Histograms

Introduction

Histograms are a feature of the cost-based optimizer (CBO) that allows the Oracle engine to determine how data is distributed within a column. They are most useful for a column that is included in the WHERE clause of SQL and the data distribution is skewed.

Example

Assume a table named PROCESS_QUEUE with one million rows including a column named PROCESSED_FLAG with five distinct values. Also assume a query similar to the following is executed:

```
SQL> SELECT id, serial_number FROM process_queue
     WHERE processed_flag = 'N';

SELECT STATEMENT Optimizer=ALL_ROWS (Cost=1087 Card=260363
Bytes=7029801)
TABLE ACCESS (FULL) OF 'PROCESS_QUEUE' (TABLE) (Cost=1087
Card=260363 Bytes=7029801)
```

Without histograms and only five distinct values, Oracle assumes an even data distribution and would most likely perform a full table scan for this query. With one million rows and five values, Oracle assumes that each value would return 200,000 rows, or 20% of the rows.

Data Skew

However, what if the data for the PROCESSED_FLAG column was skewed:

```
SQL> SELECT processed_flag, COUNT(1)
     FROM process_queue
     GROUP BY processed_flag;

PROCESSED_FLAG          COUNT

P                       24
Y                       999345
E                       30
S                       568
N                       33
```

In this case, only 33 rows have a value of 'N', so there has to be a way to tell Oracle to use the index on the PROCESSED_FLAG column. That is where histograms come into use. A histogram would include data similar to above and allow Oracle to know that only 33 rows would be returned for this query.

What is Histogram?

Histograms are feature in CBO and it helps to optimizer to determine how data are skewed(distributed) with in the column. Histogram is good to create for the column which are included in the WHERE clause where the column is highly skewed. Histogram helps optimizer to decide whether to use an index or full-table scan or help the optimizer determine the fastest table join order.

What are the advantage of Histogram? Histograms are useful in two places.

1. Histograms are useful for Oracle optimizer to choose the right access method in a table.

2. It is also useful for optimizer to decide the correct table join order. When we join multiple tables, histogram helps to minimize the intermediate result set. Since the smaller size of the intermediate result set will improve the performance.

Type of Histograms?

Oracle uses two types of histograms for column statistics: height-balanced histograms and frequency histograms.

1. Frequency Histograms : Each value of the column corresponds to a single bucket of the histogram. This is also called value based histogram. Each bucket contains the number of occurrences of that single value. Frequency histograms are automatically created instead of height-balanced histograms when the number of distinct values is less than or equal to the number of histogram buckets specified.

Oracle will ONLY create a Frequency histograms IF and ONLY IF
(Number of Distinct Values in the column) <= (Number of buckets)

If the number of buckets is < the number of distinct values in the column then a Height Balanced histogram is created

If the column contained the values
A, B, B, B, C, C, C, C, C, C, C, C, C, D, E, E, F, F, F, G
And the table was analyzed correctly to build a Frequency Histogram, the Frequency Histogram would contain:

```
A - 1
B - 3
C - 9
D - 1
E - 2
F - 3
G - 1
```

2. Height - balanced Histograms : The column values are divided into bands so that each band contains approximately the same number of rows. For instances, we have 10 distinct values in the column and only five buckets. It will create height based(Height balanced) histograms and it will evenly spread values through the buckets. A height-based histogram is when there are more distinct values than the number of buckets and the histogram statistics shows a range of rows across the buckets . When the number of distinct values in a column is > 254. This forces a Height Balanced histogram.

In a height-balanced histogram, the column values are divided into bands (Buckets) so that each band contains approximately the same number of rows.

If the column contained the values

A, B, B, B, C, C, C, C, C, C, C, C, C, D, E, E, F, F, F, G

And the number of buckets was 4 the database, the data base cuts this up into 4 bands/buckets/ranges(20 rows / 4 buckets = Store value of every 5th row) and stores the values at value of the column so if the data was

Column Value	A	B	B	B	C	C	C	C	C	C	C	C	C	D	E	E	F	F	F	G
Row Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Values Saved					Yes					Yes					Yes					Yes

The height-balanced histogram contains the values

```
0 - C
1 - C
2 - E
3 - G
```

Notice that the value of C is stored twice, this is how the database knows that a value of a column is very popular in the column. And using an index, for example to retrieve rows from the table may not be very efficient since the number of rows is likely to be high.

So In a height-balanced histogram, the column values are divided into bands (Buckets) so that each band contains approximately the same number of rows.

If the table has 100,000 rows and the number of buckets is 200 the database, the data base cuts this up into 200 bands/buckets/ranges(100,000 rows / 200 buckets = Store value of every 500th row).

Collecting Histograms

To collect histograms for this column, a command similar to the following could be used:

```
SQL> EXECUTE DBMS_STATS.GATHER_TABLE_STATS
     (User, 'PROCESS_QUEUE',
     method_opt => 'for columns processed_flag size 5')

SQL> SELECT id, serial_number
     FROM process_queue
     WHERE processed_flag = 'N';

SELECT STATEMENT Optimizer=ALL_ROWS (Cost=1 Card=28 Bytes=756)
TABLE ACCESS (BY INDEX ROWID) OF 'PROCESS_QUEUE' (TABLE) (Cost=1
Card=28 Bytes=756)
INDEX (RANGE SCAN) OF 'PQ_IX1' (INDEX) (Cost=1 Card=28).
```

The defaults for the METHOD_OPT parameter changed between Oracle 9i and 10g.
In 9i the parameter defaulted to 'for all columns size 1' which essentially turns off histograms.

The default value in Oracle 10g and later versions is 'for all columns size auto' which means that Oracle will decide whether or not to collect histograms for a column.

In my experience it seems that unnecessary histograms are collected and histogram data is not collected for some columns where it would be useful.

Method_opt Constants	Accepts: * FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause] * FOR COLUMNS [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...] size_clause is defined as size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY} - integer : Number of histogram buckets: Range [1,254]. - REPEAT : Collects histograms only on columns that already have histograms. - AUTO : Determines the columns to collect histograms based on data distribution and the workload of the columns. - SKEWONLY : Determines the columns to collect histograms based on the data distribution of the columns. The default is FOR ALL COLUMNS SIZE AUTO.The default value can be changed using the SET_PARAM Procedure.

Data dictionary objects for Histogram:

user_histograms
user_part_histograms
user_subpart_histograms
user_tab_histograms
user_tab_col_statistics

Note: The columns that really do need histograms, where it really does matter, are the indexed columns. Those are the only columns where the optimizer can choose the access path. Histograms will take quite a bit of space in SYSAUX tablespace and that space was the reason not to collect histograms for all columns in the old days of 8i.

When Not To Create A Histogram

Do not create Histograms when

1) Data in column is evenly distributed.
2) Column is not used in a where clause.
3) Do not create them on every column of every table. This requires more time when the table is analyzed. Increases parse time. And can result in poor plans being generated by the optimizer.
4) No Histograms on Primary Key (PK) of a table. You may find them on PK columns because someone used 'Auto' on the gather stats command.
5) Use Histograms like Hot Peppers, A little makes a dish great, too many well you get the point I hope ;-)

Conclusion

Using histograms works best for SQL statements that use literal values

Histograms allow Oracle to make much better performance decisions. The case we discussed in this article is one way that histograms are used and is commonly referred to as "table access method" histograms.

Another use for histograms, referred to as "table order join" histograms, is to help Oracle decide the order in which tables will be joined. This helps the CBO know the size of the result sets or "cardinality" to properly determine the correct order in which to do join.

Case Study:

```
SQL> create table naveed.bigtab as select * from dba_objects;

SQL> select count(*) from naveed.bigtab;

  COUNT(*)
----------
     14311

SQL> select object_type, count(*)
     from naveed.bigtab group by object_type;

OBJECT_TYPE           COUNT(*)
------------------- ----------
CONSUMER GROUP              25
INDEX PARTITION            126
EDITION                      1
TABLE SUBPARTITION          32
SEQUENCE                   150
QUEUE                       24
SCHEDULE                     3
TABLE PARTITION            114
RULE                         1
PROCEDURE                  111
OPERATOR                     7
LOB PARTITION               12
WINDOW                       9
SCHEDULER GROUP              4
DESTINATION                  2
LOB                        218
PACKAGE                    617
PACKAGE BODY               594
LIBRARY                    149
RULE SET                    13
PROGRAM                     19
TYPE BODY                  115
CONTEXT                      3
TRIGGER                      6
JOB CLASS                   14
UNDEFINED                   11
DIRECTORY                   10
TABLE                     1282
INDEX                     1532
SYNONYM                   3623
VIEW                      3959
FUNCTION                    94
CLUSTER                     10
TYPE                      1387
```

```
RESOURCE PLAN                  11
EVALUATION CONTEXT             10
JOB                            13
37 rows selected.

SQL> CREATE INDEX OBJECT_TYPE_IDX ON NAVEED.BIGTAB (OBJECT_TYPE);

SQL> Execute DBMS_STATS.GATHER_TABLE_STATS
     (OWNNAME => 'NAVEED',
      TABNAME => 'BIGTAB',
      ESTIMATE_PERCENT => 100,
      METHOD_OPT => 'FOR ALL COLUMNS SIZE 1',
      CASCADE => TRUE);
PL/SQL procedure successfully completed.

SQL> set autotrace on EXPLAIN
SQL> select object_type from naveed.bigtab
     where object_type ='CONTEXT';
```

```
Execution Plan
----------------------------------------------------------
Plan hash value: 1882467121
-------------------------------------------------------------------------------------------
| Id  | Operation        | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                 |   443 |  3101 |     2   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN| OBJECT_TYPE_IDX |   443 |  3101 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("OBJECT_TYPE"='CONTEXT')
```

FOR COLUMN SIZE 37 OBJECT_TYPE will create 37 bucket for column OBJECT_TYPE.
If we are not sure the distinct number of values in the column, then we can use AUTO option to collect histogram. With this histogram, oracle optimizer knows that, the column OBJECT_TYPE is highly skewed and it has 37 bucket.

Now depends upon the query, optimizer decides whether to use index or Full table scan.

```
SQL> execute DBMS_STATS.GATHER_TABLE_STATS
     (OWNNAME => 'NAVEED',
      TABNAME => 'BIGTAB',
      ESTIMATE_PERCENT => 100,
      METHOD_OPT => 'FOR COLUMNS SIZE 37 OBJECT_TYPE',
      CASCADE => TRUE);
PL/SQL procedure successfully completed.
```

```
SQL> select object_type from naveed.bigtab
     where object_type ='CONTEXT';

Execution Plan
----------------------------------------------------------
Plan hash value: 1882467121

-------------------------------------------------------------------------------------------
| Id  | Operation         | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                 |     3 |    21 |     1   (0)| 00:00:01 |

|*  1 |  INDEX RANGE SCAN| OBJECT_TYPE_IDX |     3 |    21 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("OBJECT_TYPE"='CONTEXT')
```