

Contents

- [Objectives](#)
 - [Background Information](#)
 - [When does Oracle's CBO consider Partition-level statistics](#)
 - [Oracle bugs that can impact the statistics collection](#)
 - [Statistic collection tests on small to medium sized data volumes](#)
 - [General Guidelines](#)
 - [CBO statistic collection guidelines for small to medium sized data volumes](#)
 - [Statistic collection tests for large data volumes](#)
 - [CBO statistic collection guidelines for large data volumes](#)
 - [Plan stability](#)
 - [DBMS_STATS tests for Oracle Release 9.2.0.5 and 10.1.0.2](#)
 - [Automatic Workload Repository and CBO statistic collection in 10g](#)
 - [A writers naive conclusions](#)
-

Objectives

This article looks at some of the questions and challenges you need to consider when you are planning statistics collection for Oracle's Cost Based Optimizer (CBO). Many businesses require 24 by 7 operations which often put constraints on the available timeframe and hardware resources for collecting CBO statistics.

The purpose of this investigation is to

- Highlight how Oracle's statistics collection methods `dbms_stats` and `analyze` work.
- Test and analyse ways to produce quality statistics, within a reasonable time-frame.

Notes & caveats

This investigation is based on Oracle RDBMS release 9.2.0.3 and 9.2.0.5 running on HP-UX and RDBMS 10.0.1.2 running on XP professional. All recommendations given in this document are guidelines only. Each implementation will be different from site to site and the best approach will be dependent on the database size, available resources and operational window.

Background Information

Collection of Oracle CBO statistics can be done with `analyze` or Oracle's supplied `dbms_stats` package which has been available since version 8.1.5. In order to fast-track your understanding of the basics of statistics collection, you should read the following notes from <http://www.metalink.oracle.com> which will provide you with a solid foundation:

- 236935.1 - DBMS_STATS vs. analyze
- 236935.1 - Global statistics
- 114671.1 - Gathering Statistics for the Cost Based Optimizer

The information in the `Global Statistics`, `How to gather Global Statistics`, `How to check if Global statistics are gathered` and `DBMS_STATS versus analyze` sections are a bit of a repeat of what you can find in the above mentioned metalink documents, but is probably worth a quick review here anyway.

Global Statistics

Global statistics are gathered statistics that provide information regarding an object as a whole. For a partitioned table, the statistics at the table level, can be gathered directly (and are called Global Statistics) or can be derived from the statistics on the underlying partitions (called Aggregate or Derived Statistics). It is important to understand that Global statistics are **NOT** derived from an aggregation of the statistics collected against underlying objects. Even though a SubPartition has no underlying objects, the statistics gathered at SubPartition level are called Global Statistics as they are global for that level of the object.

How to gather Global Statistics

Please note that global statistics is specific to the DBMS_STATS package. The alternative statistic collection method ANALYZE, only collect statistic at the lowest level and derives higher level statistics by aggregation. Please refer to the DBMS_STATS versus ANALYZE section.

The Granularity parameter of DBMS_STATS package specifies the level to which statistics should be gathered

DBMS_STATS Granularity	Global Statistics Level
Global	Table
Partition	Partition
SubPartition	SubPartition
Default	Table + Partition
All	Table + Partition + Subpartition

As shown above a granularity of 'DEFAULT', or 'ALL' gathers statistics at more than 1 level. If the objective is to collect global statistics from all levels of a SubPartitioned table, set the Granularity parameter to 'ALL' and DBMS_STATS will generate 3 sets of SQL to gather the statistics for the table, partition and sub-partition.

How to check if Global statistics are gathered

Verify the GLOBAL_STATS column value(either Yes or No) for the below mentioned Data Dictionary views:

- Table level:
 - [DBA USER ALL]_TABLES
 - [DBA USER ALL]_TAB_COL_STATISTICS
- Partition level:
 - [DBA USER ALL]_TAB_PARTITIONS
 - [DBA USER ALL]_PART_COL_STATISTICS
- Sub-Partition level:
 - [DBA USER ALL]_TAB_SUBPARTITIONS
 - [DBA USER ALL]_SUBPART_COL_STATISTICS
- Index level:
 - [DBA USER ALL]_INDEXES
 - [DBA USER ALL]_IND_PARTITIONS

DBMS_STATS versus ANALYZE

Oracle recommends collecting statistics with the supplied DBMS_STATS package, which can gather global statistics at multiple levels as specified by the granularity parameter. Compare this to the ANALYZE command which collects statistics only at the lowest level and derives higher level statistics by aggregation.

The key point for Oracle's recommendation are; Unless the query predicate narrows the query to a single partition, the optimizer uses the global statistics. Because most queries are not likely to be this restrictive, it is most important to have accurate global statistics. Intuitively, it can seem that generating global statistics from partition-level statistics is straightforward; however, this is true only for some of the statistics. For example, it is very difficult to figure out the number of distinct values for a column from the number of distinct values found in each partition, because of the possible overlap in values. Therefore, actually gathering global statistics with the DBMS_STATS package is highly recommended, rather than calculating them with the ANALYZE statement.

For the full detail on this please refer to the above mentioned metalink documents.

When does Oracle's CBO consider Partition-level statistics?

Do we actually need the partition level stats and under what circumstances does the CBO consider these partition level statistics?

Well, let's try to create a partitioned table with a few columns and rows. We'll then create an index on the partition key and investigate when Oracle would consider using the partition level statistics.

```
SQL> create table pba (col_num number, col_num2 number, col_text varchar2(30))
      partition by range (col_num)
      (
        partition p1 values less than (10),
        partition p2 values less than (20)
      )
```

Table created.

```
SQL> begin
  2   insert into pba values (1,100,'TEXT');
  3
  4   for i IN 0..9999 LOOP
  5     INSERT INTO pba VALUES (11,100,'NEW TEXT');
  6   end loop;
  7 end;
  8 /
```

PL/SQL procedure successfully completed.

```
SQL> commit;
```

Commit complete.

```
SQL> select count(*) from pba;
      COUNT(*)
-----
      10001
```

```
SQL> create index i_pba on pba(col_num) local;
```

Index created.

```
SQL> analyze table pba compute statistics; or could have been done with dbms_stat
```

Table analyzed.

```
SQL> set autot trace exp
```

```
SQL> select sum(col_num2) from pba where col_num = 1;
```

Execution Plan

```
-----
 0          SELECT STATEMENT Optimizer=CHOOSE (Cost=2 Card=1 Bytes=3)
 1      0      SORT (AGGREGATE)
 2      1      TABLE ACCESS (BY LOCAL INDEX ROWID) OF 'PBA' (Cost=2 Card=1 Bytes=3)
 3      2      INDEX (RANGE SCAN) OF 'I_PBA' (NON-UNIQUE) (Cost=1 Card=1)
```

```
SQL> select sum(col_num2) from pba where col_num = 11;
```

Execution Plan

```
-----
 0          SELECT STATEMENT Optimizer=CHOOSE (Cost=8 Card=1 Bytes=4)
 1      0      SORT (AGGREGATE)
 2      1      TABLE ACCESS (FULL) OF 'PBA' (Cost=8 Card=10000 Bytes=40000)
```

So if the statement contain a constant value for the partition key, Oracle are able to choose an appropriate execution plan based on the partition statistics. As shown in bold text above, when the value of the partition key column were equal to 1, the CBO can work out the the cheapest cost to access the requested data is through and index range scan of I_PBA. Also when the value of the partition key were equal to 11, the CBO knows that the selectivity of the requested value is low and we would be better of accessing the requested data through a full tale scan of our table.

Another important point to clarify, is that Oracle 9i/10g is able to use peeking of bind variables at parse time - so let's run the same test case with bind variables instead.

```
SQL> set autot off
```

```
SQL> alter session set events '10046 trace name context forever, level 12';
```

Session altered.

```
SQL> alter session set tracefile_identifier = 'bind';
```

Session altered.

```
SQL> var v1 number
```

```
SQL> execute :v1 := 1;
```

PL/SQL procedure successfully completed.

```
SQL> select /* Please parse me */
```

```
 2 sum(col_num2) from pba where col_num = :v1;
```

```
  SUM(COL_NUM2)
```

```
-----
      100
```

```
SQL> execute :v1 := 11;
```

```
SQL> select /* Parse me again */ sum(col_num2) from pba where col_num = :v1;
More ...
      SUM(COL_NUM2)
-----
      1000000
```

```
SQL> alter session set events '10046 trace name context off';
```

```
Session altered.
```

```
SQL> spool off
```

An extract of the output from the trace file is presented below (the full trace file has been omitted for reasons of clarity):

```
PARSING IN CURSOR #1 len=60 dep=0 uid=26 oct=3 lid=26 tim=5456488191271 hv=169490
select /* Please parse me */
sum(col_num2) from pba where col_num = :v1
END OF STMT
PARSE #1:c=0,e=1080,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=0,tim=5456488191251
BINDS #1:
  bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=03 oacfl2=8000000100000000
    bfp=80000001001da8c8 bln=22 avl=01 flg=05
    value=1
EXEC, FETCH and WAIT are removed for clarity.....
STAT #1 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT AGGREGATE (cr=2 r=0 w=0 time=159 us
STAT #1 id=2 cnt=1 pid=1 pos=1 obj=0 op='PARTITION RANGE SINGLE PARTITION: KEY KE
STAT #1 id=3 cnt=1 pid=2 pos=1 obj=10484 op='TABLE ACCESS BY LOCAL INDEX ROWID PB
=125 us)'
STAT #1 id=4 cnt=1 pid=3 pos=1 obj=10487 op='INDEX RANGE SCAN I_PBA PARTITION: KE

PARSING IN CURSOR #1 len=59 dep=0 uid=26 oct=3 lid=26 tim=5456837174221 hv=293931
select /* Parse me again */ sum(col_num2) from pba where c1 = :v1
END OF STMT
PARSE #1:c=0,e=1080,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=0,tim=5456837174202
BINDS #1:
  bind 0: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=03 oacfl2=8000000100000000
    bfp=80000001001da938 bln=22 avl=02 flg=05
    value=11
EXEC, FETCH and WAIT are removed for clarity.....
STAT #1 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT AGGREGATE (cr=31 r=0 w=0 time=22118
STAT #1 id=2 cnt=10000 pid=1 pos=1 obj=0 op='PARTITION RANGE SINGLE PARTITION: KE
STAT #1 id=3 cnt=10000 pid=2 pos=1 obj=10484 op='TABLE ACCESS FULL PBA PARTITION:
```

As can be seen from the trace file output, Oracle, looks at the bind variable values, shown in bold above, to help determine the optimal access path.

So from the trace files we can see that the partition-level statistics are useful as;

- Oracle uses partition statistics to generate execution plans.
- Oracle uses bind variable peeking when generating execution plans for partitioned tables.

At this point, a big thank you to Julian Dyke (www.juliandyke.com), who was the first one to tell me about this behaviour.

Known Oracle bugs that can impact the statistic collection

- Bug no 2917053 - DBMS_STATS.GATHER_TABLE_STATS RESULTS ARE INCORRECT.
http://www.metalink.oracle.com/metalink/plsql/ml2_documents.showFrameDocument?p_database_id=BUG&p_id=2917053
- Bug no 3105084 - SAMPLE SIZE OF INDEXES IS DIFFERENT THAN SPECIFIED WITH GATHER_SCHEMA_STATS
http://www.metalink.oracle.com/metalink/plsql/ml2_documents.showFrameDocument?p_database_id=BUG&p_id=3105084
- Bug no 2919423 - DBMS_STATS TAKES AT LEAST TWICE AS LONG IN 9.2.0.3.0 AS IN 8.1.7.4.0 http://metalink.oracle.com/metalink/plsql/ml2_documents.showFrameDocument?p_database_id=BUG&p_id=2919423

Not sure why 2919423, is listed as a bug, as most of the statistics collection, particular for partition is done with a different approach. However, please note, when using the dbms_stats package with the cascade option set to true, the major issue that i found in 9.2.0.3 is that the index statistics sometimes get gathered twice, first with the original given sample size (example 1%) and then with a much large sample size (example 90%). This simply just hurt too much, in large data volume environment, with for example 200 plus partitions. The below tests showed, the impact was approximately an extra minute per index per partition.

In Metalink Bug no/Doc id 2919423 it is mentioned, that statistics in 9.2.0.3 are considered to be unreliable, when the sample size is less than 919 blocks and Oracle chooses to do a second iteration. This might be a good idea but the chosen percentage for the second iteration seems to be way too big, at least at the time of writing. Patch 2919423 (Backport patch for 9.2.0.3, and should be part of patch set 9.2.0.5), does address the above issue. However, it looks like to be a trade-off at the moment, as after implementing the back-port patch, 2919423 on 9.2.0.3, isolated tests showed that the statistics gathering was processed faster, however the index statistics produced were less accurate. You can see this from the below statistics output and the number of distinct key figures highlighted in bold text. The statistics are from a test case with 3 tables, each with 3 partitions and each partition with one million rows. The tables each have an index on a number(10) column (fkey_id) and overall there are thirty thousand distinct keys per table.

STATS table is analysed with the old approach of analyze table.
STATS2 table is analysed with dbms_stats.gather_table_stats with estimate on 5 pe
STATS3 table is analysed with dbms_stats.gather_table_stats with auto sampling

```
sql>@show_tab_col_stats
TABLENAME COLUMNNAME      DISTINCT DENSITY      NULLS BUCKETS ANALYZED SAMPLE_SIZE GL
-----
STATS     STATS_ID          3004260 3.3286E-07      0      1 27/APR/04
STATS     LAST_MODIFIED    230 .004347826      0      1 27/APR/04
STATS     FKEY_ID          30144 .000033174      0      1 27/APR/04
STATS     TEXT             1      1                0      1 27/APR/04
STATS2    STATS_ID          2992720 3.3414E-07      0      1 27/APR/04 149636 Y
STATS2    LAST_MODIFIED    642 .001557632      0      1 27/APR/04 149636 Y
STATS2    FKEY_ID          30018 .000033313      0      1 27/APR/04 149636 Y
STATS2    TEXT             1      1                0      1 27/APR/04 149636 Y
STATS3    STATS_ID          3000000 3.3333E-07      0      1 27/APR/04 3000000 Y
STATS3    LAST_MODIFIED    526 .001901141      0      1 27/APR/04 3000000 Y
STATS3    FKEY_ID          30001 .000033332      0      1 27/APR/04 3000000 Y
STATS3    TEXT             1      1                0      1 27/APR/04 3000000 Y
```

```
sql>@show_ind_col_stats
TNAME  IDX_NAME  LB DISTINCT  NUM_ROWS PAR SAMPLE_SIZE ANALYZED GLO
-----
STATS  I_FKEY   7594  37687 3016532.77 NO      160082 27/APR/04 NO
STATS  I_STATS  7854  3028324 3028324.86 NO      156930 27/APR/04 NO
STATS2 I_FKEY2  7728  5641  3092825 NO      466625 27/APR/04 YES
```

```

STATS2 I_STATS2 7697 2976357 2976357 NO 434667 27/APR/04 YES
STATS3 I_FKEY3 7457 5470 2995061 NO 451875 27/APR/04 YES
STATS3 I_STATS3 7840 3023467 3023468 NO 441547 27/APR/04 YES

```

As you can see from the figures in bold the number of distinct keys are only a 1/6 of the actual distinct count. At the time of testing, this issue was not fixed in 9.2.0.5 or in 10.1.0.2. Please see [DBMS_STATS tests for Oracle Release 9.2.0.5 and 10.1.0.2](#) for more details.

An option to fix this would be to gather your index statistics separately with `dbms_stats.gather_index_stat` compute statistics, but defeat the purpose of what we originally was trying to achieve. Alternatively, if you ever find any justification for rebuilding your indexes, you could use the 'compute statistics' clause when rebuilding the indexes to collect the index stats at minimal additional cost.

Statistic collection tests on small to medium sized data implementations

The scripts and results below are from isolated tests of the three tables (STATS, STATS2 and STATS3) each table has three partitions and each partition has one million rows. Each table has two indexes, one primary key and one foreign key index, with a total of 30,000 distinct key values. On each of the tables I have used a different analyze method to confirm the background information above.

Please note that all of the tests were performed in a single stream/process on Oracle release 9.2.0.3

Test 1 - On the STATS table i used;

```
analyze table USER.STATS estimate statistics sample 5 PERCENT;
```

The analyze command was traced and executed multiple times. The analyze command executed in between 6 and 8 seconds. Not surprisingly, the statistics were similar to the statistics output produced above from the table named STATS, apart from the sample size.

Test 2 - On the STATS2 table i used;

```
dbms_stats.gather_table_stats(user, 'STATS2', granularity => 'DEFAULT', estimate_percent => 5,
cascade => TRUE, method_opt => 'FOR ALL COLUMNS SIZE 1');
```

The `dbms_stats` command was traced and executed multiple times. The `dbms_stats` command executed in between 38 and 43 seconds. As discussed earlier, everything looks fine, apart from number of distinct keys on the non-unique index.

```

SQL> @show_tab_stats
TABLENAME      ANALYZED      SAMPLE_SIZE  AVG_ROW_LEN  AVG_SPACE  NUM_ROWS  BLOCKS  G
-----
STATS2         27/APR/04    149636      23           0          2992720   11820   Y

```

```

SQL> @show_tab_col_stats
More ...
TABLENAME      COLUMNNAME      NUM_DISTINCT  DENSITY  NUM_NULLS  NUM_BUCKETS  ANALY

```

```

-----
STATS2      STATS_ID          2992720 3.3414E-07      0      1 27/APR/04
STATS2      LAST_MODIFIED    642 .001557632    0      1 27/APR/04
STATS2      FKEY_ID          30018 .000033313    0      1 27/APR/04
STATS2      TEXT             1      1      0      1 27/APR/04

```

```
SQL> @show_part_stats
```

```

TABLENAME      PARTITIONNAME      ANALYZED SAMPLE_SIZE AVG_ROW_LEN  AVG_SPACE  NUM_ROWS
-----
STATS2         P04_S1000000      27/APR/04      50004      21          0      1000
STATS2         P05_S2000000      27/APR/04      50185      23          0      1003
STATS2         P06_S3000000      27/APR/04      49982      23          0      999

```

```
SQL> @show_part_col_stats
```

```

TNAME      PARTITIONNAME      COLUMNNAME      NUM_DISTINCT      DENSITY      NUM_NULLS      NUM_BUCKETS      SAMPLE_SIZE
-----
STATS2     P04_S1000000      STATS_ID          1000080 9.9992E-07      0      1      1000
STATS2     P04_S1000000      LAST_MODIFIED    214 .004672897    0      1      1000
STATS2     P04_S1000000      FKEY_ID          10015 .00009985     0      1      1000
STATS2     P04_S1000000      TEXT             1      1      0      1      1000
STATS2     P05_S2000000      STATS_ID          1003700 9.9631E-07      0      1      1003
STATS2     P05_S2000000      LAST_MODIFIED    229 .004366812    0      1      1003
STATS2     P05_S2000000      FKEY_ID          9996 .00010004     0      1      1003
STATS2     P05_S2000000      TEXT             1      1      0      1      1003
STATS2     P06_S3000000      STATS_ID          999640 1.0004E-06     0      1      999
STATS2     P06_S3000000      LAST_MODIFIED    201 .004975124    0      1      999
STATS2     P06_S3000000      FKEY_ID          10002 .00009998     0      1      999
STATS2     P06_S3000000      TEXT             1      1      0      1      999

```

```
SQL> @show_ind_col_stats
```

```

IDX_NAME      LEAF_BLOCKS      DISTINCT_KEYS      NUM_ROWS      PAR      SAMPLE_SIZE      AN
-----
I_FKEY2       7728      5641      3092825      NO      466625      27
I_STATS2      7697      2976357      2976357      NO      434667      27

```

Interestingly enough, when we take a look at a 10046 trace that I had enabled during the statistics collection, you will see that the index statistic collection was done with approximately 15% sample size and done with block sampling. It is a known issue that index statistics collection, is gathered with a different sample (bug 3105084), but it is not mentioned that it is done with block sampling.

```

=====
PARSING IN CURSOR #25 len=370 dep=1 uid=26 oct=3 lid=26 tim=2209264218617 hv=3832
select /*+ cursor_sharing_exact dynamic_sampling(0) no_monitoring no_expand index
count(*) as nrw,count(distinct sys_op_lbid(10678,'L',t.rowid)) as nlb,coun
as ndk,sys_op_countchg(substrb(t.rowid,1,15),1) as clf
from "STATS2" sample block (15.08733911216180719726819017599159443131) t where
END OF STMT
PARSE #25:c=10000,e=2087,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=4,tim=2209264218594
EXEC #25:c=0,e=85,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=2209264218867
FETCH #25:c=3750000,e=10221942,p=3337,cr=1596,cu=10,mis=0,r=1,dep=1,og=4,tim=2209
STAT #25 id=1 cnt=1 pid=0 pos=1 obj=0 op='SORT GROUP BY (cr=1596 r=3337 w=17 time
STAT #25 id=2 cnt=466625 pid=1 pos=1 obj=10678 op='INDEX SAMPLE FAST FULL SCAN I
=====

```

Oracle need to use the block sampling to collect the index statistics as Oracle have to read (sample) the index leaf blocks sequentially to generate the clustering factor by counting the number of jumps to 'a different' table block. As mentioned earlier block sampling do save I/O operation, but i guess block sampling could also be the reason for the inaccuracy for non-unique indexes, as DBMS_STATS, to my knowledge, expects truly random samples.

Test 3 - On the STATS2 table i used;

dbms_stats.gather_table_stats(user, 'STATS2', granularity => 'DEFAULT', estimate_percent => 5, block_sample => TRUE, cascade => TRUE, method_opt => 'FOR ALL COLUMNS SIZE 1');

The dbms_stats command, was traced and executed multiple times. As highlighted in bold text below, the number of distinct keys for column fkey_id, was way too low for the table, partition and index statistics. As with test 2, index statistics were collected with a 15% block sampling size.

```
SQL> @show_tab_stats
TABLENAME      ANALYZED SAMPLE_SIZE AVG_ROW_LEN  AVG_SPACE  NUM_ROWS  BLOCKS GL
-----
STATS2         29/APR/04   157435      23          0    3148700  11820 Y
```

```
SQL> @show_tab_col_stats
TABLENAME      COLUMNNAME      NUM_DISTINCT  DENSITY  NUM_NULLS  NUM_BUCKETS  ANALY
-----
STATS2         STATS_ID        3148700  3.1759E-07  0          1  29/AP
STATS2         LAST_MODIFIED   405    .002469136  0          1  29/AP
STATS2         FKEY_ID         2162 .000462535  0          1  29/AP
STATS2         TEXT            1      1          0          1  29/AP
```

```
SQL> @show_part_stats
TABLENAME      PARTITIONNAME  ANALYZED SAMPLE_SIZE AVG_ROW_LEN  AVG_SPACE  NUM_RO
-----
STATS2         P04_S1000000  29/APR/04   48724      21          0    974
STATS2         P05_S2000000  29/APR/04   53268      23          0   1065
STATS2         P06_S3000000  29/APR/04   48516      23          0    970
```

```
SQL> @show_part_col_stats
TNAME  PARTITIONNA  COLUMNNAME  NUM_DISTINCT  DENSITY  NUM_NULLS  NUM_BUCKETS  SA
-----
STATS2 P04_S1000000  STATS_ID    974480  1.0262E-06  0          1
STATS2 P04_S1000000  LAST_MODIFI 136    .007352941  0          1
STATS2 P04_S1000000  FKEY_ID     666 .001501502  0          1
STATS2 P04_S1000000  TEXT        1      1          0          1
STATS2 P05_S2000000  STATS_ID    1065360  9.3865E-07  0          1
STATS2 P05_S2000000  LAST_MODIFI 150    .006666667  0          1
STATS2 P05_S2000000  FKEY_ID     742 .001347709  0          1
STATS2 P05_S2000000  TEXT        1      1          0          1
STATS2 P06_S3000000  STATS_ID    970320  1.0306E-06  0          1
STATS2 P06_S3000000  LAST_MODIFI 125    .008        0          1
STATS2 P06_S3000000  FKEY_ID     680 .001470588  0          1
STATS2 P06_S3000000  TEXT        1      1          0          1
```

```
SQL> @show_ind_col_stats
IDX_NAME      LEAF_BLOCKS  DISTINCT_KEYS  NUM_ROWS  PAR  SAMPLE_SIZE  AN
-----
I_FKEY2       7960        5789         3180965  NO   479923  29
I_STATS2      7649        2955945       2955945  NO   431686  29
```

Test 4 - On the STATS3 table i used

dbms_stats.gather_table_stats(user, 'STATS3', granularity => 'DEFAULT', cascade

The dbms_stats command was traced and executed multiple times. The command executed in between 155 and 160 seconds. We ended up with a sampling size of 100%, which was also reflected in the timing of the statistics collection. But yet again the index statistics collection was performed as in test 2 and 3. From what i have able to workout from the trace files with auto sampling, the

sampling size gets increased with a factor of 100 for every iteration, until Oracle is happy with the statistics gathered.

```
SQL> @show_ind_col_stats
IDX_NAME          LEAF_BLOCKS DISTINCT_KEYS  NUM_ROWS PAR SAMPLE_SIZE
ANALYZED GLO
-----
STATS3 I_FKEY3          7457      5470  2995061 NO    451875 27/APR/04
YES
STATS3 I_STATS3         7840     3023467  3023468 NO    441547 27/APR/04
YES
```

General Guidelines

The hardest type of column for statistics collection is one with very high cardinality, but where the values are not unique - for example an `account_id` column in an order table. The reason is that you have to see a lot of data before you can have a good guess at the cardinality.

The auto sample size feature was meant to address this problem. It will use small samples for columns where statistics collection is easy, and larger samples for the more difficult cases. It may be that the larger samples for the more difficult columns exceed our available time window and we therefore end up with a sample size of 100 percent - at least that is what all the tests for small, medium and large data volumes have indicated so far. In general, collecting statistics will always involve reading blocks from the database and performance will therefore be I/O dependent.

When using `dbms_stats` to estimate statistics you can specify the sampling percentage and whether sampling should be based on rows or blocks. Row sampling is most likely to produce the most random data for estimates, however block sampling reduces the amount of I/O activity for the given sample size. So we have to consider the trade-off between accuracy and speed.

Please note that sorted data combined with block sampling is likely to have a "negative" impact, or require a higher sampling size than row sampling on the estimation, since the `dbms_stats` package assumes that it is getting a truly random sample.

Guidelines for small and medium sized tables/implementations

For small sized data volumes, say less than 100,000 rows per day per partition and no more than 120 online partitions, the best approach tested was test number 2, with row sampling, and a sampling size of between 5 and 10 percent.

For medium sized implementations, say less than 1 million rows per day per partition for no more than on-line 120 partitions, the best approach tested was test number 2, with row sampling and a sampling size of 3 to 5 percent. Please note that in these tests there was hardly any difference between the time to collect the statistics with either row or block sampling, however, the distinct key count for the `fkey_id` is inaccurate with block sampling. This doesn't mean that you shouldn't try out block sampling, but the statistics need to be reasonably accurate - with block sampling you might need to increase the sampling size to get reasonably accurate statistics.

Each site and implementation is different. The above recommendations are only guidelines and the responsible DBA, will need to ensure that good quality statistics are available for the Cost Based Optimizer.

One important point you need to be aware of is, the number of distinct keys will be used to calculate the cost of an access path, both for tables and index columns. If you are interested in exploring, how the cost is calculated and i can recommend Wolfgang Breitling excellent papers on the CBO, which you can find on <http://www.centrexcc.com>.

Statistic collection tests for large data volumes

The results below are from isolated tests of statistics collection of a partitioned table in a Tier1 environment. For these tests the partitioned table had 32 partitions. The table had a total of 1 billion rows with a maximum of 40 million rows in each partition. We will use the I_ACCT index (acct_id column) on this table to validate the index statistics. The acct_id column, has an actual distinct count of 8.5 million rows. Please note that during these tests the data in the partitions was sorted by acct_id.

Acct test 4 - dbms_stats.gather_table_stats with row sampling of 1 percent and no index statistics gathering.

```
dbms_stats.gather_table_stats(user, 'table', granularity =>'DEFAULT', estimate_percent =>1);
```

The dbms_stats command had a total elapsed time of 1 hour 38 minutes and 24 seconds. The statistics for the table, columns and partitions were reasonably accurate.

Acct test 5 - dbms_stats.gather_table_stats with block sampling of 1 percent and no index statistics gathering.

```
dbms_stats.gather_table_stats(user, 'table', granularity =>'DEFAULT', estimate_percent =>1, block_sample => TRUE);
```

The analyze command had a total elapsed time of 1 hour 19 minutes and 58 seconds. Compared to test 4, a saving of 18 minutes was achieved here by using block sampling. The statistics were not as good as with row sampling, but still reasonably accurate.

Acct test 6 - dbms_stats.gather_table_stats with block sampling of 0.1 percent and no index statistics gathering.

```
dbms_stats.gather_table_stats(user, 'table', granularity =>'DEFAULT', estimate_percent =>0.1, block_sample => TRUE);
```

The total elapsed time was 16 minutes and 14 seconds. A good result, if analysing the whole schema was the goal. However, on both the table and the partitions, the count distinct key for the account_id column was very low. For example, the acct_id column stats showed only 463,000 distinct keys (actual value is 8.5 million). So in this case, block sampling with a sampling size of 0.1 % didn't seem to be a good combination.

Acct test 6B - dbms_stats.gather_table_stats with block sampling of 0.1 percent including index statistics gathering.

```
dbms_stats.gather_table_stats(user, 'table', granularity =>'DEFAULT', estimate_percent =>0.1, block_sample => TRUE, cascade => TRUE);
```

The total elapsed time was 4 hours, 1 minute and 14 seconds. 3 hours and 40 minutes for collecting

index statistics seems to be too long when compared to the overall elapsed time. A detailed look at the trace file revealed that we had run into an Oracle bug 2919423. The statistics for the table, partition and index on the acct_id column were very low. For example, the acct_id index column was estimated to have only 135,000 distinct keys.

The following test results are with Oracle patch 2919423 installed.

Acct test 7 - dbms_stats.gather_table_stats with block sampling of 0.1 percent including index statistics gathering. This was similar to acct test 6 to verify the impact of patch 2919423

```
dbms_stats.gather_table_stats(user, 'table', granularity => 'DEFAULT', estimate_percent => 0.1,
block_sample => TRUE, cascade => TRUE);
```

The dbms_stats command had a total elapsed time of 1 hour and 35 minutes. A much better elapsed time was achieved compared to acct test 6B. A look in the trace file showed that the index sampling size used was 2.4%. There could be a trade-off here between having the patch installed or having a “better” estimated sampling size without the patch. However, as with the table and partition columns, the estimated statistics for the indexed columns acct_id were still too low, reporting only around 133,000 distinct keys.

Acct test 8 - dbms_stats.gather_table_stats with block sampling of 1 percent including index statistics gathering. Attempt to improve column statistics.

```
dbms_stats.gather_table_stats(user, 'table', granularity => 'DEFAULT', estimate_percent => 1,
block_sample => TRUE, cascade => TRUE);
```

The total elapsed time was 2 hours and 59 minutes. Even though this test is executed in a single stream/process, when scaled up to two hundred plus partitions this would take a fair bit of resources to complete in less than 10 hours. The acct_id's column stats improved to 2.7 millions distinct keys, but the partitioned column stats for the acct_id column seemed to be too low in many instances. The acct_id column index stats only showed 125,000 distinct keys. The trace file again showed that an index sampling size of 2.4% was used. So in this test case, even a 1 percent block sampling size, on sorted data, does not seem to do a reasonable job.

CBO statistic collection guidelines for large volumes of partitioned data

A number of tests were carried out with different sample sizes and block vs. rows sampling was compared some of them documented above. Despite this, with the available platform and resources, I always run into the same inaccurate statistics vs. “time to collect” issue, even with 16 CPUs available. As expected, it is not really feasible to try to gather full statistics for all the tables, indexes and columns.

However there are other options and approaches to consider, which Mogens [Nørgaard](#) has been so kind to discuss with me.

1. Stale statistics are an option and require that table monitoring for the tables be enabled. Statistics will only be gathered when more that 10% of the table has been modified since the last statistics were gathered. This will limit the number of tables, partitions and indexes that need to be analysed. This could be a valid approach for some implementations. However, if for performance reasons you need to sort your data in a partition, you may need to copy the table statistics and disable the table

monitoring for the partitions before sorting the data in the partition. After the partitions are sorted you can enable table monitoring again.

2. Gather the full statistics and take the time and resources necessary to do that, including checking the accuracy of various percentages versus full. It's still a lot of work for the statistics gathering process to do, but then it's done, and the stats can be saved for later (ab)use. Please refer to the Oracle documentation and the DBMS_STATS package on how to copy, export and import statistics. An alternative method could be to copy statistics from partition to partition, maybe having one for workdays and one for weekends, if that makes sense in your environment.

3. Update statistics information directly, using DBMS_STATS, that's certainly faster! A variation of that would be to have a system with a copy of the database, then just copy the stats over. This alternative system need not have 42 CPU's and 42 GB of RAM - it could take a week or two to gather the statistics for that matter.

4. If a test system is available, it might be worth a try to import those statistics into the production system and see if it generate good execution plans. This approach has been used on a few very large, high-concurrency, real-world systems and performed satisfactory.

So do you really have to analyze table, daily, weekly or monthly? As always it depends; Since each implementation is different, the best approach will come down to the database size, data manipulation, available resources and the available operations window.

For instance, if your data structure will be of a similar pattern in the future you might be able to get away with only analyzing you data once. An exception is tables which constantly will change, were you might be better of using the dynamic sampling, which was introduced in 9i. With this approach you will save a lot of work for yourself and the database server and ensure prediction and stability for your applications.

The irony of it all is that even though it looks like we will be better off not constantly analyzing our data all the time - Oracle introduces new feaures that will let the database automatically collect statistics, so it will be interesting to follow which approach Oracle choose to use in the future.

When can Oracle change execution plans

The key is to have predictable performance of critical business areas.

That usually means:

1. Knowing when an important piece of SQL changes execution plan.
2. Verifying whether the new execution plan is better than the old one or wors

Three things can lead to changed execution plans:

1. Change of parameters.
2. Change of statistics.
3. Change of constants in Oracle (patch or upgrade)

It's not that often that 1 or 3 changes. But, keep in mind when performing statistics gathering, SQL

execution plans can change. So if you aren't confident with your current statistics collection method and how it is going to affect you statistics, you could potentially introduce a risk rather than an improvement.

DBMS_STATS tests for Oracle Release 9.2.0.5 and 10.1.0.2

To verify potential dbms_stats changes and behaviour in 9.2.0.5 and 10.1.0.2, I have used test cases based on five tables (STATS, STATS2, STATS3, STATS4 and STATS5) with each table having three partitions and each partition having one million rows. Each table has two indexes, one primary key and one non-unique index, with 30,000 distinct key values. In this section, I have chosen not to show the statistic and trace file output, as it is hard to fit into a readable format.

The test results for Oracle 9.2.0.5 and 10.1.0.2 were similar, but the statistics collection performed a bit better under 10.1.0.2. The main issues found were once again related to the gathering of index statistics. If the estimate percentage sample size is less than a certain number of blocks, Oracle chooses its own sample size for the indexes. Oracle now uses your sample size if the sample size is above a certain number of blocks, whereas in 9.2.0.3 with patch 2919423, Oracle seemed to ignore the percentage given.

In 10.1.0.2, DBMS_STATS's auto sampling feature works a lot better than in 9.2.0.3 and 9.2.0.5 where Oracle too often ended up with a 100 percent sample size. However, even with auto sampling, the number of distinct keys (NDK) for the non-unique index (fkey_id) was less than a third of its actual NDK.

As you have probably noticed by now, the behaviour that I noticed with dbms_stats running on release 9.2.0.3, 9.2.0.5 and 10.1.0.2, is that Oracle uses block sampling during the collection of index statistics. For all the tests performed, it didn't seem to be an issue for the unique indexes. However, it had a rather unfortunate effect on the non-unique indexes for the test cases below. It is a hard task for Oracle to help us out here, since the hardest type of column for statistics collection is one with very high cardinality, but where the values are not unique. The reason is that you need to see a lot of data before you can have a good guess as to the cardinality. Block sampling will accelerate this issue on sorted data. I have been unable to find anything in the 9i or 10g documentation about this behaviour.

Automatic Workload Repository and CBO statistic collection in 10g.

From release 10g, Oracle's recommended approach is to let Oracle be in charge of the statistic collection. By default, Oracle has enabled automatic statistics gathering and you can see the scheduled job by selecting from the new DBA_SCHEDULER_JOBS table.

```
SELECT * FROM DBA_SCHEDULER_JOBS WHERE JOB_NAME = 'GATHER_STATS_JOB';
```

The automated statistic collection will gather statistic on all objects in the database which have missing or staled statistics. You can either wait for the scheduler to start the automated CBO statistic collection, or you can start it by running;

```
execute DBMS_SCHEDULER.RUN_JOB('GATHER_STATS_JOB',TRUE); .
```

For our test case Oracle did the following; please note that below statistic and trace file output is

heavily reduced, as it is hard to fit into a readable format. Noticed from the below trace file and statistic output, that Oracle is cable of only sampling larger sample size for the columns Oracle need further sampling.

```
select /*+ cursor_sharing_exact use_weak_name_resl dynamic_sampling(0) no_monitoring */ count
(*) ,count(distinct "STATS_ID"),sum(vsize("STATS_ID")),substrb(dump(min
("STATS_ID"),16,0,32),1,120),substrb(dump(max("STATS_ID"),16,0,32),1,120),count(distinct
"LAST_MODIFIED"),substrb(dump(min("LAST_MODIFIED"),16,0,32),1,120),substrb(dump(max
("LAST_MODIFIED"),16,0,32),1,120),count(distinct "FKEY_ID"),sum(vsize("FKEY_ID")),substrb
(dump(min("FKEY_ID"),16,0,32),1,120),substrb(dump(max("FKEY_ID"),16,0,32),1,120),count
("TEXT"),count(distinct "TEXT"),sum(vsize("TEXT")),substrb(dump(min(substrb
("TEXT",1,32)),16,0,32),1,120),substrb(dump(max(substrb("TEXT",1,32)),16,0,32),1,120) from
"STATS2" sample (.1727023522) t
```

```
select /*+ cursor_sharing_exact use_weak_name_resl dynamic_sampling(0) no_monitoring */ count
(*) ,count(distinct "STATS_ID"),count(distinct "FKEY_ID") from "STATS2" sample
(1.7270235221) t
```

```
select /*+ cursor_sharing_exact use_weak_name_resl dynamic_sampling(0) no_monitoring */ count
(*) ,count(distinct "STATS_ID") from "STATS2" sample (17.2702352206) t
```

TNAME	COLUMNNAME	NUM_DISTINCT	DENSITY	NUM_NULLS	NUM_BUCKETS	LAST_A
STATS2	STATS_ID	3002252	3.3308E-07	0	1	26-NOV
STATS2	LAST_MODIFIED	332	.003012048	0	1	26-NOV
STATS2	FKEY_ID	29850	.000033501	0	1	26-NOV
STATS2	TEXT	1	1	0	1	26-NOV

And for a partition

```
select /*+ cursor_sharing_exact use_weak_name_resl dynamic_sampling(0) no_monitoring */ count
(*) ,count(distinct "STATS_ID"),sum(vsize("STATS_ID")),substrb(dump(min
("STATS_ID"),16,0,32),1,120),substrb(dump(max("STATS_ID"),16,0,32),1,120),count(distinct
"LAST_MODIFIED"),substrb(dump(min("LAST_MODIFIED"),16,0,32),1,120),substrb(dump(max
("LAST_MODIFIED"),16,0,32),1,120),count(distinct "FKEY_ID"),sum(vsize("FKEY_ID")),substrb
(dump(min("FKEY_ID"),16,0,32),1,120),substrb(dump(max("FKEY_ID"),16,0,32),1,120),count
("TEXT"),count(distinct "TEXT"),sum(vsize("TEXT")),substrb(dump(min(substrb
("TEXT",1,32)),16,0,32),1,120),substrb(dump(max(substrb("TEXT",1,32)),16,0,32),1,120) from
"SYS"."STATS2" partition ("P04_S1000000") sample (.5508096902) t
```

```
select /*+ cursor_sharing_exact use_weak_name_resl dynamic_sampling(0) no_monitoring */ count
(*) ,count(distinct "STATS_ID"),count(distinct "FKEY_ID") from "STATS2" partition
("P04_S1000000") sample (5.5080969024) t
```

PARTITIONNAM	COLUMNNAME	NUM_DISTINCT	DENSITY	NUM_NULLS	NUM_BUCKET
P04_S1000000	STATS_ID	1003940	9.9608E-07	0	
P04_S1000000	LAST_MODIFIED	108	.009259259	0	
P04_S1000000	FKEY_ID	10000	.0001	0	
P04_S1000000	TEXT	1	1	0	

And for the indexes

```
select /*+ cursor_sharing_exact use_weak_name_resl dynamic_sampling(0) no_monitoring
no_expand index_ffs(t,"I_FKEY2") */ count(*) as nrw,count(distinct sys_op_lbid
(49247,'L',t.rowid)) as nlb,count(distinct "FKEY_ID") as ndk,sys_op_countchg(substrb
```

(t.rowid,1,15),1) as clf from "STATS2" **sample block (15.0873391122,1)** t where "FKEY_ID" is not null

IDX_NAME	LEAF_BLOCKS	DISTINCT_KEYS	NUM_ROWS	PAR	SAMPLE_SIZE	LA
I_FKEY2	7397	5349	2943528	NO	444100	26
I_STATS2	7351	2841329	2841329	NO	418244	26

In general the automated optimizer statistic collection worked well for this test case, and trying to compared with the estimated auto sampling method in 9.2.0.3 and 9.2.05, 10g choose better sampling sizes, however the problem with the the number of distinct keys (NDK) for the non-unique index (fkey_id) still remain and in this test case only 1/6 of it's actual size.

It is not my intention to describe the new features in the DBMS_STATS package, but there are a few things that are worth a mention. As seen above there are cases where you need to use manual statistic collection in 10g as well. You can use the DBMS_SCHEDULER package to enable and disable the automated statistic sampling and i really like the DBMS_STATS.LOCK_TABLE_STATS('OWNER','TABLE NAME'), which will prevent new statistics being gathered on a table or schema by the DBMS_STATS_JOB. So instead of trying to do everything manually we can lock the targeted objects and address statistic collection separately.

Some of Oracle recommendations have also changed for some the DBMS_STATS procedures. For instance Oracle recommends setting the ESTIMATE_PERCENT parameter to DBMS_STATS.AUTO_SAMPLE_SIZE and mentioned that COMPUTE and ESTIMATE is only there for backward compatibility. Also Oracle recommends setting the DEGREE parameter to DBMS_STATS.AUTO_DEGREE. This setting will allow Oracle to choose a degree of parallelism based on the size of the object and the settings for the parallel-related init.ora settings. For the Column statistics and Histograms Oracle recommend setting the METHOD_OPT to FOR ALL COLUMNS SIZE AUTO.

To help the CBO estimate the I/O and CPU resources required for a query, you can collect System Statistics gather information about the systems hardware, such as I/O and CPU performance and utilization. Oracle recommend that you gather system statistics. The system statistics are not gather automatically in 10.1.0.2, like the CBO statistics, however the system statistics can be collected using the DBMS_STATS.GATHER_SYSTEM_STATS procedure.

A writer's naive conclusions

I hope this article has motivated you to test some of the statistics collection strategies you plan to use or are using, and work out how to produce quality statistics within a reasonable timeframe and with available resources

These type of challenges can be interesting to deal with, however if you don't understand how your data is structured and how the business uses the data you are not likely to provide the Cost Based Optimizer with the best data for optimizing the SQL access paths. However, DBA's often have many different databases to support and application vendors, could consider, sharing the responsibility and making the job easier. After all, it is in the software vendors interest that the application performance is reasonable and that it is well received in the market place.

It would probably not take a huge investment from the software vendor to share the statistics collection responsibility. The software vendor should be able to make the crucial data relationship assumptions for a start-up implementation. For example, they could provide a pre-load method for

tables and indexes with CBO statistics, to ensure SQL query plan stability, during a start-up period.

When rotating partitions, software vendors should consider, whether it is possible to copy partition statistics from the previous partition.

For large databases with a high data volume, it would be ideal to provide a kind of statistic collection profiler, where you can either manipulate the statistics directly or use one of the existing statistics profiles for the whole system or for an individual object.

Thank you to Mogens Nørregaard and Jonathan Lewis for providing feedback and comments.

Information sources

- Oracle 9i release 2 documentation
 - Oracle 10g documentation
 - Oracle Metalink
 - OakTable Network
-