A Smart Optimizer Gets Even Smarter: Top Frequency and Hybrid Histograms in Oracle 12cR1
by Jim Czuprynski

How Many Ways Can You Monitor Oracle GoldenGate?
by Bobby Curtis

Oracle WebLogic Server: The Fusion Middleware Foundation
by Eric Mader
A Smart Optimizer Gets Even Smarter: Top Frequency and Hybrid Histograms in Oracle 12cR1

By Jim Czuprynski
Arup Nanda, Editor

Oracle Database 12cR1’s latest improvements to SQL tuning — Adaptive Execution Plans (AEP), Automatic Re-Optimization (ARO) and SQL Plan Directives (SPDs) — were the focus of last issue’s article. In this issue, we’ll take a deeper look into one of the more misunderstood and often underutilized features of Oracle optimizer statistics — histograms — and look at the new top frequency and hybrid histogram types.

Histograms: A Brief Summary

Consider a column in a database table named GENDER that identifies a person’s gender. In most cases, this column will be populated with a value either indicating (M)ale or (F)emale. But what if a small number of people answered “None of your business!” when asked about their gender, and this was coded as (U)nreported? The GENDER column would now have three distinct values, but one value would have a severely skewed distribution. And therein lies the rub.

• If the query optimizer recognizes that three distinct values for GENDER exist, but incorrectly assumes that these three values are evenly distributed across all rows, it may decide that a table scan is the most effective way to apply selection criteria against all people with a GENDER of (U)nreported when, say, an indexed search against that column would be most cost-effective.

• Equally important, the optimizer would also assume that the cardinality of the row set returned for a GENDER value of (U)nreported would be approximately one-third of all entries in the table. When it joins this table to another row source, it may make an incorrect decision on which row source to join first — the join order — or which join method — NESTED LOOP, HASH or SORT MERGE — is most optimal for constructing the combined row set.

• Finally, what about a query whose selection criteria cared nothing about the unpopular value of (U)nreported, but instead was concerned with a popular value like (F)emale? The optimizer would need to recognize that a table scan is the most efficient way of retrieving data because nearly 50 percent of all rows would meet that criteria, and an indexed search might be considerably more inefficient.

The answer, of course, is to provide the optimizer with additional information about the skewed nature of data in the GENDER column so that it can make the most intelligent decision. Before Oracle 12cR1, there were just two types of histograms — frequency-based and height-based — and each offered different advantages for data values that were seriously skewed from a normal distribution.

Listing 1 shows SQL statements to create table APRANDOMIZED_SORTED and related database objects to demonstrate these concepts. The table contains 100,000 rows with a sufficiently random value for one of its columns (KEY_STS) to illustrate how histograms can make a difference in the optimizer’s decisions:

```
DROP TABLE ap.randomized_sorted PURGE;
CREATE TABLE ap.randomized_sorted (  
  key_id NUMBER(8)   ,key_date    DATE   ,key_desc    VARCHAR2(32)   ,key_sts     NUMBER(2) NOT NULL )    TABLESPACE ap_data    NOLOGGING;
------- Populate table so that NDV of KEY_STS column = 17 and with two top values
-- (40 and 50) accounting for 60% of all values
-----
DECLARE  
  ctr NUMBER := 0;
BEGIN
  FOR ctr IN 1..100000
    LOOP
      INSERT INTO ap.randomized_sorted
      VALUES(  ... - DBMS_RANDOM.VALUE(1,3650))               ,LPAD(' ',DBMS_RANDOM.VALUE(1,32), SUBSTR('abcdefghijklmnopqrstuvwxzy',DBMS_RANDOM.VALUE(1,26), 1))
        ,0,10,  1,30,  2,20,  3,10,  4,40
        ,5,11,  6,31,  7,21,  8,11,  9,40
        ,10,12, 11,32, 12,22, 13,12, 14,40
        ,15,13, 16,33, 17,23, 18,13, 19,40
        ,20,14, 21,34, 22,24, 23,14, 24,40
        ,25,20, 26,10, 27,30, 28,20, 29,40
        ,30,21, 31,11, 32,31, 33,21, 34,40
        ,35,22, 36,12, 37,32, 38,22, 39,40
        ,40,23, 41,13, 42,33, 43,23, 44,40
        ,45,24, 46,14, 47,34, 48,24, 49,40, 50)
    IF MOD(ctr, 5000) = 0 THEN
      COMMIT;
    END IF;
  END LOOP;
EXCEPTION    WHEN OTHERS THEN NULL;END;/
-----
ALTER TABLE ap.randomized_sorted DROP CONSTRAINT randomized_sorted_pk;
--- Create primary key index
ALTER TABLE ap.randomized_sorted ADD CONSTRAINT randomized_sorted_pk
  PRIMARY KEY (key_id, key_desc);
--- Add DEFAULT value of (M)ale
UPDATE ap.randomized_sorted
SET key_sts = CASE WHEN key_desc LIKE '%male' THEN 1 ELSE 0 END;
```

continued on page 8
And that’s the first key point about a frequency-based histogram: It’s really nothing more than a frequency distribution — in other words, a tallying of the number of distinct values (NDV) for a given data sample. When the Oracle Database is requested to create a histogram via DBMS_STATS, it will construct a frequency histogram as long as the column's NDV is less than or equal to the number of histogram buckets specified. If the number of buckets (NB) for the histogram isn’t specified via the SIZE parameter of the METHOD_OPT directive, the default value of 254 is assumed. Listing 2.1 shows how to create a frequency-based histogram for the AP.RANDOMIZED_SORTED.KEY_STS column using the DBMS_STATS.GATHER_TABLE_STATS procedure.

Listing 2.1: Creating a Frequency Distribution Histogram

A simple frequency distribution of the values for the KEY_STS column is shown in Figure 1. Note that there are just 17 distinct values (or, in histogram-speak, the number of distinct values (NDV) is 17), but also note that the majority of values (60 percent) for KEY_STS are clustered around just two: 40 and 50.

<table>
<thead>
<tr>
<th>KEY_STS Value</th>
<th>Number of Occurrences</th>
<th>Percent of Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3024</td>
<td>3.0%</td>
</tr>
<tr>
<td>11</td>
<td>3059</td>
<td>3.1%</td>
</tr>
<tr>
<td>12</td>
<td>2967</td>
<td>3.0%</td>
</tr>
<tr>
<td>13</td>
<td>2973</td>
<td>3.0%</td>
</tr>
<tr>
<td>14</td>
<td>2934</td>
<td>2.9%</td>
</tr>
<tr>
<td>20</td>
<td>3028</td>
<td>3.0%</td>
</tr>
<tr>
<td>21</td>
<td>3018</td>
<td>3.0%</td>
</tr>
<tr>
<td>22</td>
<td>3027</td>
<td>3.0%</td>
</tr>
<tr>
<td>23</td>
<td>3024</td>
<td>3.0%</td>
</tr>
<tr>
<td>30</td>
<td>2021</td>
<td>2.0%</td>
</tr>
<tr>
<td>31</td>
<td>2075</td>
<td>2.1%</td>
</tr>
<tr>
<td>32</td>
<td>1959</td>
<td>2.0%</td>
</tr>
<tr>
<td>33</td>
<td>1994</td>
<td>2.0%</td>
</tr>
<tr>
<td>34</td>
<td>2025</td>
<td>2.0%</td>
</tr>
<tr>
<td>40</td>
<td>9950</td>
<td>10.0%</td>
</tr>
<tr>
<td>50</td>
<td>49960</td>
<td>50.0%</td>
</tr>
</tbody>
</table>

Figure 1: Simple Frequency Distribution of AP.RANDOMIZED_SORTED.KEY_STS

Listing 2.2: Frequency-Based Histogram Metadata
Once the frequency-based histogram has been constructed, the optimizer can more intelligently determine the expected cardinality of the row set that will be returned based on the selection criteria specified for the KEY_STS column, especially when considering less popular values other than 40 or 50.

**Height-Based Histograms**

Another type of histogram, the height-based histogram, works quite differently from its frequency-based cousin. It distributes the count of all rows evenly across all histogram buckets, so all buckets will have almost exactly the same number of rows. If the NB is reduced below the NDV for the column — in this example, to anything below a value of 17 — there is obviously no way to capture the frequency distribution of each value for KEY_STS within its own bucket, and a height-based histogram is now the only option.

Creating a height-based histogram in Oracle 12cR1 now requires that a full (100 percent) sample be specified during statistics gathering. Listing 3 shows the code invoked to build a height-based histogram for the same column, as well as the resulting histogram’s metadata.

```sql
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS (    ownname => 'AP',
    .tabname => 'RANDOMIZED_SORTED',
    .method_opt => 'RANDOMIZED_SORTED',
    .sample_size = 100
  );
END;
/
```

### Listing 3: Creating a Height-Based Histogram

Even though 16 buckets were requested during the histogram’s creation, note that some of the buckets — specifically, 7 and 9 through 15 — appear to be missing. This is actually expected behavior in Oracle 12cR1, and it reflects that all values in those buckets contain the same range of values as the next highest bucket. For example, even though buckets 9 through 16 will each contain approximately 6.25 percent of all rows in a height-based histogram, those eight buckets contain exactly 50 percent (16* 6.25 = 50) of all the rows with a value of 50 for the column KEY_STS. So the optimizer only needs to care about buckets with different values — 8 and 16 — to make an accurate prediction about the cardinality for that value.

### Needed: A New Breed of Histograms

So which of these two histograms is the best choice? Prior to Oracle 12cR1, the answer to this question was: It really depends on the column’s NDV. A histogram has an internal limit of 254 buckets, so if a column’s NDV exceeded 254, DBMS_STATS simply had no other choice but to create a height-based histogram.

If a column’s NDV was less than 254, a DBA could either allow DBMS_STATS to derive the NDV automatically, or attempt to override the actual NDV by setting it to a lower value. However, unless the DBA was really savvy about the rules of precedence that the optimizer uses when accessing a histogram and deeply understood the impact of manually reducing the number of buckets (NB) for a histogram, it was probably best to let DBMS_STATS make the decision as to which histogram was most appropriate.

To add to the complexity of histogram behavior prior to 12cR1, two special cases often hampered the usefulness of frequency-based and height-based histograms:

- **NDV > 254 Plus Extremely Popular Values:** When the NDV for a column exceeded 254 values, a height-based histogram was created, as noted above. However, if a significant proportion of the buckets contained values that tended to be more popular than other values, the resulting histogram would be of little use to help the optimizer discern them from truly unpopular values.

- **“Bucket-Spanning” Popular Values:** Another issue with height-balanced histograms is that a truly popular value might just miss spanning two full histogram buckets. In this case, the optimizer would not recognize the value as truly popular and may choose an access path, join order or join method that is significantly inefficient.

The good news is that Oracle 12cR1 implements two new histograms that are designed to overcome just these situations:

- **Top Frequency Histograms** are most useful whenever a small number of distinct values dominate the majority of distinct values. This histogram is gathered using a full table scan of a table, and the appropriate number of histogram buckets is automatically chosen.

- **Hybrid Histograms** combine the best features of height-based and frequency-based histograms because not only do they store the endpoints of each histogram bucket, but also the frequency of NDVs within those buckets, recorded in the ENDPOINT_REPEAT_COUNT column of DBA_HISTOGRAMS. This histogram is most useful for producing more accurate cardinality estimates than height-based histograms, especially when values that are almost popular dominate multiple histogram buckets.

Interestingly, the height-balanced histogram can still be generated through DBMS_STATS, but only if a 100 percent sample size is chosen, as demonstrated in Listing 4. In fact, Oracle 12cR1 documentation now refers to height-based histograms as “legacy” histograms, which may imply they will be deprecated in a future release.

Figure 2 shows the decision tree that DBMS_STATS uses to determine which type of histogram should be generated.

The good news is that Oracle 12cR1 implements two new histograms that are designed to overcome just these situations:

- **Top Frequency Histograms** are most useful whenever a small number of distinct values dominate the majority of distinct values. This histogram is gathered using a full table scan of a table, and the appropriate number of histogram buckets is automatically chosen.

- **Hybrid Histograms** combine the best features of height-based and frequency-based histograms because not only do they store the endpoints of each histogram bucket, but also the frequency of NDVs within those buckets, recorded in the ENDPOINT_REPEAT_COUNT column of DBA_HISTOGRAMS. This histogram is most useful for producing more accurate cardinality estimates than height-based histograms, especially when values that are almost popular dominate multiple histogram buckets.

Interestingly, the height-balanced histogram can still be generated through DBMS_STATS, but only if a 100 percent sample size is chosen, as demonstrated in Listing 4. In fact, Oracle 12cR1 documentation now refers to height-based histograms as “legacy” histograms, which may imply they will be deprecated in a future release.

Figure 2 shows the decision tree that DBMS_STATS uses to determine which type of histogram should be generated.

The good news is that Oracle 12cR1 implements two new histograms that are designed to overcome just these situations:

- **Top Frequency Histograms** are most useful whenever a small number of distinct values dominate the majority of distinct values. This histogram is gathered using a full table scan of a table, and the appropriate number of histogram buckets is automatically chosen.

- **Hybrid Histograms** combine the best features of height-based and frequency-based histograms because not only do they store the endpoints of each histogram bucket, but also the frequency of NDVs within those buckets, recorded in the ENDPOINT_REPEAT_COUNT column of DBA_HISTOGRAMS. This histogram is most useful for producing more accurate cardinality estimates than height-based histograms, especially when values that are almost popular dominate multiple histogram buckets.

Interestingly, the height-balanced histogram can still be generated through DBMS_STATS, but only if a 100 percent sample size is chosen, as demonstrated in Listing 4. In fact, Oracle 12cR1 documentation now refers to height-based histograms as “legacy” histograms, which may imply they will be deprecated in a future release.

Figure 2 shows the decision tree that DBMS_STATS uses to determine which type of histogram should be generated.
Figure 2: Decision Tree for Oracle 12cR1 Histogram Types

Top Frequency Histograms

Listing 4.1 demonstrates how to create a top frequency histogram for the KEY_STS column as well as the resulting metadata produced for this new histogram type.

Listing 4.1: Creating a Top Frequency Histogram

BEGIN
    DBMS_STATS.GATHER_TABLE_STATS (       
        ownname => 'AP', 
        tablename => 'RANDOMIZED_SORTED', 
        method_opt => 'FOR COLUMNS KEY_STS SIZE 16'
    );
END;
/

Histogram Metadata
for AP.RANDOMIZED_SORTED.KEY_STS
(from DBA_TAB_COL_STATISTICS)

<table>
<thead>
<tr>
<th>Distinct Histogram</th>
<th># of Values</th>
<th># of Buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOP-FREQUENCY</td>
<td>17</td>
<td>16</td>
</tr>
</tbody>
</table>

Histogram Endpoints
for AP.RANDOMIZED_SORTED.KEY_STS
(from DBA_HISTOGRAMS)

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Repeat</th>
<th>Value</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>3024</td>
<td>10</td>
<td>3024</td>
<td>10</td>
</tr>
<tr>
<td>6083</td>
<td>11</td>
<td>6083</td>
<td>11</td>
</tr>
<tr>
<td>9050</td>
<td>12</td>
<td>9050</td>
<td>12</td>
</tr>
<tr>
<td>12023</td>
<td>13</td>
<td>12023</td>
<td>13</td>
</tr>
<tr>
<td>14957</td>
<td>14</td>
<td>14957</td>
<td>14</td>
</tr>
<tr>
<td>17985</td>
<td>15</td>
<td>17985</td>
<td>15</td>
</tr>
<tr>
<td>21003</td>
<td>16</td>
<td>21003</td>
<td>16</td>
</tr>
<tr>
<td>24030</td>
<td>17</td>
<td>24030</td>
<td>17</td>
</tr>
<tr>
<td>27054</td>
<td>18</td>
<td>27054</td>
<td>18</td>
</tr>
<tr>
<td>30016</td>
<td>19</td>
<td>30016</td>
<td>19</td>
</tr>
<tr>
<td>32037</td>
<td>20</td>
<td>32037</td>
<td>20</td>
</tr>
<tr>
<td>34112</td>
<td>21</td>
<td>34112</td>
<td>21</td>
</tr>
<tr>
<td>36106</td>
<td>22</td>
<td>36106</td>
<td>22</td>
</tr>
<tr>
<td>38131</td>
<td>23</td>
<td>38131</td>
<td>23</td>
</tr>
<tr>
<td>48081</td>
<td>24</td>
<td>48081</td>
<td>24</td>
</tr>
<tr>
<td>98041</td>
<td>25</td>
<td>98041</td>
<td>25</td>
</tr>
</tbody>
</table>

Listing 4.2: Using a Frequency Histogram.

Plan hash value: 2690174173

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (CPU)</th>
<th>Time</th>
<th>TQ</th>
<th>IN-OUT</th>
<th>PQ Distrib</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>3</td>
<td>55 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PX COORDINATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PX SEND QC (RANDOM)</td>
<td>:T010000</td>
<td>1</td>
<td>3</td>
<td></td>
<td>Q1,00</td>
<td>P-&gt;S</td>
<td>QC RAND</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>3</td>
<td></td>
<td>Q1,00</td>
<td>P-&gt;S</td>
<td>QC RAND</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PX BLOCK ITERATOR</td>
<td>50041 144K</td>
<td>55 (0)</td>
<td>00:00:01</td>
<td>01,00</td>
<td>PWQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 6</td>
<td>INDEX FAST FULL SCAN</td>
<td>RANDOMIZED_SORTED_STS</td>
<td>50041 144K</td>
<td>55 (0)</td>
<td>00:00:01</td>
<td>01,00</td>
<td>PWQ</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

6 - filter("KEY_STS"<40)
Note that, as expected, the optimizer now leverages the top frequency histogram to correctly capture the significance of the almost popular value (40) while ignoring the least-significant value (32) for KEY_STS.

**Hybrid Histograms**

While frequency-based histograms are certainly valuable, they still have one shortfall: They tend to over-estimate the cardinality of a row set when a value’s frequency made that value not the most popular value, but an almost popular value. The frequency distribution in Figure 1 shows the value of 40 for KEY_STS fits this description because even though it has occurred only about 10 percent of the time, it’s not the most popular value when compared to a frequency of 50 percent for the value of 50.

In this case, a hybrid histogram can make a difference because while it still records an endpoint for each bucket’s endpoint value, it also tallies an endpoint repeat count of the number of values within each bucket. To illustrate this, Listing 5.1 shows the creation of a hybrid histogram for the KEY_STS column and its resulting metadata.

```
BEGIN      DBMS_STATS.GATHER_TABLE_STATS (         ownname     =>  'AP'       ,tabname     =>  'RANDOMIZED_SORTED' ,method_opt  => 'FOR COLUMNS KEY_STS SIZE 13');
END;
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
<th>TO</th>
<th>IN-OUT</th>
<th>PQ Distrib</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>3</td>
<td>55 (0)</td>
<td>00:00:01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>P-&gt;S</td>
<td>QC (RAND)</td>
</tr>
<tr>
<td>2</td>
<td>PX COORDINATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PX SEND QC (RANDOM)</td>
<td>:TQ10000</td>
<td>1</td>
<td>3</td>
<td>Q1,00</td>
<td>01:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>3</td>
<td>Q1,00</td>
<td>01:00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PX BLOCK ITERATOR</td>
<td></td>
<td>48082</td>
<td>140K</td>
<td></td>
<td></td>
<td>Q1,00</td>
<td>PCWP</td>
<td></td>
</tr>
<tr>
<td>* 6</td>
<td>INDEX FAST FULL SCAN</td>
<td>RANDOMIZED_SORTED_STS</td>
<td>48082</td>
<td>140K</td>
<td></td>
<td></td>
<td>Q1,00</td>
<td>PCWP</td>
<td></td>
</tr>
</tbody>
</table>

Predicate Information (identified by operation id):

6 - filter("KEY_STS"<=40)

Listing 4.3 Leveraging a Top Frequency Histogram

Oracle 12cR1 will automatically create a top frequency histogram when it detects the following conditions are fulfilled:

- **NB < NDV**: The number of buckets specified (via the SIZE parameter of the METHOD_OPT argument) is less than the number of distinct values.
- **Sampling Size**: The sample size (defined by the SAMPLE_SIZE directive of DBMS_STATS) must be left at its default value of AUTO_SAMPLE_SIZE.
- **TFTP Satisfied**: Finally, the top frequency threshold percentage (TFTP) for all buckets must be less than the percentage of all rows remaining in the top NB number of buckets.
  - The formula for TFTP is simple: 1 - ( 1 / NB ). Using the example above, TFTP is calculated as (1–(1/16)), or 93.8 percent.
  - Because the top 16 buckets contain 98,041 entries (100,000 – 1,959 for the KEY_STS bucket for the value of 32), the top 16 buckets contain 98 percent of all entries.
  - Since the top 16 buckets contain 98 percent of all entries, and this exceeds the TFTP OF 93.8 percent, DBMS_STATS creates a top frequency histogram.

This simple illustration is fine for a column with a NDV less than the built-in histogram limit of 254, but what about a column whose NDV exceeds 254? In this situation, a top frequency histogram becomes even more valuable because it focuses the optimizer on the most popular distinct values by ignoring the least popular distinct values beyond the 254-bucket limit of histograms.

To prove the value of the top frequency histogram, here are two EXPLAIN PLANS for a simple query against AP.RANDOMIZED_SORTED with selection criteria against the KEY_STS column.

```
SELECT COUNT(key_sts) FROM ap.randomized_sorted
WHERE key_sts IN (13,40,50);
```

Note that, as expected, the optimizer now leverages the top frequency histogram to correctly capture the significance of the almost popular value (40) while ignoring the least-significant value (32) for KEY_STS.

**Hybrid Histograms**

While frequency-based histograms are certainly valuable, they still have one shortfall: They tend to over-estimate the cardinality of a row set when a value’s frequency made that value not the most popular value, but an almost popular value. The frequency distribution in Figure 1 shows the value of 40 for KEY_STS fits this description because even though it has occurred only about 10 percent of the time, it’s not the most popular value when compared to a frequency of 50 percent for the value of 50.

In this case, a hybrid histogram can make a difference because while it still records an endpoint for each bucket’s endpoint value, it also tallies an endpoint repeat count of the number of values within each bucket. To illustrate this, Listing 5.1 shows the creation of a hybrid histogram for the KEY_STS column and its resulting metadata.

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS ( ownname => 'AP' ,
                                  tablename => 'RANDOMIZED_SORTED',
                                  method_opt => 'FOR COLUMNS KEY_STS SIZE 13');
END;
```

<table>
<thead>
<tr>
<th>Histogram Metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Distinct Values</td>
</tr>
<tr>
<td>HYBRID</td>
</tr>
</tbody>
</table>

**Continued on page 14**
TFTP Not Satisfied: When the TFTP for all buckets exceeds the percentage of all rows remaining in the top NB number of buckets, then the rule for top frequency histograms is broached.

- Using the example above, TFTP is calculated as \((1-(1/13))\), or 92.3 percent.
- Because the top 13 buckets contain just 92,001 entries (100,000 – 7,999 for the KEY_STS buckets containing values 34, 30, 33 and 32), they comprise only 92 percent of all entries.
- Since the top 13 buckets contain 92 percent of all entries, and this falls below the TFTP of 92.3 percent, DBMS_STATS creates a hybrid histogram instead of a top frequency histogram.

To prove the value of the hybrid histogram, here are two EXPLAIN PLANs for a similar query against AP_RANDOMIZED_SORTED, but with slightly different selection criteria against the KEY_STS column.

Listing 5.1: Creating a Hybrid Histogram

Oracle 12cR1 creates a hybrid histogram when it detects the following conditions are fulfilled:

- **NB < NDV:** Again, the number of buckets specified must be less than the number of distinct values.
- **Sampling Size:** Again, the sample size (defined by the SAMPLE_SIZE directive of DBMS_STATS) must be left at its default value of AUTO_SAMPLE_SIZE.

Listing 5.2 shows the cardinality estimates using the height-based histogram created in Listing 3, while Listing 5.3 shows the same statement's EXPLAIN PLAN when the hybrid histogram created in Listing 5.1 is used instead:

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Value</th>
<th>Repeat</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>161</td>
<td>10</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>507</td>
<td>12</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>836</td>
<td>14</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>1132</td>
<td>21</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>1278</td>
<td>22</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>1459</td>
<td>23</td>
<td>181</td>
<td></td>
</tr>
<tr>
<td>1641</td>
<td>24</td>
<td>182</td>
<td></td>
</tr>
<tr>
<td>1760</td>
<td>30</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>1871</td>
<td>31</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>1967</td>
<td>32</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>2116</td>
<td>33</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>2218</td>
<td>34</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>5503</td>
<td>50</td>
<td>2768</td>
<td></td>
</tr>
</tbody>
</table>

Listing 5.2: Without a Hybrid Histogram

Listing 5.3: Leveraging a Hybrid Histogram
Note that, as expected, the optimizer now leverages the hybrid histogram to somewhat underestimate the number of rows (3,106) that will be retrieved for the equality search for a KEY_STS value of 40, versus overestimating the number of rows (12,500) via the standard height-balanced histogram.

**Conclusions**

Even though the sample set used to demonstrate them is rather trivial in size, it's easy to see that the new top frequency and hybrid histograms in Oracle 12cR1 offer some significant improvements. They fill in the gaps for the optimizer when it needs to determine the best methods to filter columns with skewed distributions of data by reducing potentially incorrect cardinality estimates, especially when a column’s NDV is overpopulated with either almost-popular or extremely popular values.

**What’s Next for 12cR1?**

In next issue's column, I’ll complete my review of Oracle 12cR1 SQL performance enhancements as I investigate and demonstrate some significant improvements to SQL Plan Management (SPM), automatic dynamic sampling and extended statistics in Oracle 12cR1.

---

**About the Author**

Jim Czuprynski has accumulated more than 30 years of experience during his career in information technology. He has served diverse roles at several Fortune 1000 companies in those three decades — mainframe programmer, applications developer, business analyst and project manager — before becoming an Oracle Database administrator in 2001. He currently holds OCP certification for Oracle 9i, 10g and 11g. Jim teaches the core Oracle University database administration courses on behalf of Oracle and its education partners throughout the United States and Canada, instructing several hundred Oracle DBAs since 2005. He was selected as Oracle Education Partner Instructor of the Year in 2009. He continues to write a steady stream of articles that focus on the myriad facets of Oracle Database administration, with nearly 100 articles to his credit since 2003 at databasejournal.com. Jim’s monthly blog, Generally … It Depends (http://jimczuprynski.wordpress.com), contains his regular observations on all things Oracle. Jim is also a regular public speaker on Oracle Database technology features, and has presented topics at Oracle OpenWorld (2008 and 2013), IOUG's COLLABORATE conferences (2011 and 2013) and OUG Norway (2013). Be sure to join Jim when he presents topics at COLLABORATE 14 in Las Vegas.

---

**Submit an Article to IOUG**

**SELECT Journal** is IOUG’s Quarterly Publication

We are always looking for new authors and articles for 2014.

Interested in submitting an article? Visit www.ioug.org and click on Publications > SELECT Journal for more information. Questions? Contact SELECT Journal Managing Editor Alexa Schlosser at (312) 673-5791, or email her at aschlosser@ioug.org.

**IOUG Is Looking for New Materials for the 2014 Best Practices Booklet**

Submissions should be 500-1,000 words long; due to space constraints, we ask that your submission have a specific focus as opposed to any overarching database principles. Tips can range from beginning- to advanced-level skills and should include the actual code and queries used (screenshots and other small graphics are also acceptable).

If you have any questions about this project, please contact our Best Practices Booklet Managing Editor Alexa Schlosser, at (312) 673-5791, or email her at aschlosser@ioug.org.