New Density calculation in 11g

Alberto Dell'Era
alberto.dellera@gmail.com

ABSTRACT

A short note on the new formula for the "density" statistic used internally by the Oracle™ Cost Based Optimizer in 11g.

Introduction

Starting from 11g (11.1.0.6), the CBO does not use anymore the old statistic "density" computed by dbms_stats and stored in the data dictionary, but instead computes internally a new value (named *NewDensity* in 10053 trace files) which is used to estimate the cardinality of SQL statements when histograms are present.

The CBO classifies at runtime the histogram type (since the histogram type is not stored in the data dictionary) using a heuristic (probably a formula similar to the one exposed in the view DBA_TAB_COLUMNS for column HISTOGRAM). If the histogram is deemed an HB, the formula we are going to see in a moment is used; if is considered a FH, the traditional setting of 0.5 / num_rows is used instead.

The new setting is controlled by the hidden (and so not to be changed without the approval of Oracle Support) parameter _optimizer_enable_density_improvements; setting this parameter to false gets back the old behavior.

Note: for simplicity, we are going to consider only not-null values. Corrections for columns containing nulls are relatively trivial.

Acknowledgments

Thanks to Wolfgang Breitling for reading the first version of this note, and pointing out some weak points.

NewDensity for histograms classified as Height-Balanced

The script 11g_NewDensity_base_case.sql creates table T with 15 rows, with a skewed data distribution for column VALUE, and computes a histogram with SIZE N, N=5.

The histogram from DBA_HISTOGRAMS, plus two computed columns we are going to use below, is:

```
     VALUE         EP        BKT POPULARITY
---------- ---------- ---------- ----------
         1          0          0          0
         2          2          2          1
         9          3          1          0
        15          5          2          1
```

- VALUE is DBA_HISTOGRAMS.ENDPOINT_VALUE and EP is DBA_HISTOGRAMS.ENDPOINT_ NUMBER.
- BKT is simply EP - previous (EP), and can be considered the number of buckets "covered" by the value in the *uncompressed* histogram. It's worth remembering that when dbms_stats computes an HB of SIZE N it sorts the column values first, then samples the sorted set on an uniform grid of width approximately equal to num_rows/N, labels the samples with EP=1..N, adds the minimum value as EP=0, removes the samples whose VALUE is equal to the next (compression), and eventually stores the remaining (EP, VALUE) pairs in the data dictionary. For more information, see [Dell'Era, JOH, page 4, "Height-Balanced Histograms"], [Breitling, HMF, page 1], [Lewis, CBO, page 169].
- POPULARITY is 1 if the value is a popular one, zero otherwise. A value is considered popular if BKT>1, that is, if it was sampled more than once on the uniform grid. See the same papers quoted above for more information.

In 11.1.0.6, in the 10053 trace for "where value=<unpopular value>", we find:

```
Column (#1):
NewDensity:0.050000, OldDensity:0.066667 BktCnt:5, PopBktCnt:4, PopValCnt:2, NDV:6
Using density: 0.050000 of col #1 as selectivity of unpopular value pred
```

- *OldDensity* is the density stored in the data dictionary; here the CBO decides to ignore it and to use the brand-new NewDensity instead.
- *BktCnt* is max(EP), which is the same as N for HB histograms (the number of buckets in the uncompressed histogram), here equal to 5.
- *PopBktCnt* is the number of popular buckets, that is, sum(BKT) for all popular values. We have two popular values, 2 and 15, for a total of 2+2=4.
- *PopValCnt* is the number of popular values (we have two: 2 and 15).
- *NDV* (Number of Distinct Values) is num_distinct from DBA_TAB_COLUMNS.

The formula for NewDensity is

```
NewDensity = [(BktCnt - PopBktCnt) / BktCnt] / (NDV - PopValCnt) =
           = [(5 - 4) / 5] / (6-2) = .05 (as required)
```

I have validated the formula above using an automated script (not shown here) that builds thousands of value distributions and checks the formula against the CBO output.

Rationale for the NewDensity formula

It is well-known that when the CBO doesn't know the value distribution of a column (no histogram is collected), it assumes a uniform distribution, that is, assumes that every distinct value in the column occurs as frequently as any other distinct value. This is because without any other information, this is the simplest distribution one can assume.

Assuming a uniform distribution mathematically (statistically) translates into estimating the cardinality of a filter predicate such as "where column=constant" as num_rows / num_distinct, that is, each distinct value is assumed occurring num_rows / num_distinct times. This is exactly what the CBO does for columns without histograms.

If an HB histogram computed with SIZE N is available, we can (conceptually) divide the table in a Populars SubTable (PST), that contains all popular values, and a Not-Populars SubTable (NPST), that contains all the other values (whether they are represented in the histogram or not). Our filter predicate "where column=constant" will select values from one of the two SubTables.

When the filter predicate selects from the PST, density (or NewDensity) is not used, since the CBO can simply search for "constant" in the histogram and then easily compute a good estimate; but when the filter predicate selects from the NPST, searching for "constant" in the histogram is useless. The reason for this uselessness is simple but requires some space to be illustrated effectively, so I'll point the interested ones to [Dell'Era, JOH, page 4-6] instead of rehashing the same material[1].

Not being able to use the histogram is equivalent to say that the distribution of values in the NPST is not known, that is, the real distribution is not represented in the histogram. And since the CBO has no other information available to factor in the real distribution, it resorts to assuming a uniform distribution for the NPST, and applying the formula used for columns without histograms on the NPST.

This is easily shown - if the NPST were actually materialized in an actual table with statistics collected (of course, with no histogram), the estimated cardinality for our filter predicate would be

```
   cardinality = num_rows(NPST) / num_distinct (NPST)
```

but

```
   num_rows(NPST) = [(BktCnt - PopBktCnt) / BktCnt] * num_rows (table)
   num_distinct (NPST) = NDV - PopValCnt
```

[1] To link the two paper material, keep in mind that "COUNTS" in the other paper is the same as "(BKT / BktCnt) * num_rows" in this.

hence

```
   cardinality = NewDensity * num_rows (table)
```

And in fact, NewDensity * num_rows is exactly the formula the CBO uses for "where column=constant" when constant is not a popular value.

NewDensity is used in 11g also to estimate the cardinality of a join. This is not surprising since a join has a strong relation with filter predicates (a join is nothing else, conceptually, than a nested loop, where the inner table is probed with filter predicates). I have verified that the same formula described in [Dell'Era, JOH] applies in 11g (11.1.0.6 while I'm writing this) once you substitute "density" with "NewDensity".

NewDensity for histograms classified as Frequency

When the CBO classifies at runtime the histogram as "Frequency", the formula for NewDensity becomes simply the traditional 0.5 / num_rows.

If we run script 11g_NewDensity_base_case.sql again, but collecting a Frequency Histogram (i.e. using SIZE 254) instead of an HB one, in the 10053 trace file we find, again for "where value=<unpopular value>":

```
Column (#1):
NewDensity:0.033333, OldDensity:0.033333 BktCnt:15, PopBktCnt:11, PopValCnt:2, NDV:6
Using density: 0.033333 of col #1 as selectivity of unpopular value pred
```

So NewDensity = 0.5 / num_rows = 0.5 / 15 = 0.033333333, which is incidentally the same value computed by dbms_stats and shown above as "OldDensity".

Histogram type classification mistakes (for FHs)

The script 11g_NewDensity_base_case.sql offers a nice opportunity to show what happens when a histogram which is structurally a Frequency one is mistaken at runtime for an HB, and so the formula for HB is used. Deceiving the CBO is as easy as (in this case) changing the statistic density by using the procedure set_density provided in the script, which sets density preserving the histogram.

If we increase density from 0.033333333 to 0.2 we get:

```
Column (#1):
NewDensity:0.066667, OldDensity:0.200000 BktCnt:15, PopBktCnt:11, PopValCnt:2, NDV:6
Using density: 0.066667 of col #1 as selectivity of unpopular value pred

NewDensity = [(BktCnt - PopBktCnt) / BktCnt] / (NDV - PopValCnt) =
             [(15 - 11) / 15] / (6 - 2) = .066666667 (as required)
```

It is interesting to note that .066666667 is simply 1.0 / num_rows, and moreover, it can be shown mathematically that it is always the case: the HB formula above, when applied to histograms that are structurally FH, always gets back 1.0 / num_rows.

Is this bad? For our filter predicate "where value=<unpopular value>" is it exactly the same, since the cardinality estimate num_rows*NewDensity is always rounded (up), and of course round (0.5) = round (1). Actually, the formula for HB is more accurate - it estimates the correct cardinality even without rounding.

Here's a scenario where setting NewDensity = 1.0 / num_rows gets much more accuracy.

The scenario is that of a pseudo-parent table and a pseudo-child one. "Pseudo" because we are going to slightly violate (one single exception) the uniqueness constraint on the parent; this is to reproduce the typical scenario of applications that try to manage constraints by themselves, without enforcing PK/FK constraints in the database, a sure way to experience at least slight constraint violations in the tables.

Script 11g_NewDensity_frequency_HBformula_is_better.sql builds the scenario:

```
  -- parent table with 100 unique values
  create table parent as select rownum-1 value from dual connect by level <= 100;
  -- child table with the same 100 distinct values
  create table child as select mod(rownum-1,100) value from dual connect by level <= 1000;
```

and then computes histograms with SIZE 254 on both tables.
Incidentally, it makes perfect sense to build a histogram on a column with unique values (parent.value) if, for example, other queries apply range predicates on the column (where parent.value < ..., where parent.value between ...) and there are outliers (e.g. parent.value contains approximately equally-spaced values, say 1,2,3,4 and then an outlier such as 999). Skewness is not only about different distinct value multiplicity.

The join over the pseudo-FK gets:

select count(*) from parent, child where parent.value = child.value;

```
    COUNT(*)
  ----------
        1000

  --------------------------------------------
  | Id  | Operation           | Name   | Rows  |
  --------------------------------------------
  |   0 | SELECT STATEMENT    |        |     1 |
  |   1 |  SORT AGGREGATE     |        |     1 |
  |*  2 |   HASH JOIN         |        |  1001 |
  |   3 |    TABLE ACCESS FULL| PARENT |   100 |
  |   4 |    TABLE ACCESS FULL| CHILD  |  1000 |
  --------------------------------------------
```

So the cardinality is estimated with fantastic accuracy. This is because the histogram on the unique column parent.value is classified at runtime as an HB, and density set to 1.0 / num_rows(parent); how density (or NewDensity) influences the join cardinality estimation is explained in [Dell'Era, JOH], and here the only important contributor is what I call the "populars not matching populars" one in the paper.

The script continues by making a single violation of the uniqueness:

```
  insert into parent (value) values (1000);
  insert into parent (value) values (1000);
```

This produces:

```
    COUNT(*)
  ----------
        1000

  --------------------------------------------
  | Id  | Operation           | Name   | Rows  |
  --------------------------------------------
  |   0 | SELECT STATEMENT    |        |     1 |
  |   1 |  SORT AGGREGATE     |        |     1 |
  |*  2 |   HASH JOIN         |        |   506 |
  |   3 |    TABLE ACCESS FULL| PARENT |   102 |
  |   4 |    TABLE ACCESS FULL| CHILD  |  1000 |
  --------------------------------------------
```

This is because the histogram on parent.value is now structurally a FH, it is classified as such at runtime, and so NewDensity is set to 0.5 / num_rows (parent). The net effect is that the cardinality is badly estimated with approximately a 50% error (506 / 1000).

What if NewDensity were set to 1.0 / num_rows (parent) instead? The last section of the script checks this, by manually setting density = 1.0 / num_rows and setting _optimizer_enable_density_improvements=false (not a recommended setting at all, it is done here just to simulate a setting of NewDensity = 1.0 / num_rows). The result is:

```
    COUNT(*)
  ----------
        1000

  --------------------------------------------
  | Id  | Operation           | Name   | Rows  |
  --------------------------------------------
  |   0 | SELECT STATEMENT    |        |     1 |
  |   1 |  SORT AGGREGATE     |        |     1 |
  |*  2 |   HASH JOIN         |        |  1011 |
  |   3 |    TABLE ACCESS FULL| PARENT |   102 |
  |   4 |    TABLE ACCESS FULL| CHILD  |  1000 |
  ---------------------------------------------
```

So, we would be back to fantastic accuracy.

So in conclusion: the NewDensity formula for HBs, when applied to FHs, sets NewDensity = 1.0 / num_rows instead of 0.5 / num_rows, and the former seems to get better cardinality estimation accuracy than the latter. So there's nothing to be concerned (quite the opposite) if that happens - one might perhaps be concerned about the opposite, that is, if a HB were classified at runtime as a FH by mistake.

NewDensity for 9i and 10g?

Since the old density formula is deemed obsolete in 11g, and superceded by NewDensity, it is probably worth considering setting manually density in 9i and 10g (the set_density procedure used in the scripts can make this easily) using the formula for NewDensity (the variation for HBs can be computed by using the "newdensity" view in 11g_NewDensity_base_case.sql). As per the above discussion, it is probably worth using the HB formula whatever the histogram type is.

I wouldn't do it for every table in the database, but only for the "problematic" ones, of course testing the effect of this setting in my scenario. Yet, it is an interesting troubleshooting option.

Bibliography

[Lewis, CBO]	Jonathan Lewis, *Cost Based Oracle: Fundamentals*, Apress, 2006, ISBN 978-1590596364.
[Breitling, HMF]	Wolfgang Breitling, *Histograms - Myths and Facts*, http://www.centrexcc.com
[Dell'Era, JOH]	Alberto Dell'Era, *Join Over Histograms*. Available on www.adellera.it/investigations/join_over_histograms
[Dell'Era, SWR]	Alberto Dell'Era, *Select without replacement* Available on www.adellera.it/investigations/select_without_replacement

Paper version: 1.1, 2007-12-11
Converted into pdf format by PrimoPdf, configuration "_prepress.ini"