULL,'HTAB1', method_opt => 'FOR ALL COLUMNS SIZE AUTO');
```

The b=5 query now does a Full Table Scan

```
 select * from htab1 where b=5;

---------------------------------------------------------------------------
| Id  | Operation         | Name  | Rows  | Bytes | Cost  (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |       |  9991 | 69937 |     7    (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| HTAB1 |  9991 | 69937 |     7    (0)| 00:00:01 |
---------------------------------------------------------------------------
```

The query where B is 3 still uses an index:

```
 select * from htab1 where b=3;

-------------------------------------------------------------------------------------
| Id  | Operation                   | Name    | Rows  | Bytes | Cost  (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |         |     1 |     7 |     2    (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| HTAB1   |     1 |     7 |     2    (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | HTAB1_B |     1 |       |     1    (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

This is because a FREQUENCY Histogram has been created:

```
COL   NUM_DISTINCT        LOW       HIGH DENSITY NUM_NULLS NUM_BUCKETS LAST_ANALYZED        SAMPLE_SIZE HISTOGRAM
----- ------------ ---------- ---------- ------- --------- ----------- -------------------- ----------- ---------
A            10000          1      10000       0         0           1 31-jan-2013 09:58:01       10000 NONE
B               10          1      10000       0         0          10 31-jan-2013 09:58:01       10000 FREQUENCY

TAB        COL        ENDPOINT_NUMBER ENDPOINT_VALUE
```

```
---------- ---------- --------------- --------------
     HTAB1          A               0              1
     HTAB1          A               1          10000
     HTAB1          B               1              1
     HTAB1          B               2              2
     HTAB1          B               3              3
     HTAB1          B               4              4
     HTAB1          B            9995              5
     HTAB1          B            9996           9996
     HTAB1          B            9997           9997
     HTAB1          B            9998           9998
     HTAB1          B            9999           9999
     HTAB1          B           10000          10000

12 rows selected.
```

On Column B there are 10 buckets matching up with the 10 distinct values.

The ENDPOINT_VALUE shows the column value and the ENDPOINT_NUMBER shows the cumulative number of rows. So the number of rows for ENDPOINT_VALUE 2, it has an ENDPOINT_NUMBER 2, the previous ENDPOINT_NUMBER is 1, hence the number of rows with value 2 is 1. Another example is ENDPOINT_VALUE 5. Its ENDPOINT_NUMBER is 9995. The previous bucket ENDPOINT_NUMBER is 4, so 9995 - 4 = 9991 rows containing the value 5.

Frequency histograms work fine with a low number of distinct values, but when the number exceeds the maximum number of buckets, you cannot create a bucket for each value. In this case the Optimizer creates Height balanced histograms.

Height Balanced Histograms

You can demonstrate this situation by forcing the optimizer to create fewer buckets than the Number of Distinct Values. i.e. using 8 buckets for 10 Distinct Values:

```
exec DBMS_STATS.GATHER_TABLE_STATS (NULL,'HTAB1', method_opt => 'FOR COLUMNS B SIZE 8');
```

So now we have gathered a HEIGHT BALANCED HISTOGRAM for Column B:

```
COL   NUM_DISTINCT        LOW       HIGH DENSITY NUM_NULLS NUM_BUCKETS LAST_ANALYZED        SAMPLE_SIZE HISTOGRAM
----- ------------ ---------- ---------- ------- --------- ----------- -------------------- ----------- ---------
A            10000          1      10000       0         0           1 31-jan-2013 09:58:01       10000 NONE
B               10          1      10000       0         0           8 31-jan-2013 09:59:09       10000 HEIGHT BA

TAB        COL        ENDPOINT_NUMBER ENDPOINT_VALUE
---------- ---------- --------------- --------------
     HTAB1          A               0              1
     HTAB1          A               1          10000
     HTAB1          B               0              1
     HTAB1          B               7              5
     HTAB1          B               8          10000
```

Notice that there are 8 Buckets against B now.

Oracle puts the same number of values in each bucket and records the endpoint of each bucket.

With HEIGHT BALANCED Histograms, the ENDPOINT_NUMBER is the actual bucket number and ENDPOINT_VALUE is the endpoint value of the bucket determined by the column value.

From the above, bucket 0 holds the low value for the column.

Because buckets 1-7 have the same endpoint, Oracle does not store all these rows to save space. But we have: bucket 1 with an endpoint of 5, bucket 2 with an endpoint of 5, bucket 3 with an endpoint of 5, bucket 4 with an endpoint of 5, bucket 5 with an endpoint of 5, bucket 6 with an endpoint of 5, bucket 7 with an endpoint of 5 AND bucket 8 with an endpoint of 10000 So bucket 1 contains values between 1 and 5, bucket 8 contains values between 5 and 10000.

All buckets contain the same number of values (which is why they are called height-balanced histograms), except the last bucket may have fewer values then the other buckets.

Storing Character Values in Histograms

For character columns, Oracle only stores the first 32 bytes of any string (there are also limits on numeric columns, but these are less frequently an issue since the majority of numbers are insufficiently large to encounter any problems). See:

Document 212809.1 Limitations of the Oracle Cost Based Optimizer

Any predicates that contain strings greater than 32 characters will not use histogram information and the selectivity will be 1 / Number of DISTINCT Values. Data in histogram endpoints is normalized to double precision floating point arithmetic.

For Example

```
SQL> select * from example;

A
----------
a
b
c
d
e
e
e
e
```

The table contains 5 distinct values. There is one occurence of 'a', 'b', 'c' and 'd' There are 4 occurrences of 'e'. If we create a histogram: Looking in user_histograms:

```
TABLE      COL   ENDPOINT_NUMBER ENDPOINT_VALUE
---------- ----- --------------- --------------
   EXAMPLE     A               1     5.0365E+35
   EXAMPLE     A               2     5.0885E+35
   EXAMPLE     A               3     5.1404E+35
   EXAMPLE     A               4     5.1923E+35
   EXAMPLE     A               8     5.2442E+35
```

So:

```
 ENDPOINT_VALUE          5.0365E+35 represents a
                         5.0885E+35 represents b
                         5.1404E+35 represents c
                         5.1923E+35 represents d
                         5.2442E+35 represents e
```

Then, if you look at the cumulative values for ENDPOINT_NUMBER, the corresponding ENDPOINT_VALUE's are correct.

REFERENCES

NOTE:212809.1 - Limitations of the Oracle Cost Based Optimizer
NOTE:336267.1 - Optimizer Dynamic Statistics (OPTIMIZER_DYNAMIC_SAMPLING)
NOTE:1445372.1 - Histograms: An Overview (10g and Above)