DBA

DBMS_STATS

STATISTICS, HISTOGRAM IN ORACLE

General Terms:

Predicate: Predicate are the syntax used to specify a subset of rows to be returned. Predicate are specified in the **WHERE** clause of a SQL Statement. For example, emp_id > 10 is the predicate in

*Select * from employee where emp_id > 10;*

Selectivity: Selectivity is the fraction of rows in the table that a SQL Statement's predicate chooses.

Cardinality: The number of rows in the table or number of distinct row links in the index. The cardinality of a query is the number of rows that is expected to be returned by it.

How Statistics helps the optimizer?

The optimizer uses the selectivity of a predicate to estimate the cost of a particular access method and to determine the optimal join order and join method.

However, when there is a significant **changes** in column records specified in **WHERE** clause, then we should collect statistics to let optimizer decide the cost of a particular access method and to determine join methods. We should gather statistics periodically for objects where the statistics become stale over time because of changing data volumes or changes in column values.

For example, after loading a significant number of rows into a table, we should collect new statistics on the number of rows. After updating data in a table, we do not need to collect new statistics on the number of rows, but we might need new statistics on the average row length.

Use **DBMS_STATS** package to collect statistics. Statistics are stored in **data dictionary** and can be transferred to another database.

Statistics generated include the following:

- Table statistics
    - Number of rows
    - Number of blocks
    - Average row length
- Column statistics
    - Number of distinct values (NDV) in column
    - Number of nulls in column
    - Data distribution (histogram)
- Index statistics
    - Number of leaf blocks
    - Levels
    - Clustering factor
- System statistics
    - I/O performance and utilization
    - CPU performance and utilization

Generating Statistics:

Because the cost-based approach relies on statistics, we should generate statistics for all tables and clusters and all indexes accessed by your SQL statements before using the cost-based approach. If the size and data distribution of the tables change frequently, then regenerate these statistics regularly to ensure the statistics accurately represent the data in the tables.

Oracle Corporation recommends setting the **ESTIMATE_PERCENT** parameter of the DBMS_STATS gathering procedures to **DBMS_STATS.AUTO_SAMPLE_SIZE** to maximize performance gains while achieving necessary statistical accuracy. **AUTO_SAMPLE_SIZE** lets

Oracle determine the best sample size for good statistics. For example, to collect table and column statistics for all tables in the OE schema with auto-sampling:

EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('OE',DBMS_STATS.AUTO_SAMPLE_SIZE);

When we generate statistics for a table, column, or index, if the data dictionary already contains statistics for the object, then Oracle updates the existing statistics. Oracle also invalidates any currently parsed SQL statements that access the object.

The next time such a statement executes, the optimizer automatically chooses a new execution plan based on the new statistics. Distributed statements issued on remote databases that access the analyzed objects use the new statistics the next time Oracle parses them.

When we associate a statistics type with a column or domain index, Oracle calls the statistics collection method in the statistics type, if we analyze the column or domain index.

Oracle highly recommends to collect statistics from **DBMS_STATS** package than **ANALYZE** statement for cost based optimization. However, we must use ANALYZE statement to collect statistics of non cost based optimization.

For example to collect statistics of index we can use:

DBMS_STATS.GATHER_INDEX_STATS(<schema_name>, <index_name>);

or

ANALYZE INDEX <index_name> COMPUTE STATISTICS;

Gather Statistics with DBMS_STATS package:

The list of procedures in DBMS_STATS package are as follow:

Procedure	**Collects**
GATHER_INDEX_STATS	Index Statistics
GATHER_TABLE_STATS	Table, Column, and Index Statistics
GATHER_SCHEMA_STATS	Statistics for all objects in a schema
GATHER_SYSTEM_STATS	CPU and I/O statistics for the system
GATHER_DATABASE_STATS	Statistics for all objects in a database

Gathering System Statistics:

System statistics enable the optimizer to consider a **system's I/O** and **CPU** performance and utilization. For each plan, the optimizer computes, estimates for I/O and CPU costs. It is important to know system characteristics to pick the most efficient plan with optimal proportion between I/O and CPU cost. Oracle highly recommends to gather system statistics.

The two possible type of system statistics are:

1. **No Workload:**

When gathering noworkload stats, the database issues a series of random I/Os and tests the speed of the CPU.

Exec DBMS_STATS.GATHER_SYSTEM_STATS; -- noworkload

2. **Workload:**

When initiated using the start/stop or interval parameters, the database uses counters to keep track of all system operations, giving it an accurate idea of the performance of the system. If workload statistics are present, they will be used in preference to noworkload statistics.

Exec DBMS_STATS.GATHER_SYSTEM_STATS('start');

Exec DBMS_STATS.GATHER_SYSTEM_STATS(‘stop’);

Exec DBMS_STATS.GATHER_SYSTEM_STATS(

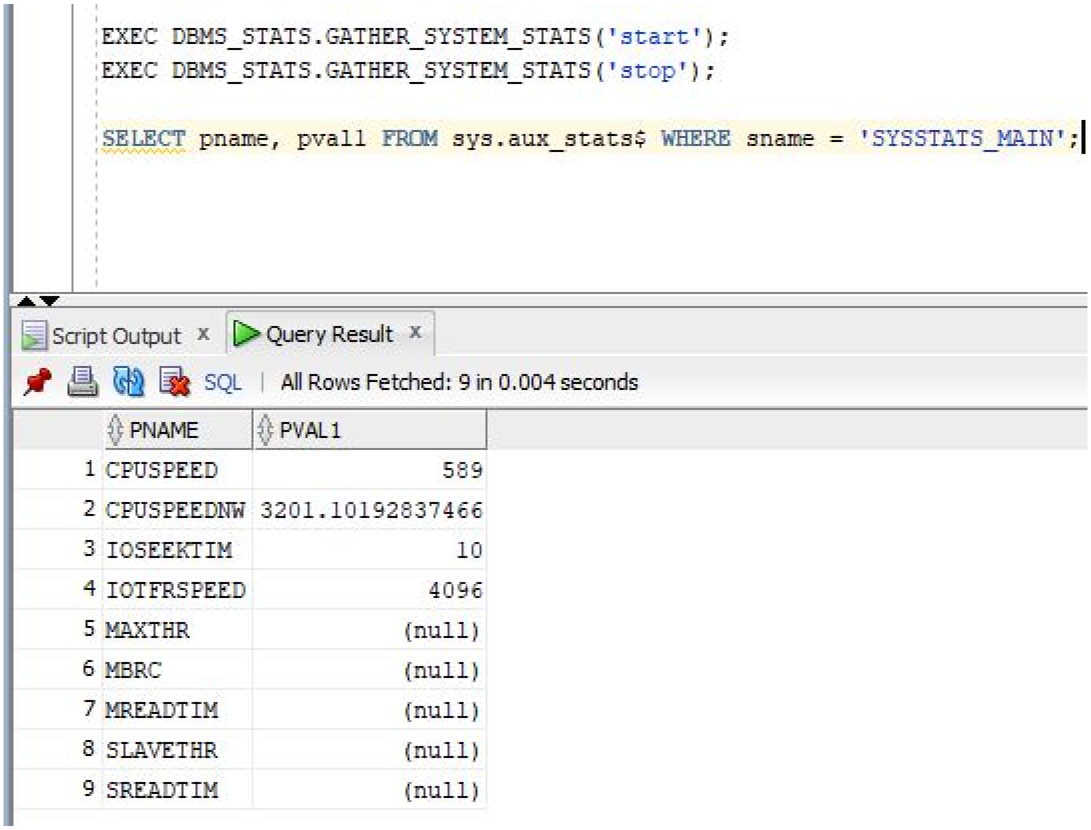
Gathering_mode => ‘interval’,

interval=>720 --Gathering of statistics ends after 720 minutes

);

We can view the current system statistics from **AUX_STATS$** table.

SELECT pname, pval1 FROM sys.aux_stats$ WHERE sname = 'SYSSTATS_MAIN';



```
EXEC DBMS_STATS.GATHER_SYSTEM_STATS('start');
EXEC DBMS_STATS.GATHER_SYSTEM_STATS('stop');

SELECT pname, pval1 FROM sys.aux_stats$ WHERE sname = 'SYSSTATS_MAIN';
```

All Rows Fetched: 9 in 0.004 seconds

	PNAME	PVAL1
1	CPUSPEED	589
2	CPUSPEEDNW	3201.10192837466
3	IOSEEKTIM	10
4	IOTFRSPEED	4096
5	MAXTHR	(null)
6	MBRC	(null)
7	MREADTIM	(null)
8	SLAVETHR	(null)
9	SREADTIM	(null)

Fig : System Statistics

We can **delete** system stat using:

EXEC DBMS_STATS.DELETE_SYSTEM_STATS;

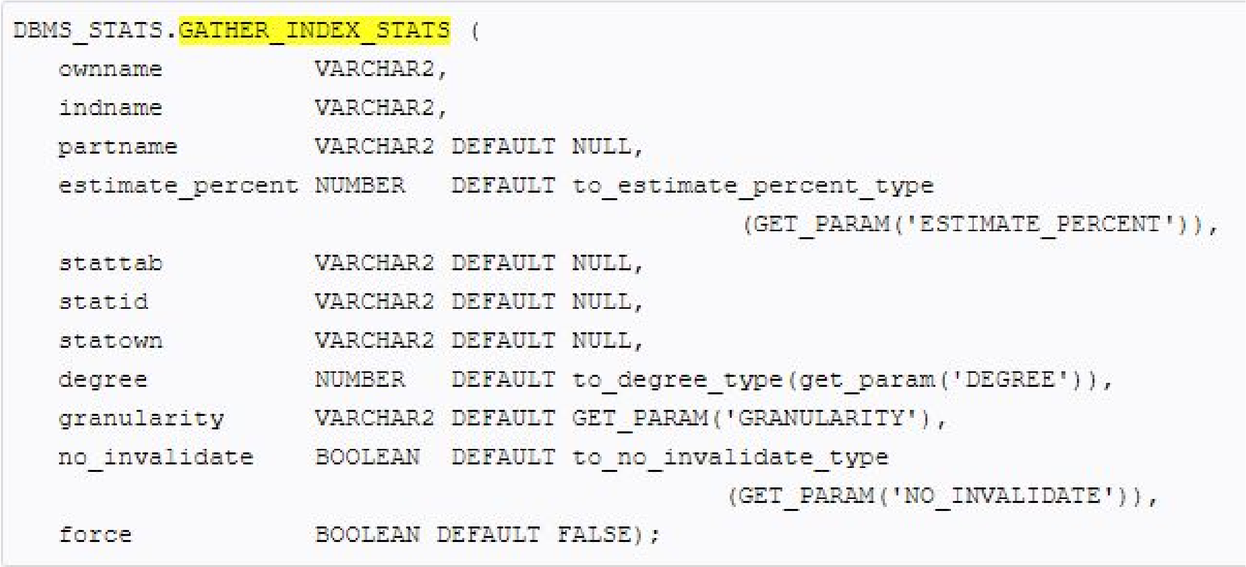
Gathering index statistics:

Index Statistics are **necessary** for rebuilding an index.

We can use **COMPUTE STATISTICS** clause to collect the statistics. Oracle always uses base tables when creating an index with the COMPUTE STATISTICS option.

If we do not use the **COMPUTE STATISTICS** clause, or if we have made significant changes to the data, then we should use the **DBMS_STATS.GATHER_INDEX_STATS** procedure to collect index statistics.

Syntax of Gather_index_stats:

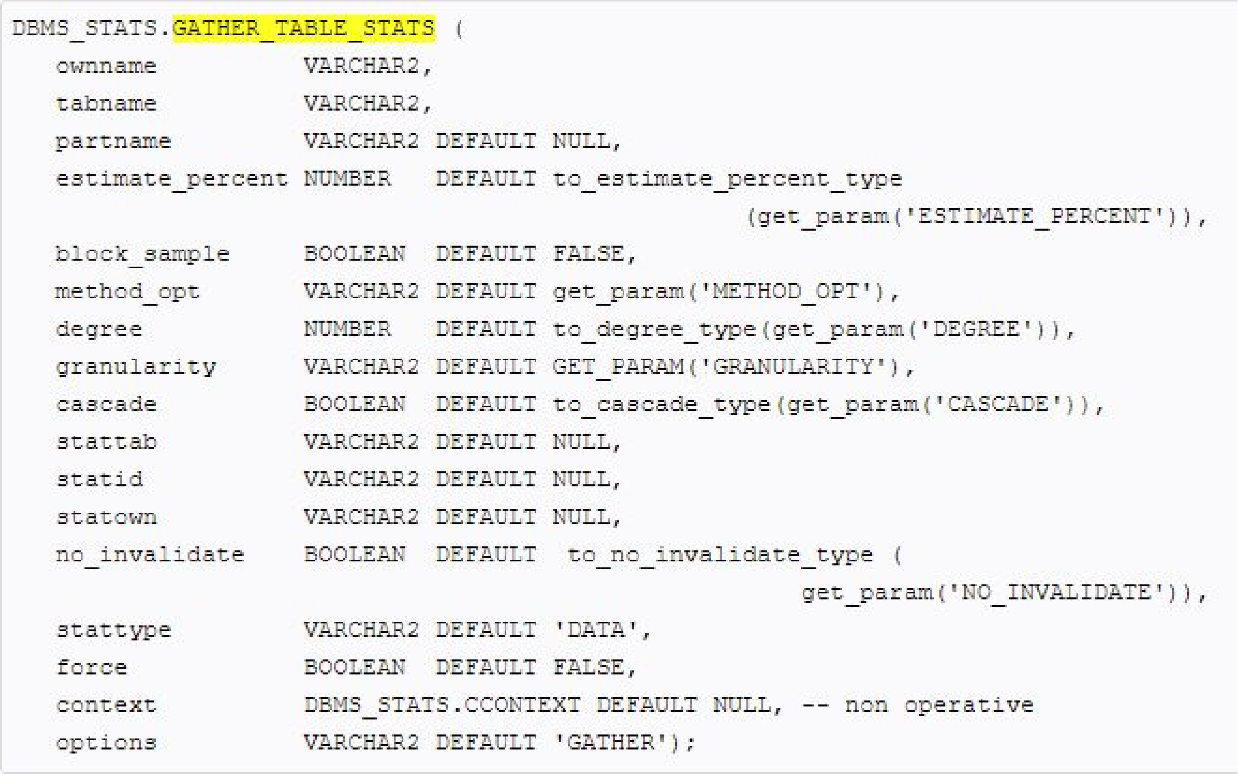


```
DBMS_STATS.GATHER_INDEX_STATS (
   ownname          VARCHAR2,
   indname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                                                (GET_PARAM('ESTIMATE_PERCENT')),
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL,
   degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
   granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
   no_invalidate    BOOLEAN  DEFAULT to_no_invalidate_type
                                               (GET_PARAM('NO_INVALIDATE')),
   force            BOOLEAN DEFAULT FALSE);
```

Fig: Syntax of DBMS_STATS.GATHER_INDEX_STATS

Oracle Corporation recommends setting the **ESTIMATE_PERCENT** parameter of the DBMS_STATS gathering procedures to **DBMS_STATS.AUTO_SAMPLE_SIZE** to maximize performance gains while achieving necessary statistical accuracy.

Gathering table statistics:



```
DBMS_STATS.GATHER_TABLE_STATS (
   ownname          VARCHAR2,
   tabname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT to_estimate_percent_type
                                                (get_param('ESTIMATE_PERCENT')),
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT get_param('METHOD_OPT'),
   degree           NUMBER   DEFAULT to_degree_type(get_param('DEGREE')),
   granularity      VARCHAR2 DEFAULT GET_PARAM('GRANULARITY'),
   cascade          BOOLEAN  DEFAULT to_cascade_type(get_param('CASCADE')),
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL,
   no_invalidate    BOOLEAN  DEFAULT  to_no_invalidate_type (
                                                    get_param('NO_INVALIDATE')),
   stattype         VARCHAR2 DEFAULT 'DATA',
   force            BOOLEAN  DEFAULT FALSE,
   context          DBMS_STATS.CCONTEXT DEFAULT NULL, -- non operative
   options          VARCHAR2 DEFAULT 'GATHER');
```

Fig : Syntax Gather Table Stats

For histogram:

method_opt	`METHOD_OPT` -
	● `FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]`
	● `FOR COLUMNS [column_clause] [size_clause]`
	
	`size_clause` is defined as `size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}`
	`column_clause` is defined as `column_clause := column_name | extension name | extension`
	
	- `integer` : Number of histogram buckets. Must be in the range [1,2048].
	- `REPEAT` : Collects histograms only on the

	columns that already have histograms – `AUTO` : Oracle determines the columns on which to collect histograms based on data distribution and the workload of the columns. – `SKEWONLY` : Oracle determines the columns on which to collect histograms based on the data distribution of the columns. - `column_name` : Name of a column - `extension` **:** can be either a column group in the format of (`column_name`, `Colume_name` [, ...]) or an expression The default is `FOR ALL COLUMNS SIZE AUTO`.

Transferring Stats:

We could transfer system stats but first we must create a stat table then export it.

EXEC DBMS_STATS.CREATE_STAT_TABLE(<schema_name>, <table_name>);

EXEC DBMS_STATS.EXPORT_SYSTEM_STATS(<source_schema_name>,<source_stat_table>, NULL, <destination_schema>);

This table can then be transferred to another server using your preferred method (**Export/Import, SQL*Plus COPY** etc.) and the stats imported into the data dictionary as follows.

EXEC DBMS_STATS.IMPORT_SYSTEM_STATS(<source_schema_name>,<source_stat_table>, NULL, <destination_schema>);

EXEC DBMS_STATS.DROP_STAT_TABLE(<schema_name>, <table_name>);

Similarly, we can transfer **schema** stats using

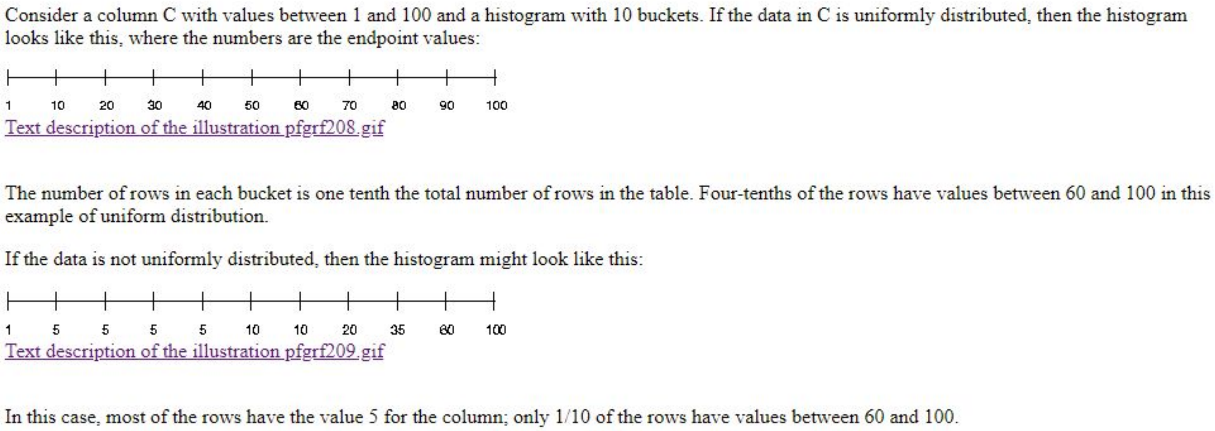
DBMS_STATS.EXPORT_SCHEMA_STATS(<source_schema_name>,<source_stat_table>, NULL, <destination_schema>));

DBMS_STATS.IMPORT_SCHEMA_STATS(<source_schema_name>,<source_stat_table>, NULL, <destination_schema>));

Histogram:

The cost-based optimizer can use data value histograms to get accurate estimates of the distribution of column data. A **histogram** partitions the values in the column into bands, so that all column values in a band fall within the same range. Histograms provide improved selectivity estimates in the presence of data skew, resulting in optimal execution plans with nonuniform data distributions.

The cost based optimizer uses height based histograms on a specified attributes (columns) to describe the distribution of **nonuniform** data. In height-based histogram, the columns are divided into bands so that **each bands** contains approximately **equal** number of **values**. Histogram provides the **end point** values associated with that column which might be very useful.



Consider a column C with values between 1 and 100 and a histogram with 10 buckets. If the data in C is uniformly distributed, then the histogram looks like this, where the numbers are the endpoint values:

Text description of the illustration pfgrf208.gif

The number of rows in each bucket is one tenth the total number of rows in the table. Four-tenths of the rows have values between 60 and 100 in this example of uniform distribution.

If the data is not uniformly distributed, then the histogram might look like this:

Text description of the illustration pfgrf209.gif

In this case, most of the rows have the value 5 for the column; only 1/10 of the rows have values between 60 and 100.

Fig: Histogram Distribution

When to use Histogram?

We should use Histogram on columns that are used frequently in **WHERE** clauses of queries and have a **highly skewed data distribution.**

Histograms are ***not*** useful for columns with the following characteristics:

- All predicates on the column use **bind variables**.
- The column data is **uniformly distributed.**
- The column is **unique** and is used only with **equality** predicates.

Creating a Histogram?

For example, to create a 10-bucket histogram on the SAL column of the emp table, issue the following statement:

EXECUTE DBMS_STATS.GATHER_TABLE_STATS ('scott','emp', METHOD_OPT => 'FOR COLUMNS SIZE 10 sal');

We would create a histogram on SAL column if there are abnormally large number of employees with the same salary and few employee with different salary(i.e. If the data is skewed).

The **SIZE** keyword declares the maximum number of bucket for the histogram. Oracle recommends to have the database automatically decide which columns need histograms. This is done by using **SIZE AUTO**.

Viewing the histogram

To view the histogram we have:

- DBA_HISTOGRAMS
- DBA_PART_HISTOGRAMS
- DBA_SUBPART_HISTOGRAMS
- DBA_TAB_COL_STATISTICS

Reference:

1. https://docs.oracle.com/cd/A97630_01/server.920/a96533/stats.htm
2. https://docs.oracle.com/database/121/ARPLS/d_stats.htm#ARPLS68582