Everything You Want to Know About Oracle Histograms

Part – 1: Basic Concepts & Terminology

Neeraj Bhatia
11/12/2010

Table of Contents

Introduction

Consider following two examples where charts are presented to give a pictorial view of underlying data. For example, chart 1 is taken from a Stanford University study conducted in 2000 about Internet and Society. It displays how many different activities do Internet users engage in. Study suggests that 14% users are engaged in 6 activities (email, surfing, job search etc) and average internet user engages in 7.2 activities. Chart 2 is showing distribution of wealth in America. Data being used is presented in a paper by Edward N Wolff [1]. These data suggest that wealth is concentrated in the hands of a small number of families. The wealthiest 1 percent of American families owns roughly 34.6% of the nation's net worth and the bottom 40% of the population owns only 0.2%.

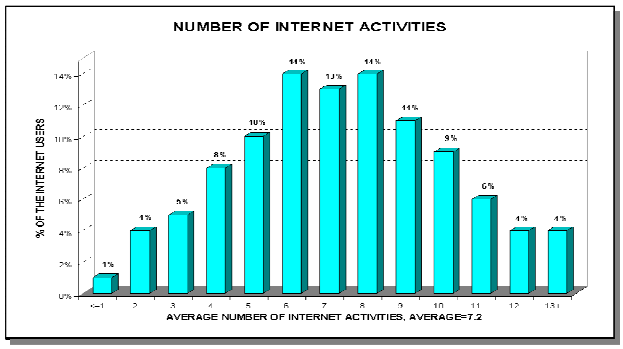


Chart-1: Number of Internet activities

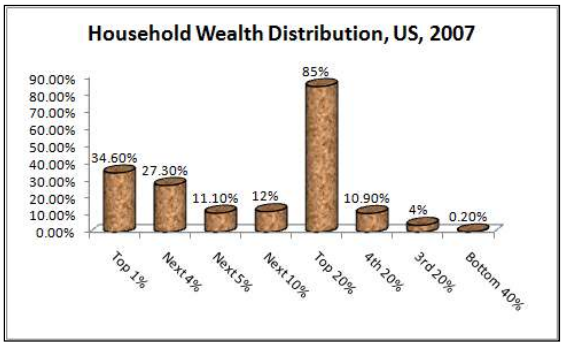


Chart-2: Household wealth distribution in US

In statistics such charts are generally known as histograms. Even if you're not from statistics background, must have encountered with newspaper stories, survey reports etc. containing histograms. Histogram is a way to store the data into particular fashion, different categories along with their corresponding frequencies. It is a very efficient way as it takes very less space and conclusions can be made very easily, which otherwise is a very difficult. For example take an example of household wealth distribution, where we need to take every family and give their wealth information.

Now you might be thinking what is the use of histogram in Oracle? So my answer would be, every SQL we execute [2] goes through the optimization step and SQL optimizer (a.k.a Cost-based Optimizer or CBO) should know the details of underlying objects before giving an optimal execution plan.

One of the object information is number of distinct values in a column and their corresponding frequencies. Now storing each distinct value along with the frequency is not feasible, especially for big table having millions of distinct values, Oracle use histograms for storing useful information about the distribution of the data and Oracle's Cost-based optimizer (CBO) make use of histogram information to come up with optimal execution plan.

[1] Recent Trends in Household Wealth in the United States, Mar 2010
http://www.levyinstitute.org/pubs/wp_589.pdf

[2] CBO comes in the picture during the parse stage, subsequent executions of the SQL reuse the same execution plan.

Oracle Histogram is a vast topic and it's relation with other topics like bind variables, extended statistics make it possible to write 50-60 pages on its practical details. To make this paper under readable limits, I've decided to release it in parts (probably 2 or 3). This is a first paper in the series, where I'm discussing about concepts like how many types of histograms are there, how data are stored in each type of histograms and how to interpret the information in the data dictionary views.

Unless otherwise stated, examples mentioned in the paper are tested on Oracle Database release 10.2.0.4.

CBO and Histograms

The estimated proportion of rows to be returned by a row source, most commonly known as the selectivity, plays an important role in query optimization by CBO. It has a value between 0 and 1. Roughly speaking, CBO will prefer index scan if a small proportion of rows satisfy the predicate and a full table scan if large portion of data from the table needs to be fetched. For example, the query given below will most likely choose to use index scan, if there will be small fraction of rows having deptno equals to 10:

```
select * from emp where deptno=10;
```

In order to estimate the selectivity (or in other words come up with optimal execution plan), CBO takes various inputs in the form of statistics, configuration parameters etc. From a table's column perspective, CBO collects the following statistics:

- Number of distinct values in a column
- Low and high value of a column
- Number of nulls in a column
- Data distribution or histogram information (optional)

In the absence of histograms, optimizer calculates the statistics based on the first three pieces of information i.e., number of distinct values in column, low and high values for a column, number of nulls in a column and number of records in the underlying table. With this information, optimizer assumes the uniform data distribution between low and high values of a column, or in other words occurrence of every distinct value of a column would be same.

To explain it further, let's take an example of a table having 10,000 rows and two columns. Column all_distinct contains distinct rows ranging from 1-10000. Column skew has values 1-10 for first 10 rows and value 10000 for rest of the records i.e. 9990 rows.

```
test@ORADB10G> create table histogram as select rownum all_distinct, 10000 skew from dual connect
by level <= 10000;

Table created.

test@ORADB10G> update histogram set skew=all_distinct where rownum<=10;
```

```
10 rows updated.

test@ORADB10G> select skew, count(*) from histogram group by skew order by skew;

      SKEW   COUNT(*)
---------- ----------
         1          1
         2          1
         3          1
         4          1
         5          1
         6          1
         7          1
         8          1
         9          1
        10          1
     10000       9990

11 rows selected.
```

Now let's gather the optimizer statistics. Please note that this time we're not creating histograms.

```
test@ORADB10G> exec dbms_stats.gather_table_stats(user,'HISTOGRAM', method_opt=>'for all columns
size 1');

PL/SQL procedure successfully completed.

test@ORADB10G> select column_name,num_distinct,density from user_tab_col_statistics where
table_name='HISTOGRAM';

COLUMN_NAME          NUM_DISTINCT    DENSITY
-------------------- ------------ ----------
ALL_DISTINCT                10000      .0001
SKEW                           11 .090909091
```

Density statistics for a column represent its selectivity that is calculated as 1/#num_distinct, if no histogram is present for a column. In case of histograms, density calculation depends on type of histogram and Oracle version. The value of density is between 0 and 1. Optimizer use this statistics to estimate the number of rows to be returned (also called cardinality) by a query using this column in the predicate.

So cardinality = selectivity * Number of rows

Let's see cardinality values for different values of skew in the predicate:

```
test@ORADB10G> explain plan for select * from histogram where skew=1;

Explained.

test@ORADB10G> select * from table(dbms_xplan.display);

------------------------------------------------------------------------------
| Id  | Operation         | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |           |   909 |  5454 |     7   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| HISTOGRAM |   909 |  5454 |     7   (0)| 00:00:01 |
------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("SKEW"=1)
```

```
13 rows selected.

test@ORADB10G> explain plan for select * from histogram where skew=10000;

Explained.

test@ORADB10G> select * from table(dbms_xplan.display);

------------------------------------------------------------------------------
| Id  | Operation         | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |           |   909 |  5454 |     7   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| HISTOGRAM |   909 |  5454 |     7   (0)| 00:00:01 |
------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("SKEW"=10000)

13 rows selected.
```

So Oracle is assuming uniform data distribution in the column skew values and estimating the cardinality = density * 10000 = 909.09 rows. However we know that we have only one row with skew=1 and 9990 rows with skew=10000. This assumption is bound to result in sub-optimal execution plan. For example, if we have an index on column skew, Oracle will use it for the predicate skew=10000 considering the number of rows to be returned equals to 909 or only 9.09%.

```
test@ORADB10G> create index skew_idx on histogram(skew);

Index created.

test@ORADB10G> exec dbms_stats.gather_index_stats(user,'SKEW_IDX');

PL/SQL procedure successfully completed.
```

test@ORADB10G> explain plan for select * from histogram where skew=10000;

Explained.

```
test@ORADB10G> select * from table(dbms_xplan.display);

-----------------------------------------------------------------------------------------
| Id  | Operation                   | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |           |   909 |  5454 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| HISTOGRAM |   909 |  5454 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN          | SKEW_IDX  |   909 |       |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("SKEW"=10000)
```

So we understand that without giving additional inputs, CBO assumes uniform distribution of data between low and high values of a column and chooses sub-optimal plan.

Oracle Histograms

Once histogram is created for a column, it tells the CBO the frequency of a column value. So in our case it would tell the optimizer that we have 1 occurrence (frequency) of skew=1 and 9990 of skew=10000. Thus it will enable the optimizer to choose better execution plans.

In Oracle, we have two types of histograms. First is where Oracle chooses to store every distinct value together with the number of rows or frequency for that value. In Oracle terminology, we call it width-balanced histograms or frequency histograms. This is efficient and possible for columns having small number of distinct values. However for columns having large number of distinct values, it's not feasible to store each and every value along with its frequency. Of course with unlimited resources (space for storing and computing power and time during parsing), we could store frequency for each distinct value in every situation and provide ultimate information to the optimizer, but that is not the case in real life and in such cases, Oracle uses a different method of storing data called height-balanced histograms.

From user's perspective there is a single clause to create histogram for a column, and Oracle automatically decides which type of histogram to create based on the number of distinct values. However, the information the histogram store is interpreted differently depending on the histogram type; hence I am going to cover them one by one.

Width-balanced or Frequency Histograms

In this type of histogram column values are partitioned into equally sized categories or in Oracle terminology buckets. This type of histogram is similar to bar graphs, but in case of Oracle each bucket contains only a single value which corresponds to a distinct value.

Figure given below is a graphical representation of data values in the skew column. Some points are clear while looking at the graph:

- There are 11 buckets represented at the x-axis, each for a single distinct value.
- Y-axis is showing their corresponding frequencies. Frequency is 1 for data values 1-10 and 9990 for value 10000.
- One can easily tell the frequency of a particular data value, by looking at this type of information (graphical or other formats).

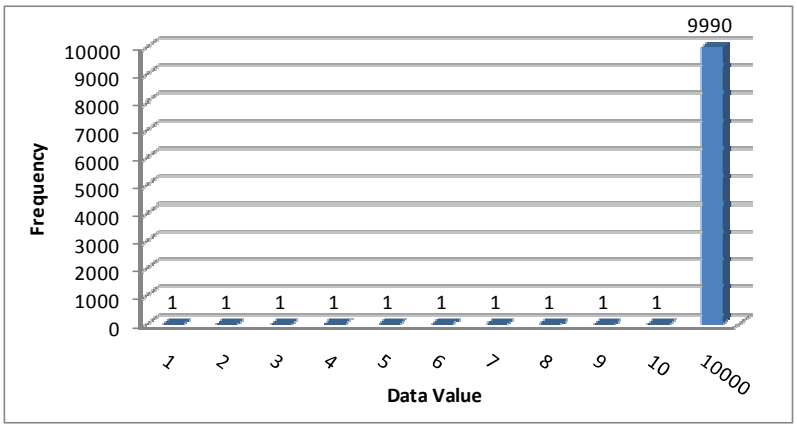


Figure: Frequency histogram representation of our example data

Now let's create frequency histogram for column skew and see how data is stored in data dictionary views. We'll see the different options available for creating histogram in a later section, for the time being let's understand that method_opt 'for column column_name size n' creates histogram for the column column_name with n number of buckets.

```
test@ORADB10G> exec dbms_stats.gather_table_stats(user,'HISTOGRAM',method_opt=>'for columns skew
size 11');

PL/SQL procedure successfully completed.

test@ORADB10G> select column_name,endpoint_number,endpoint_value from user_tab_histograms where
table_name='HISTOGRAM' and column_name='SKEW';

COLUMN_NAME    ENDPOINT_NUMBER ENDPOINT_VALUE
-------------- --------------- --------------
SKEW                         1              1
SKEW                         2              2
SKEW                         3              3
SKEW                         4              4
SKEW                         5              5
SKEW                         6              6
SKEW                         7              7
SKEW                         8              8
SKEW                         9              9
SKEW                        10             10
SKEW                     10000          10000

11 rows selected.
```

The first statement creates histogram for column name skew with 11 buckets; since we know that there are 11 distinct values for skew. The second statement shows histogram data that is stored in Oracle dictionary. The way the information stored in histogram is interpreted differently depending on whether number of buckets requested is less than the number of distinct values or it's the same, or in other words depending on the histogram type. Below explanation pertains to only frequency histograms.

The ENDPOINT_VALUE shows the actual column value and the ENDPOINT_NUMBER shows the cumulative number of rows or in other words cumulative frequency. In order to calculate the frequency of a particular column value, we need to subtract previous cumulative value from corresponding ENDPOINT_NUMER.

For example, for ENDPOINT_VALUE 5, we have ENDPOINT_NUMBER 5 and the previous ENDPOINT_NUMBER is 4, hence the number of rows with value 5 is 1.
Similarly, for ENDPOINT_VALUE 10000, its ENDPOINT_NUMBER is 10000 and the previous bucket ENDPOINT_NUMBER is 10, so the number of rows with value 10000 would be 10000 - 10 = 9990.

Following SQL can be used to translate the histogram information stored in the data dictionary into a user-friendly report of underlying data:

```
test@ORADB10G> select   endpoint_value as column_value,
                        endpoint_number as cummulative_frequency,
          endpoint_number - lag(endpoint_number,1,0) over (order by endpoint_number) as frequency
          from     user_tab_histograms
          where    table_name  = 'HISTOGRAM' and column_name = 'SKEW';

COLUMN_VALUE CUMMULATIVE_FREQUENCY  FREQUENCY
------------ --------------------- ----------
           1                     1          1
           2                     2          1
           3                     3          1
           4                     4          1
           5                     5          1
           6                     6          1
           7                     7          1
           8                     8          1
           9                     9          1
          10                    10          1
       10000                 10000       9990

11 rows selected.
```

Storing running total or cumulative frequency instead of individual frequency is particularly useful in range scans, where cardinality for predicate like where skew <=10 is readily available.

Now since we have created histogram for the column skew, let's see how it makes the difference.

```
test@ORADB10G> select column_name, density, histogram from user_tab_col_statistics where
table_name='HISTOGRAM' and column_name='SKEW';

COLUMN_NAME      DENSITY HISTOGRAM
------------- ---------- ---------------
SKEW              .00005 FREQUENCY

test@ORADB10G> explain plan for select * from histogram where skew=10000;

Explained.

test@ORADB10G> select * from table(dbms_xplan.display);

-------------------------------------------------------------------------------
| Id  | Operation         | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |           |  9990 | 59940 |     6   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| HISTOGRAM |  9990 | 59940 |     6   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("SKEW"=10000)

13 rows selected.

test@ORADB10G> explain plan for select * from histogram where skew=1;
```

```
Explained.

test@ORADB10G> select * from table(dbms_xplan.display);

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |     1 |     6 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| HISTOGRAM |     1 |     6 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | SKEW_IDX  |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("SKEW"=1)

14 rows selected.
```

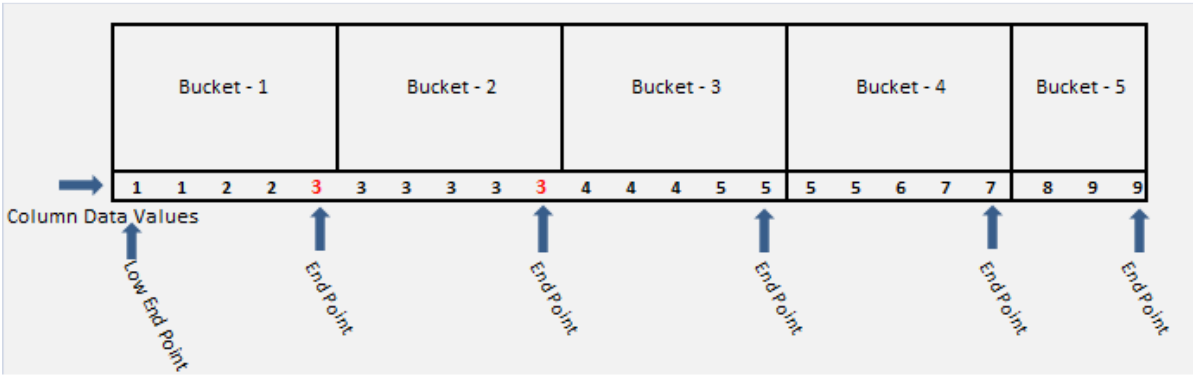
So now optimizer is precisely calculating the cardinality of 9990 rows and due to this calculation now optimizer is choosing full table scan for predicate skew=10000. Also, note that now density value is changed to 0.00005 which is 1/(2*num_rows) or 0.5 / num_rows.

Height-balanced Histograms

In the case of Frequency histograms Oracle allocates a bucket for each distinct value. However the maximum possible number of buckers is 254, so if you have tables with a huge number of distinct values (greater than 254); you would have to go for height-balanced histograms.

In height-balanced histograms, since we have more distinct values than number of buckets, hence Oracle first sorts the column data and then the complete data set is divided into number of buckets and all buckets contain the same number of values (which is why they are called height-balanced histograms), except the last bucket that may have fewer values than the other buckets.

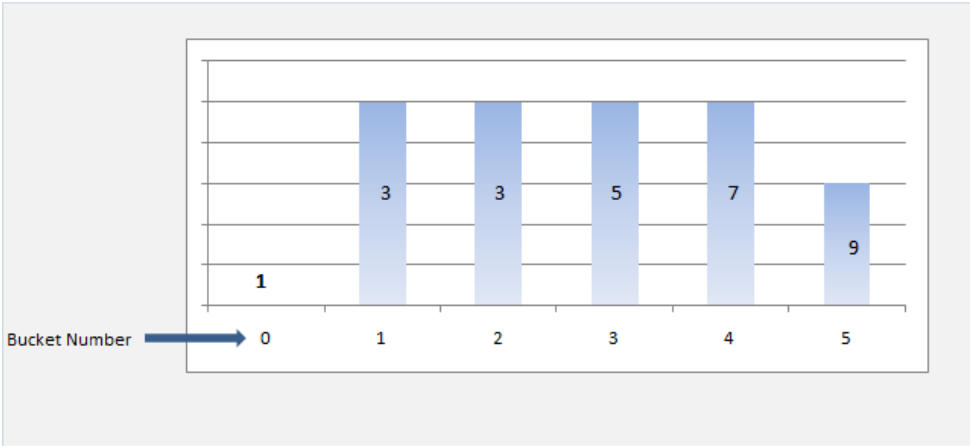
There is no separate statement to create height-balanced histograms. When the number of buckets requested is less than the number of distinct values in a column, Oracle creates height-balanced histograms and the meaning of ENDPOINT_VALUE and ENDPOINT_NUMBER are quite different. To understand how to interpret histogram information, let's take another example of a column data which has 23 values and there are 9 distinct values in the column. Let's suppose we have requested for 5 buckets. Below is a pictorial representation of how data will be stored in histogram.



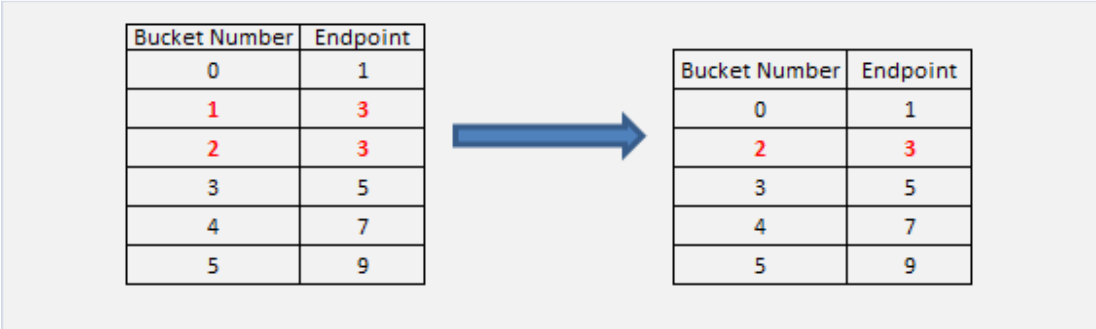
We can make following points based on above picture:

- Number of buckets is less than number of distinct values in the column.
- Since we've requested for 5 buckets, so the total dataset will be divided into equally sized buckets, except the last bucket, which in this case has only 3 values.
- End points of each bucket and first point of the first bucket are marked, as they are of special interest.
- Data value '3' is marked in red color; it is special in the sense that it is end point in multiple buckets.

Following figure is an alternate way of showing the histogram.



With 5 buckets and 23 values means there are 5 values in each bucket except that last bucket which has 3 values. Actually this is the way Oracle stores height-balanced histogram information in data dictionary views, with a minor change. Since Bucket 1 and 2 both have 3 as an end point, Oracle doesn't store bucket 1 so as to save space. So when both buckets will be merged, single entry will be stored.



Bucket Number	Endpoint
0	1
1	3
2	3
3	5
4	7
5	9

Bucket Number	Endpoint
0	1
2	3
3	5
4	7
5	9

Let's create histogram on column skew, this time with number of buckets less than the actual number of distinct values that is 11.

```
test@ORADB10G> exec dbms_stats.gather_table_stats(user,'HISTOGRAM',method_opt=>'for columns skew
size 5');

PL/SQL procedure successfully completed.

test@ORADB10G> select table_name, column_name,endpoint_number,endpoint_value from
user_tab_histograms where table_name='HISTOGRAM' and column_name='SKEW';

TABLE_NAME     COLUMN_NAME    ENDPOINT_NUMBER ENDPOINT_VALUE
-------------- -------------- --------------- --------------
HISTOGRAM      SKEW                         0              1
HISTOGRAM      SKEW                         5          10000
```

Here buckets 1-5 all have 10000 as an end point so these buckets 1-4 are not stored so as to save space. Following SQL query can be used to display bucket numbers and respective endpoint numbers.

```
test@ORADB10G> SELECT bucket_number, max(skew) AS endpoint_value
  2  FROM (
  3       SELECT skew, ntile(5) OVER (ORDER BY skew) AS bucket_number
  4       FROM histogram)
  5  GROUP BY bucket_number
  6  ORDER BY bucket_number;

BUCKET_NUMBER ENDPOINT_VALUE
------------- --------------
            1          10000
            2          10000
            3          10000
            4          10000
            5          10000
```

Here ntile(5) is an analytic function, It divides an ordered data set into a 5 buckets.

So in nutshell, in height-balanced histograms, data is divided into different 'buckets' where each bucket contains the same number of values. The highest value in each bucket is recorded together (ENDPOINT_VALUE) with the lowest value in the first bucket (bucket 0). Also, ENDPOINT_NUMBER represents the bucket number. Once the data is recorded in buckets we recognize 2 types of data value - Non-popular values and popular values.

Popular values are those that occur multiple times as end points. For instance, in our previous example 3 is a popular value and in the column skew 10000 is a popular value. Non-popular values are those that do not occur multiple times as end times. As you might be thinking, popular and non-popular values are not fixed and depend on bucket size. Changing the bucket size will result in different popular values.

Let me summarize our discussion w.r.t two histogram types:

- Distinct values less than or equal to the number of buckets: When you have less number of distinct values than the number of buckets, the ENDPOINT_VALUE column contains the distinct values themselves while the ENDPOINT_NUMBER column holds the CUMULATIVE number of rows with less than that column value (Frequency Histograms).

- More number of distinct values than the number of buckets: When you have more number of distinct values than the number of buckets, then the ENDPOINT_NUMBER column contains the bucket id and the ENDPOINT_VALUE holds the highest value in each bucket. Bucket 0 is special in that it holds the low value for that column (Height-balanced Histograms).

Creating Histograms

The GATHER_TABLE_STATS procedure of DBMS_STATS package is used to gather table and column statistics and optionally we can instruct to create histogram on certain column(s) using the method_opt parameter:

The 'method_opt' parameter of the procedure accepts following values:

- For all [Indexed/Hidden] Columns [Size option]

- For Columns column_name [Size option] column_name [Size option] …

The SIZE keyword specifies the maximum number of buckets for the histogram and takes following values:

SIZE {integer | REPEAT | AUTO | SKEWONLY}

- integer: Number of histogram buckets. Must be in the range 1-254.
- Repeat: Collects histograms only on the columns that already have histograms.
- Auto: Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.
- Skewonly: Oracle determines the columns to collect histograms based on the data distribution of the columns.

Auto option also considers the workload of the columns. What it means is that it checks for the SQL queries having column name in where predicate.

The default for method_opt is changed to 'For all Columns Size Auto' in 10g, which in 9i used to have 'For all columns size 1'. In other words, Oracle now automatically decides for us which columns need histograms and number of buckets also. This seems ideal situation, but this has many caveats which are not in the scope of this paper. In next part, I'll touch base on this topic in greater detail. The default value can be changed using the SET_PARAM Procedure.

Viewing Histograms

We are fetching histogram information for a while, now it's time to see in detail the various options available for histogram information. We can find information about existing histograms in the database through DBA_TAB_HISTOGRAMS data dictionary view. This view lists histograms on columns of all tables. The actual value may be stored in ENDPOINT_ACTUAL_VALUE if the column is not a number (i.e. a varchar2) and the first six bytes of some values are the same.

Number of buckets in each column's histogram and density value can be found in DBA_TAB_COLUMNS and DBA_TAB_COL_STATISTICS data dictionary views. The latter extracts the data from DBA_TAB_COLUMNS only.

There are corresponding views available for partition and sub-partitions columns, for example DBA_SUBPART_HISTOGRAMS etc.

Next Steps …

So far we've discussed basic concepts of Oracle histograms and how to interpret histogram data stored in data dictionary for both type of histograms. As mentioned earlier, basic usage of histogram is to provide best estimation of table proportion (cardinality) which fulfills the filtering criteria. Understanding the cardinality calculations is the next step in the journey of Oracle histograms. Similarly, bind variables and histograms share a very special relationship – a contradiction in their purpose. Also, so far we have only discussed histograms on numeric values and haven't touched base on histograms on other data types like date and character columns. There is enough material to be covered in next parts, but I believe this text will lay a foundation for upcoming parts.

Please feel free to contact me, in case you have any questions or comments...