Database statistics gathering: Synopsis

Introduction

It is known that having proper database statistics is crucial for query optimizer. Statistics should properly describe data within the database. To gather statistics efficiently and have correct statistics is not an easy process. Several algorithms have been used to accomplish this. Until Oracle 11g, the database gathered statistics based on a sample, in an iterative method. In this process, the database started with a small sample size and then, based on some analysis, it would determine whether the sample size was sufficiently large. So the database first would need to determine the optimal sample size and then scan the database objects to compute object statistics using the sample. But, if an object was very large then the sample size deemed optimal would actually be too small and of course the computed statistics would not be accurate. In addition, such an approach required rescanning all partitions in order to approximate a global level of object statistics for partitioned tables. For non-partitioned tables the database had to perform a sort operation to generate a number of distinct values (NDV) for the columns – this is a resource-intensive operation.

To improve the quality of database statistics and the process of gathering them, in Oracle 11g the one-pass distinct sampling algorithm was introduced, which allows us to solve the above-mentioned problems. The approach has been refined in Oracle 12c R2 via implementing the Hyperloglog algorithm. To support this mechanism a special data structure called 'synopsis' is introduced within the database. It contains information about the objects (i.e., columns of tables, partitions) that helps approximate the required statistics efficiently.

Synopsis Implementation

Synopsis is a data structure that describes a database object (table, partition). Synopsis helps to estimate important object statistics like NDV. To construct a synopsis a uniform hash function (for example, 64 bit hash function) is used, which maps column values to hash values that are stored in the synopsis. The uniform hash function means that each column value has an approximately equal probability of mapping to any synopsis value. Initially, the synopsis is empty; the database starts to scan the object (table, partition) and picks up the column values $a_1, a_2, a_3 \ldots\ldots a_n$; then the uniform hash function is applied to the column values and generates $h(a_i) = h_i\ (bits\ of\ hash\ values)$. If the synopsis does not contain this hash value then it is added to the synopsis. If the hash value is already stored in the synopsis then the database proceeds to read another column value. The synopsis has a storage limit. When the synopsis reaches its capacity, its size is reduced by half by discarding all hash values that have "1" in any of their leading "i" bits. This is called splitting the synopsis and "i" is the number of times the synopsis was split. This process is continued until the database reads all column values. So the synopsis contains hash values (bits of hash values) and a number of splits. This information is enough to estimate the NDV of the column. So, according to the algorithm:

$$NDV \approx N * 2^{i} \text{ (Formula 1)}$$

Here N is number of distinct values in the synopsis. This approach is one-pass distinct sampling and Oracle implemented this algorithm in version 11g. In Oracle 12c r2 it is called Adaptive Sampling (AS). But in Oracle 12c R2 the mechanism has been improved via the Hyperloglog (HLL) algorithm (https://en.wikipedia.org/wiki/HyperLogLog). HLL uses randomization to approximate the NDV and this is achieved by applying the uniform hash function as described for one-pass distinct sampling. The algorithm observes the maximum number of leading zeros that occur for all hash values. But, hash values with more leading zeros are less likely and will indicate a larger NDV. And in this case, the estimation error will be large. To minimize the estimation error the given object (table, partition) is divided into sub parts (approximately of equal size using the first p bits of the hash values, where $\mathrm{m=power(2,p)}$) called buckets (B_i) But the same hash function is used for them. In each bucket, the maximum number of leading zeros is calculated. These numbers are stored in an array M, where M[i] stores the maximum number of leading zeros plus one for the bucket with the index “i”. So:

$$\mathrm{M[i]:=}\max Q(x)\ x \in B_i$$

…where $Q(x)$ is a function that returns the number of leading zeros in the binary representation of x plus one. According to the algorithm the NDV will be:

$$\mathrm{E=}\ \alpha_m * m^2 * \left(\sum_{j=1}^{m} 2^{-Mj}\right)^{-1} \quad \alpha_{16} = 0.673; \alpha_{32} = 0.697; \alpha_{64} = 0.709$$

For more information you can refer to the article (https://hal.archives-ouvertes.fr/file/index/docid/406166/filename/FlFuGaMe07.pdf).

A synopsis based on AS is referred to as “old-style”, and one based on HLL is referred to as a new-style synopsis. Let’s investigate how Oracle manages and use synopses. First we are going to see the HLL method.

```
CREATE TABLE sh.sales1
    (prod_id                           NUMBER NOT NULL,
    cust_id                           NUMBER NOT NULL,
    time_id                           DATE NOT NULL,
    channel_id                        NUMBER NOT NULL,
    promo_id                          NUMBER NOT NULL,
    quantity_sold                     NUMBER(10,2) NOT NULL,
    amount_sold                       NUMBER(10,2) NOT NULL)
     PARTITION BY RANGE (TIME_ID)
  (
  PARTITION sales_q1_1998 VALUES LESS THAN (TO_DATE(' 1998-04-01 00:00:00',
'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')),
  PARTITION sales_q2_1998 VALUES LESS THAN (TO_DATE(' 1998-07-01 00:00:00',
'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
   )
/


insert /*+ append parallel(4)*/into sh.sales1
select * from (select * from sh.sales where
time_id<=to_date(‘6/30/1998’,’mm/dd/yyyy’)

---Performed several insert
insert /*+ append parallel(4)*/into sh.sales1
select * from sh.sales1;
commit;

BEGIN
    DBMS_STATS.set_table_prefs (ownname   => 'sh',
                                tabname   => 'sales1',
                                pname     => 'approximate_ndv_algorithm',
                                pvalue    => 'hyperloglog');

    DBMS_STATS.set_table_prefs ('sh',
                                'sales1',
                                'INCREMENTAL',
                                'TRUE');
END;
```

The table size is **5.5G**. So, we can gather table statistics (and it can be traced).

```
BEGIN
    DBMS_STATS.set_global_prefs ('trace', TO_CHAR (2048 + 32768 + 4 + 16));
    DBMS_STATS.gather_table_stats (ownname => 'sh', tabname => 'SALES1');
END;
```

From the DBMS_STATS trace file we could see the following lines:

```
DBMS_STATS: gather stats on partition SALES_Q2_1998: synopsis not gathered
yet; not analyzed yet;
DBMS_STATS: Start gather_stats.. pfix:  ownname: SH tabname: SALES1 pname:
SALES_Q2_1998 spname:  execution phase: 1
DBMS_STATS: APPROX_NDV_ALGORITHM chosen: HLL (incremental)
DBMS_STATS: reporting_man_log_task: target: SH.SALES1.SALES_Q2_1998 objn:
76755 auto_stats: FALSE status: IN PROGRESS ctx.batching_coeff: 0
```

It seems Oracle started to gather statistics for one partition (`SALES_Q2_1998`) of the table. And the following lines indicate that the database was going to gather the mentioned statistics for the columns using special SQL.

```
DBMS_STATS: no AS synopses to delete for #76753
DBMS_STATS: Using approximate NDV pct=0
DBMS_STATS:  NNV  NDV  AVG  MMX  HST  EP   RP   NNNP IND  CNDV HSTN HSTR
COLNAME
DBMS_STATS:        Y    Y    Y
PROD_ID
```

```
DBMS_STATS:        Y    Y    Y
CUST_ID
DBMS_STATS:        Y         Y
TIME_ID
DBMS_STATS:        Y    Y    Y
CHANNEL_ID
DBMS_STATS:        Y    Y    Y
PROMO_ID
DBMS_STATS:        Y    Y    Y
QUANTITY_SOLD
DBMS_STATS:        Y    Y    Y
AMOUNT_SOLD
DBMS_STATS: APPROX_NDV_ALGORITHM chosen: HLL (incremental)
DBMS_STATS: Approximate NDV Options
DBMS_STATS:
SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,N
IL,NIL, NDV_HLL, B76753
DBMS_STATS: Starting query at 29-NOV-17 11.56.57.948197000 AM +04:00
DBMS_STATS: select /*+  full(t)    no_parallel(t) no_parallel_index(t)
dbms_stats cursor_sharing_exact use_weak_name_resl dynamic_sampling(0)
no_monitoring xmlindex_sel_idx_tbl opt_param('optimizer_inmemory_aware'
'false') no_substrb_pad
*/to_char(count("PROD_ID")),substrb(dump(min("PROD_ID"),16,0,64),1,240),subst
rb(dump(max("PROD_ID"),16,0,64),1,240),to_char(count("CUST_ID")),substrb(dump
(min("CUST_ID"),16,0,64),1,240),substrb(dump(max("CUST_ID"),16,0,64),1,240),t
o_char(count("TIME_ID")),substrb(dump(min("TIME_ID"),16,0,64),1,240),substrb(
dump(max("TIME_ID"),16,0,64),1,240),to_char(count("CHANNEL_ID")),substrb(dump
(min("CHANNEL_ID"),16,0,64),1,240),substrb(dump(max("CHANNEL_ID"),16,0,64),1,
240),to_char(count("PROMO_ID")),substrb(dump(min("PROMO_ID"),16,0,64),1,240),
substrb(dump(max("PROMO_ID"),16,0,64),1,240),to_char(count("QUANTITY_SOLD")),
substrb(dump(min("QUANTITY_SOLD"),16,0,64),1,240),substrb(dump(max("QUANTITY_
SOLD"),16,0,64),1,240),to_char(count("AMOUNT_SOLD")),substrb(dump(min("AMOUNT
_SOLD"),16,0,64),1,240),substrb(dump(max("AMOUNT_SOLD"),16,0,64),1,240) from
"SH"."SALES1" t  where TBL$OR$IDX$PART$NUM("SH"."SALES1",0,4,0,"ROWID") =
:objn /*
SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,N
IL,NIL, NDV_HLL, B76753*/
```

Using the same approach the database computed the column statistics for the second partition – SALES_Q1_1998. Then, finally, Oracle calculated the global statistics for the partitioned table by aggregating the partition-level statistics. Oracle did it by merging synopses. We can see it clearly from the trace file.

```
DBMS_STATS: Number of rows in the table = 89470976, blocks = , average row
length = 29, chain count = , scan rate = 0, sample size = 89470976
DBMS_STATS: prepare reporting structures...
DBMS_STATS: reporting_man_update_task: objn: 76754 auto_stats: FALSE status:
COMPLETED ctx.batching_coeff: 0
DBMS_STATS: Start gather_stats.. pfix:  ownname: SH tabname: SALES1 pname:
spname:  execution phase: 1
DBMS_STATS: APPROX_NDV_ALGORITHM chosen: HLL (incremental)
DBMS_STATS: APPROX_NDV_ALGORITHM chosen: HLL (incremental)
DBMS_STATS: Synopsis Aggregation Degree: 1
DBMS_STATS: APPROX_NDV_ALGORITHM chosen: HLL (incremental)
```

```
DBMS_STATS: get_agg_colstats: HLL only
DBMS_STATS: Derive global stats from partition synopses/stats for table
SALES1.
```

So, if there are statistics for table partitions then it is enough to compute global statistics by merging the synopses of appropriate partitions. Starting with Oracle 12c R2 the synopsis (which is created based on HLL algorithm) is stored in the **WRI$_OPTSTAT_SYNOPSIS_HEAD$** table.

```
SQL> desc WRI$_OPTSTAT_SYNOPSIS_HEAD$
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------------------
 BO#                                       NOT NULL NUMBER
 GROUP#                                    NOT NULL NUMBER
 INTCOL#                                   NOT NULL NUMBER
 SYNOPSIS#                                          NUMBER
 SPLIT                                              NUMBER
 ANALYZETIME                                        DATE
 SPARE1                                             NUMBER
 SPARE2                                             BLOB
```

The column BO# is equal to **dba_objects.object_id**. Then

```
SQL> SELECT   bo#,
         group#,
         intcol#,
         synopsis#,
         split,
         spare1,
         DBMS_LOB.SUBSTR (spare2, 10) spare2
  FROM   wri$_optstat_synopsis_head$
 WHERE   bo# IN (SELECT   object_id
                   FROM   dba_objects
                  WHERE   object_name = 'SALES1');

       BO#     GROUP#    INTCOL#  SYNOPSIS#      SPLIT     SPARE1 SPARE2
---------- ---------- ---------- ---------- ---------- ---------- --------------------
     76753     153510          1                     0          1 0D0C00B7003100000000
     76753     153510          2                     0          1 0D0C000107F600000000
     76753     153510          3                     0          1 0D0C001C005B00000000
     76753     153510          4                     0          1 0D0C0303000400000000
     76753     153510          5                     0          1 0D0C01F6000200000000
     76753     153510          6                     0          1 0D0C00E9000100000000
     76753     153510          7                     0          1 0D0C000E00D600000000
     76753     153508          1                     0          1 0D0C009B003C00000000
     76753     153508          2                     0          1 0D0C000108A000000000
     76753     153508          3                     0          1 0D0C0005005A00000000
     76753     153508          4                     0          1 0D0C0303000400000000
     76753     153508          5                     0          1 0D0C01F6000200000000
     76753     153508          6                     0          1 0D0C00E9000100000000
     76753     153508          7                     0          1 0D0C0004017C00000000
```

We have a partitioned table with two partitions. Also, the table has 7 columns and therefore we have 7*2=14 synopses in the dictionary. In my understanding, the descriptions of the above columns are:

BO# – object id of the table
GROUP# – object id of the partition (half of group# is equal to the object id of the partition)
INTCOL# – column number (position, that refers to sys.col$.col#)
SPLIT – number of splits performed for the synopsis
SPARE1 – this column has value “1” if the synopsis is created based on the HLL algorithm
SPARE2 – this column contains hashed values (synopsis values) that were generated by applying HyperLogLog or Adaptive Sampling to the corresponding column values.

So, in our case we have the synopsis and its properties: hash values and the number of splits. It means these values are sufficient to calculate the NDV for the columns at the local or global level.

When working with results from approximate queries that contain aggregate functions, it is difficult to approximate a result across various dimensions. We cannot use an aggregated approximate result as a basis for the next, higher-level dimensions of the query. In this case we would have to rescan the table(s) to compute approximately for the given dimensions. But, in Oracle 12c R2 the following new functions have been introduced that help us to solve this problem and these also allow us to compute the NDV by aggregating the hash values of the synopsis.

approx_count_distinct_detail – returns information about the approximate number of rows. This is a special format and it produces as a blob.

approx_count_distinct_agg – This function creates a higher level of the summary based on the results from approx_count_distinct_detail. It allows us to avoid rescan of the base table in order to get new aggregates.

to_approx_count_distinct – This function returns the result from the above functions as number.

Now we can check the column statistics from the dictionary and can compare them with the result of using the above-mentioned functions that are going to be applied to the synopsis.

```
  SELECT   partition_name, column_name, num_distinct
    FROM   dba_part_col_statistics
   WHERE   owner = 'SH' AND table_name = 'SALES1'
ORDER BY   1, 2;
```

```
PARTITION_NAME  COLUMN_NAME        NUM_DISTINCT
--------------- --------------    ---------------
SALES_Q1_1998   AMOUNT_SOLD        398
SALES_Q1_1998   CHANNEL_ID         4
SALES_Q1_1998   CUST_ID            3172
SALES_Q1_1998   PROD_ID            60
SALES_Q1_1998   PROMO_ID           2
SALES_Q1_1998   QUANTITY_SOLD      1
SALES_Q1_1998   TIME_ID            91
```

```
SALES_Q2_1998   AMOUNT_SOLD        219
SALES_Q2_1998   CHANNEL_ID         4
SALES_Q2_1998   CUST_ID            2819
SALES_Q2_1998   PROD_ID            49
SALES_Q2_1998   PROMO_ID           2
SALES_Q2_1998   QUANTITY_SOLD      1
SALES_Q2_1998   TIME_ID            92
```

Let’s query from synopsis data.

```
  SELECT   subobject_name part_name, name colname, ndv
    FROM   (  SELECT   group#,
                       intcol#,
                       to_approx_count_distinct (
                           approx_count_distinct_agg (spare2))
                           ndv
                FROM   wri$_optstat_synopsis_head$
               WHERE   bo# = 76753 --This is the object id of the partitioned
table
            GROUP BY   group#, intcol#) s,
           sys.col$ c,
           dba_objects o
   WHERE   c.obj# = 76753 AND c.col# = s.intcol# AND o.object_id = s.group# /
2
ORDER BY   1, 2;
```

```
PART_NAME       COL_NAME          NDV
--------------- --------------    -------
SALES_Q1_1998   AMOUNT_SOLD        398
SALES_Q1_1998   CHANNEL_ID         4
SALES_Q1_1998   CUST_ID            3172
SALES_Q1_1998   PROD_ID            60
SALES_Q1_1998   PROMO_ID           2
SALES_Q1_1998   QUANTITY_SOLD      1
SALES_Q1_1998   TIME_ID            91
SALES_Q2_1998   AMOUNT_SOLD        219
SALES_Q2_1998   CHANNEL_ID         4
SALES_Q2_1998   CUST_ID            2819
SALES_Q2_1998   PROD_ID            49
SALES_Q2_1998   PROMO_ID           2
SALES_Q2_1998   QUANTITY_SOLD      1
SALES_Q2_1998   TIME_ID            92
```

As you see, both of the above queries return exactly the same result. Oracle actually implemented the HLL algorithm on above mentioned **`approx_*`** function and derives partition and global level statistics via applying that function to the synopsis data. Let’s get table level statistics.

```
SELECT   ds.column_name, ds.num_distinct, s.ndv
  FROM   (  SELECT   intcol#,
                     to_approx_count_distinct (
                         approx_count_distinct_agg (spare2))
                         ndv
              FROM   wri$_optstat_synopsis_head$
             WHERE   bo# = 76753 --This is the object id of the partitioned
table
          GROUP BY   intcol#) s,
         sys.col$ c,
         dba_tab_col_statistics ds
 WHERE       c.obj# = 76753
         AND c.col# = s.intcol#
         AND ds.table_name = 'SALES1'
         AND c.name = ds.column_name
```

```
COLUMN_NAME       NUM_DISTINCT  NDV
----------------  ------------ ----------
PROD_ID           60            60
QUANTITY_SOLD     1             1
CUST_ID           4276          4276
CHANNEL_ID        4             4
PROMO_ID          2             2
TIME_ID           183           183
AMOUNT_SOLD       428           428
```

As we see, the values for the **NUM_DISTINCT** column are selected from the **DBA_TAB_COLSTATISTICS** view as column statistics, but for **NDV** are derived from the synopsis. They are equal to each other.

Now let’s interpret the adaptive sampling algorithm (AS).

```
execute DBMS_STATS.delete_table_stats('sh','SALES1');

BEGIN
    DBMS_STATS.set_table_prefs (ownname   => 'sh',
                                tabname   => 'sales1',
                                pname     => 'approximate_ndv_algorithm',
                                pvalue    => 'ADAPTIVE SAMPLING');
    DBMS_STATS.gather_table_stats (ownname => 'sh', tabname => 'SALES1');
END;
```

As in the Hyperloglog algorithm, when Oracle uses AS to compute column statistics it first gathers partition-level statistics separately and then, finally, to compute global-level statistics, Oracle uses synopsis.

```
DBMS_STATS: gather stats on partition SALES_Q2_1998: synopsis not gathered yet; not
analyzed yet;
DBMS_STATS: Start gather_stats.. pfix:  ownname: SH tabname: SALES1 pname:
SALES_Q2_1998 spname:  execution phase: 1
DBMS_STATS: APPROX_NDV_ALGORITHM chosen: AS
DBMS_STATS: reporting_man_log_task: target: SH.SALES1.SALES_Q2_1998 objn: 76755
auto_stats: FALSE status: IN PROGRESS ctx.batching_coeff: 0
```

This indicates that Oracle started to gather statistics for the `SALES_Q2_1998` partition and it used the following SQL statement in order to compute stats.

```
SELECT /*+  full(t)    no_parallel(t) no_parallel_index(t) dbms_stats
cursor_sharing_exact use_weak_name_resl dynamic_sampling(0) no_monitoring
xmlindex_sel_idx_tbl opt_param('optimizer_inmemory_aware' 'false')
no_substrb_pad  */
      TO_CHAR(COUNT("PROD_ID")),
         SUBSTRB (DUMP (MIN ("PROD_ID"),
               16,
               0,
               64), 1, 240),
SUBSTRB (DUMP (MAX ("PROD_ID"),
               16,
               0,
               64), 1, 240),
TO_CHAR (COUNT ("CUST_ID")),
SUBSTRB (DUMP (MIN ("CUST_ID"),
               16,
               0,
               64), 1, 240),
SUBSTRB (DUMP (MAX ("CUST_ID"),
               16,
               0,
               64), 1, 240),
TO_CHAR (COUNT ("TIME_ID")),
SUBSTRB (DUMP (MIN ("TIME_ID"),
               16,
               0,
               64), 1, 240),
SUBSTRB (DUMP (MAX ("TIME_ID"),
               16,
               0,
               64), 1, 240),
TO_CHAR (COUNT ("CHANNEL_ID")),
SUBSTRB (DUMP (MIN ("CHANNEL_ID"),
               16,
               0,
               64), 1, 240),
SUBSTRB (DUMP (MAX ("CHANNEL_ID"),
               16,
               0,
               64), 1, 240),
TO_CHAR (COUNT ("PROMO_ID")),
SUBSTRB (DUMP (MIN ("PROMO_ID"),
               16,
               0,
               64), 1, 240),
SUBSTRB (DUMP (MAX ("PROMO_ID"),
               16,
               0,
               64), 1, 240),
TO_CHAR (COUNT ("QUANTITY_SOLD")),
SUBSTRB (DUMP (MIN ("QUANTITY_SOLD"),
               16,
               0,
               64), 1, 240),
SUBSTRB (DUMP (MAX ("QUANTITY_SOLD"),
               16,
               0,
               64), 1, 240),
TO_CHAR (COUNT ("AMOUNT_SOLD")),
SUBSTRB (DUMP (MIN ("AMOUNT_SOLD"),
               16,
               0,
               64), 1, 240),
         SUBSTRB (DUMP (MAX ("AMOUNT_SOLD"),
                        16,
                        0,
                        64), 1, 240)
  FROM   "SH"."SALES1" t
 WHERE   tbl$or$idx$part$num ("SH"."SALES1",
                              0,
                              4,
                              0,
                              "ROWID") = :objn /*
SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,NIL,NIL,SYN,N
IL,NIL, B76753*/
```

Then the database gathered the statistics for the SALES_Q1_1998 partition.

```
DBMS_STATS: gather stats on partition SALES_Q1_1998: synopsis not gathered yet; not
analyzed yet;
DBMS_STATS: Start gather_stats.. pfix:  ownname: SH tabname: SALES1 pname:
SALES_Q1_1998 spname:  execution phase: 1
DBMS_STATS: APPROX_NDV_ALGORITHM chosen: AS
DBMS_STATS: reporting_man_log_task: target: SH.SALES1.SALES_Q1_1998 objn: 76754
auto_stats: FALSE status: IN PROGRESS ctx.batching_coeff: 0
DBMS_STATS: delete synopses of a single partition
```

Finally, Oracle computes global level statistics for the SALES1 table based on the information provided by the partitions synopsis.

```
DBMS_STATS: Start gather_stats.. pfix:  ownname: SH tabname: SALES1 pname:
spname:  execution phase: 1
DBMS_STATS: APPROX_NDV_ALGORITHM chosen: AS
DBMS_STATS: APPROX_NDV_ALGORITHM chosen: AS
DBMS_STATS: Synopsis Aggregation Degree: 1
DBMS_STATS: APPROX_NDV_ALGORITHM chosen: AS
DBMS_STATS: get_agg_colstats: AS only
DBMS_STATS: Derive global stats from partition synopses/stats for table
SALES1
```

How does Oracle use synopsis data to compute statistics? First of all, when gathering statistics with the AS algorithm the synopsis data will be stored in both the **wri$_optstat_synopsis$** and **wri$_optstat_synopsis_head$** tables. Oracle inserts some information in the **wri$_optstat_synopsis_head$** table; specifically, the split column is very important. But the spare1 and spare2 columns are null. Also, the synopsis values (hash values) are stored in the **wri$_optstat_synopsis$-hashvalue** column. So, according to formula 1, to compute the NDV we need to know the number of distinct values of the corresponding column's hash values (synopsis values) and the number of splits (which have been performed for the column).

The number of splits for the column will be

```
  SELECT   (SELECT   name
              FROM   sys.col$
             WHERE   obj# = 76753 AND col# = t.intcol#)
               column_name,  MAX (split) maxsplit
    FROM   sys.wri$_optstat_synopsis_head$ t
   WHERE   t.bo# = 76753 --AND group# = 153510
  GROUP BY   t.intcol#

COLUMN_NAME       MAXSPLIT
----------------  ----------
PROD_ID           0
QUANTITY_SOLD     0
CUST_ID           0
CHANNEL_ID        0
PROMO_ID          0
TIME_ID           0
AMOUNT_SOLD       0
```

We can find the number of splits of columns for each partition to estimate the NDV of the column within the partition of the table (by adding the predicate **group#=<value>**). Also we can find the distinct number of hash values of the column as

```
    SELECT   (SELECT   name
              FROM   sys.col$
             WHERE   obj# = 76753 AND col# = t.intcol#)
               column_name, COUNT (DISTINCT (hashvalue)) dhv
    FROM   sys.wri$_optstat_synopsis$ t
   WHERE   bo# = 76753    --and group#=153510
   GROUP BY   intcol#;

COLUMN_NAME       DHV
----------------- ---------
PROD_ID           60
CUST_ID           4305
TIME_ID           181
CHANNEL_ID        4
PROMO_ID          2
QUANTITY_SOLD     1
AMOUNT_SOLD       425
```

So, we have “N” and “i” for formula 1. For instance, if we take the column `AMOUNT_SOLD`, then its NDV will be equal **to NDV=N*power(2,i) = 425*power(2,0)=425**. Now we can check the table statistics in the dictionary.

```
SELECT   column_name, num_distinct
  FROM   dba_tab_col_statistics
 WHERE   table_name = 'SALES1'

COLUMN_NAME       NUM_DISTINCT
----------------- ---------
PROD_ID           60
CUST_ID           4305
TIME_ID           181
CHANNEL_ID        4
PROMO_ID          2
QUANTITY_SOLD     1
AMOUNT_SOLD       425
```

Actually, we do not have any splits for the columns and therefore the number of distinct hash values of the columns is equal to their NDV.

In addition, if we have a partitioned table and statistics have been gathered with “ADAPTIVE SAMPLING” and we want to gather statistics for newly added partitions with the “HYPERLOGLOG” option, then Oracle deletes all previous synopses and regathers statistics for them with the “HYPERLOGLOG” algorithm. This occurs even if we want to gather statistics for a single partition, as below:

```
/* SALES1 tables has two partitions SALES_Q1_1998
SALES_Q2_1998  */

begin
    DBMS_STATS.set_table_prefs (ownname   => 'sh',
                                tabname   => 'sales1',
                                pname     => 'approximate_ndv_algorithm',
                                pvalue    => 'ADAPTIVE SAMPLING');
end;

execute DBMS_STATS.gather_table_stats('sh','SALES1');
```

So, in this case we will have only an old-style synopsis. But, now we try to add a new partition and gather statistics for it.

```
ALTER TABLE sh.sales1 ADD PARTITION
 sales_q3_1998 VALUES LESS THAN (TO_DATE(' 1998-10-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
INSERT INTO sh.sales1
    SELECT   *
      FROM   sh.sales PARTITION (sales_q3_1998);

begin
    DBMS_STATS.set_table_prefs (ownname   => 'sh',
                                tabname   => 'sales1',
                                pname     => 'approximate_ndv_algorithm',
                                pvalue    => 'HYPERLOGLOG');
   dbms_stats.gather_table_stats
    (ownname=>'sh',
     tabname=>'SALES1',
     partname=>'sales_q3_1998');
end;
```

As a result Oracle will delete all old-style synopses and will create a new synopsis with the “HYPERLOGLOG” option.

What can we say about the performance efficiency of these two algorithms? I have tested both algorithms for a table with a size of125GB and 56 partitions. First, I gathered statistics with “HYPERLOGLOG” and then deleted the statistics and regathered them with “ADAPTIVE SAMPLING”. It seems there is no dramatic (big) difference between these two statistics-gathering process. And the accuracy of the both algorithms is ≈2%. The main difference is the required memory. HLL allows us to compute NDV with minimal memory (and disk storage) with high accuracy. Execution statistics can be seen from the lines below:

Execution statistics for HLL:

```
OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        1      0.03       0.06          6         24          0           0
Execute      1      0.56       1.08         55       1416        125           1
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        2      0.59       1.14         61       1440        125           1

Misses in library cache during parse: 1


OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse      796      0.21       0.21          0         94          0           0
Execute   3351      0.84       1.27         82       2459       2606         554
Fetch     4678   3566.07    3908.89   32675780   32689035        233        9174
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     8825   3567.13    3910.38   32675862   32691588       2839        9728
```

Execution statistics for AS:

```
OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS
call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse        2      0.03       0.09          6         24          0           0
Execute      2      0.57       0.97         55       1416        142           1
Fetch        0      0.00       0.00          0          0          0           0
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total        4      0.61       1.07         61       1440        142           1

Misses in library cache during parse: 1


OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS

call     count       cpu    elapsed       disk      query    current        rows
------- ------  -------- ---------- ---------- ---------- ----------  ----------
Parse      772      0.17       0.21          0         90          0           0
Execute   3090      0.82       1.72         78       1538       3568         588
Fetch     4233   3636.26    4011.21   32675764   32687874        249        8275
------- ------  -------- ---------- ---------- ---------- ----------  ----------
total     8095   3637.26    4013.15   32675842   32689502       3817        8863
```

In summary, we see new statistics gathering mechanism(s) in latest version of Oracle database. In Oracle 11g, one-pass distinct sampling was implemented, which gave us the ability to compute statistics very accurately and efficiently. In Oracle 12c R1, to approximate NDV a new function was introduced – `APPROX_COUNT_DISTINCT`, which uses the HLL algorithm. But, in Oracle 12c R2 the HLL algorithm has been implemented to approximate database statistics (NDV) with the DBMS_STATS package. Oracle database still can use both algorithms (AS/HLL).

For AS, synopsis data is stored in both (**wri$_optstat_synopsis_head$, wri$_optstat_synopsis$**) tables but for HLL, synopsis data is stored only in the **wri$_optstat_synopsis$** table. If table (partitioned) statistics were gathered with AS then the option approximate_ndv_algorithm="REPEAT OR HYPERLOGLOG" permits us to continue creating synopses via the ADAPTIVE SAMPLING method; this is the default setting. If we change the approximate_ndv_algorithm algorithm to "HYPERLOGLOG" and INCREMENTAL_STALENESS is NULL then all old-style synopses will be deleted and new-style synopses will be created for previous and newly added table partitions. If approximate_ndv_algorithm="HYPERLOGLOG" and INCREMENTAL_STALENESS= ALLOW_MIXED_FORMAT then the database does not delete the old-style synopses immediately, but does so gradually.