



An Oracle White Paper
April 2012

Best Practices for Gathering Optimizer Statistics

Introduction	2
How to gather statistics	3
Automatic statistics gathering job	3
Manually statistics collection.....	3
ESTIMATE_PERCENT.....	4
METHOD_OPT	6
Pending Statistics	11
When to gather statistics	12
Automatic statistics gathering job	12
Manual statistics collection	13
Preventing "out of range" condition.....	14
Improving the efficiency of gathering statistics.....	15
Using parallelism	15
Incremental statistics	17
When not to gather statistics	18
Volatile tables	18
Global Temporary Tables	18
Intermediate work tables.....	19
Gathering other types of statistics.....	19
Dictionary statistics.....	19
Fixed object statistics	19
System statistics.....	20
Conclusion	20

Introduction

The Oracle Optimizer examines all of the possible plans for a SQL statement and picks the one with the lowest cost, where cost represents the estimated resource usage for a given plan. In order for the Optimizer to accurately determine the cost for an execution plan it must have information about all of the objects (table and indexes) accessed in the SQL statement as well as information about the system on which the SQL statement will be run.

This necessary information is commonly referred to as **Optimizer statistics**. Understanding and managing Optimizer statistics is key to optimal SQL execution. Knowing when and how to gather statistics in a timely manner is critical to maintaining acceptable performance. This whitepaper is the second of a two part series on Optimizer statistics. The first part of this series, [Understanding Optimizer Statistics](#), focuses on the concepts of statistics and will be referenced several times in this paper as a source of additional information. This paper will discuss in detail, when and how to gather statistics for the most common scenarios seen in an Oracle Database. The topics are

- How to gather statistics
- When to gather statistics
- Improving the efficiency of gathering statistics
- When not to gather statistics
- Gathering other types of statistics

How to gather statistics

The preferred method for gathering statistics in Oracle is to use the supplied automatic statistics-gathering job.

Automatic statistics gathering job

The job collects statistics for all database objects, which are missing statistics or have stale statistics by running an Oracle AutoTask task during a predefined maintenance window. Oracle internally prioritizes the database objects that require statistics, so that those objects, which most need updated statistics, are processed first.

The automatic statistics-gathering job uses the `DBMS_STATS.GATHER_DATABASE_STATS_JOB_PROC` procedure, which uses the same default parameter values as the other `DBMS_STATS.GATHER_*_STATS` procedures. The defaults are sufficient in most cases. However, it is occasionally necessary to change the default value of one of the statistics gathering parameters, which can be accomplished by using the `DBMS_STATS.SET_*_PREF` procedures. Parameter values should be changed at the smallest scope possible, ideally on a per-object bases. For example, if you want to change the staleness threshold for a specific table, so its statistics are considered stale when only 5% of the rows in the table have changed rather than the default 10%, you can change the `STALE_PERCENT` table preference for that one table using the `DBMS_STATS.SET_TABLE_PREFS` procedure. By changing the default value at the smallest scope you limit the amount of non-default parameter values that need to be manually managed.

```
SQL> BEGIN
 2  DBMS_STATS.SET_TABLE_PREFS('SH', 'SALES', 'STALE_PERCENT', '5');
 3  END;
 4  /

PL/SQL procedure successfully completed.
```

Figure 1. Using `DBMS_STATS.SET_TABLE_PREFS` procedure to change `STALE_PERCENT` to 5% on `SALES` table

Manually statistics collection

If you already have a well-established statistics gathering procedure or if for some other reason you want to disable automatic statistics gathering for your main application schema, consider leaving it on for the dictionary tables. You can do so by changing the value of `AUTOSTATS_TARGET` parameter to `ORACLE` instead of `AUTO` using `DBMS_STATS.SET_GLOBAL_PREFS` procedure.

```
SQL> BEGIN
 2  DBMS_STATS.SET_GLOBAL_PREFS('AUTOSTATS_TARGET', 'ORACLE');
 3  END;
 4  /

PL/SQL procedure successfully completed.
```

Figure 2. Changing the automatic statistics gather job to just gather dictionary statistics

To manually gather statistics you should use the PL/SQL package, `DBMS_STATS`, which replaces the now obsolete, `ANALYZE`¹ command for collecting statistics. The package `DBMS_STATS` provides multiple `DBMS_STATS.GATHER_*_STATS` procedures to gather statistics on both user schema objects as well as dictionary and fixed objects. Ideally you should let all of the parameters for these procedures default except for schema name and object name. The defaults and adaptive parameter settings chosen by the Oracle are sufficient in most cases.

```
SQL> BEGIN
  2  dbms_stats.gather_table_stats('SH','SALES');
  3  END;
  4  /

PL/SQL procedure successfully completed.
```

Figure 3. Using the `DBMS_STATS.GATHER_TABLE_STATS` procedure

As mentioned above, if it does become necessary to change the default value of one of the statistics gathering parameters, using the `DBMS_STATS.SET_*_PREF` procedures to make the change at the smallest scope possible, ideally on a per-object basis.

The two parameters most frequently changed from their default values are `ESTIMATE_PERCENT` and `METHOD_OPT`, especially in releases prior to Oracle Database 11g Release 1.

ESTIMATE_PERCENT

The most commonly asked questions around statistics gathering best practices is ‘what sample size should I use?’ This question relates to setting the `ESTIMATE_PERCENT` parameter of the `DBMS_STATS.GATHER_*_STATS` procedures. The `ESTIMATE_PERCENT` parameter determines the percentage of rows used to calculate the statistics. The most accurate statistics are gathered when all rows in the table are processed (i.e., 100% sample). However, the larger the sample size the longer the statistics gathering operation will take. So how do you come up with a sample size that provides accurate statistics in a timely manner?

ESTIMATE_PERCENT prior to Oracle Database 11g

In Oracle Database 10g, the default value for `ESTIMATE_PERCENT` changed from a 100% sample to `AUTO_SAMPLE_SIZE`. The goal of `AUTO_SAMPLE_SIZE` was for Oracle to determine the appropriate sample size for each table, each time statistics were gathered. This would allow Oracle to automatically change the sample size as the data within each table changed but still ensure statistics were gathered in a timely manner. This approach worked well for most tables but it had some issues if there was an extreme skew in the data. The `AUTO_SAMPLE_SIZE` algorithm often chose too small a sample size

¹ `ANALYZE` command should only be used to `VALIDATE` or `LIST CHAINED ROWS`.

when an extreme skew was present. In such cases it was still better to manually specify the `ESTIMATE_PERCENT`.

ESTIMATE_PERCENT in Oracle Database 11g

Oracle Database 11g introduced a new hash-based sampling algorithm that provides deterministic statistics, addressing the two key aspects of accuracy and speed². It has accuracy close to a 100% sample (“reading all data”) but with the cost of, at most, a 10% sample. This new algorithm is only used when `ESTIMATE_PERCENT` is set to `AUTO_SAMPLE_SIZE` (the default) in any of the `DBMS_STATS.GATHER_*_STATS` procedures.

The table below shows the results, of an easily reproducible test that gathers statistics with a 1% sample, a 100% sample, and `AUTO_SAMPLE_SIZE` on the `Lineitem` table (230GB) from the TPC-H benchmark with a scale factor of 300. The first line compares the elapsed time for each run, while the subsequent lines shows the number of distinct values (NDV) calculated for each run for two different columns, `L_ORDERKEY` and `L_COMMENT`.

	1% SAMPLE	AUTO_SAMPLE_SIZE	100% SAMPLE
Elapse time (sec)	797	1,908	18,772
NDV for L_ORDERKEY Column	225,000,000	450,000,000	450,000,000
NDV for L_COMMENT COLUMN	7,244,885	177,499,684	181,122,127

Figure 4. Comparison of the elapse time taken & statistic generated by a 1%, 100% sample & `AUTO_SAMPLE_SIZE`

In this scenario the new `AUTO_SAMPLE_SIZE` algorithm was 9 times faster than the 100% sample and only 2.4 times slower than the 1 % sample, while the quality of the statistics it produced were nearly identical to a 100% sample (not different enough to change the execution plans). Note the timings for this experiment may vary on your system and the larger the data set, the more speedup you will encounter from the new algorithm.

It is highly recommended from Oracle Database 11g onward you let `ESTIMATE_PERCENT` default. If you manually set the `ESTIMATE_PERCENT` parameter, even if it is set to 100%, you will get the old statistics gathering algorithm.

² For more information on the `AUTO_SAMPLE_SIZE` algorithm please refer to the VLDB paper, [Efficient and scalable statistics gathering for large databases in Oracle 11g](#)

METHOD_OPT

By far the most controversial parameter in the `DBMS_STATS.GATHER_*_STATS` procedures is the `METHOD_OPT` parameter. The `METHOD_OPT` parameter controls the creation of histograms³ during statistics collection. Histograms are a special type of column statistic created to provide more detailed information on the data distribution in a table column. So why are histograms such a controversial issue?

Histogram creation does extend the elapse time and the system resources needed for statistics collection but the far bigger concerns people have with histograms comes from their interaction with the bind peeking feature and how their presence affects the cardinality estimates for near popular values.

Histograms and bind peeking

The adverse affect of histograms on bind peeking has been relieved in Oracle Database 11g with the introduction of Adaptive Cursor Sharing but the fear still remains today. In order to explain how Adaptive Cursor Sharing addresses this problem, let's first examine the cause of the problem.

Prior to Oracle Database 11g histograms and bind peeking

Prior to Oracle Database 11g, when optimizing a SQL statement that contains bind variables in the `WHERE` clause the Optimizer peeks at the values of these bind variables on the first execution (during hard parse) of the statement. The Optimizer then determines the execution plan based on these initial bind values. On subsequent executions of the query, no peeking takes place (no hard parse happens), so the original execution plan, determined with the first set of bind values, will be used by all future executions, even if the values of the bind variables change.

The presence of a histogram on the column used in the expression with the bind variable will help to determine the most optimal execution plan for the initial set of bind values. Consequently, the execution plan for the same SQL statement can be different, depending on the values of the bind variables used for the initial hard parse.

There were two solutions to avoid this behavior: drop the histogram and stop collecting them in the future or disable the bind peeking. You can determine which approach suits your pre-Oracle Database 11g environment best, by answering a simple question; Do all of the SQL statements in this environment use bind variables or not?

³ More information on the creation of histograms can be found in part one of this series, [Understanding Optimizer Statistics](#).

Disabling histogram creation

If your environment only has SQL statements with bind variables then it is better to drop the existing histograms and disable histograms from being created in the future. Disabling histogram creation ensures the execution plan will not change depending on the bind variable value and will furthermore reduce the time it takes to gather statistics. Without the histograms on the column the Optimizer will assume a uniform distribution of rows across the distinct values in a column and will use NDV to determine the cardinality estimate when it peeks at the initial bind values in the SQL statements.

You can begin by dropping the existing statistics including the histogram using the

`DBMS_STATS.DELETE_TABLE_STATS` procedure.

```
SQL> BEGIN
 2  DBMS_STATS.DELETE_TABLE_STATS('SH','SALES');
 3  END;
 4  /
```

PL/SQL procedure successfully completed.

Figure 5. Drop statistics including histograms on the Sales table prior to 11g

Next, change the default value for the `METHOD_OPT` parameter to prevent histograms from being created in the future by using the `DBMS_STATS.SET_PARAM` procedure. This ensures that both the `DBMS_STATS.GATHER_*_STATS` procedures and the automatic statistics gathering job will not collect histograms in the future.

```
SQL> BEGIN
 2  DBMS_STATS.SET_PARAM(pname=>'METHOD_OPT', pval=>'FOR ALL COLUMNS SIZE 1');
 3  END;
 4  /
```

PL/SQL procedure successfully completed.

Figure 6. Change the default value of the `METHOD_OPT` parameter prior to 11g

Finally you can re-gather statistics for the effected object using the

`DBMS_STATS.GATHER_TABLE_STATS` procedure.

Note in Oracle database 11g you can drop unwanted histograms without dropping all column statistics by using `DBMS_STATS.DELETE_COLUMN_STATS` and setting the `col_stat_type` to histogram. You can also disable histogram creation on a single table or just a column within that table using the `DBMS_STATS.SET_TABLE_PREFS` procedure.

You should be aware that histograms can also be used for some join predicates and dropping them may have an adverse effect on join cardinality estimates. In these cases it may be safer to disable bind peeking.

Disabling bind peeking

If your environment has some SQL statements with bind variables and some with literal values then you should disable bind peeking. By disabling bind peeking you will prevent the Optimizer from peeking at the initial bind value and it will not use the histogram to determine the cardinality estimates

for the statement. Instead the Optimizer will assume a uniform distribution of rows across the distinct values in a column and will use NDV to determine the cardinality estimate. This will ensure a consistent execution plan for the statements with binds. But the SQL statements with literal values will still be able to take advantage of the histograms and get the most optimal plan. You can disable bind peeking by setting the underscore parameter `_OPTIM_PEEK_USER_BINDS` to `FALSE`.

Oracle Database 11g histograms and bind peeking

In Oracle Database 11g, the Optimizer has been enhanced to allow multiple execution plans to be used for a single statement with bind variables. This functionality is called Adaptive Cursor Sharing ⁴and relies on the monitoring of execution statistics to ensure the correct plan is used for each bind value.

On the first execution the Optimizer will peek the bind value and determine the execution plan based on the bind values selectivity, just like it did in previous release. The cursor will be marked bind sensitive if the Optimizer believes the optimal plan may depend on the value of the bind variable (for example, a histogram is present on the column or the predicate is a range, or `<`, `>`). When a cursor is marked bind sensitive, Oracle monitors the behavior of the cursor using different bind values, to determine if a different plan is called for.

If a different bind value is used in a subsequent execution, it will use the same execution plan because Oracle initially assumes it can be shared. However, the execution statistics for this new bind value will be recorded and compared to the execution statistics for the previous value. If Oracle determines that the new bind value caused the data volumes manipulated by the query to be significantly different it "adapts" and hard parses based on the new bind value on its next execution and the cursor is marked bind-aware. Each bind-aware cursor is associated with a selectivity range of the bind so that the cursor is only shared for a statement when the bind value in the statement is believed to fall within the range.

When another new bind value is used, the Optimizer tries to find a cursor it thinks will be a good fit, based on similarity in the bind value's selectivity. If it cannot find such a cursor, it will create a new one. If the plan for the new cursor is the same as an existing cursor, the two cursors will be merged to save space in the shared pool. And the selectivity range for that cursor will be increased to include the selectivity of the new bind.

By allowing multiple execution plans for a single SQL statement, histograms no longer have a negative impact for statements that use bind variables in Oracle Database 11g.

⁴ More information on Adaptive Cursor Sharing can be found in [Closing the Query Loop in Oracle 11g](#)

Histograms and near popular values

When the Optimizer encounters a where clause predicate on a column that has a histogram created on it, it uses different formulas to calculate the cardinality estimates based on the popularity of the literal value. For example, let's assume there was a height-balanced histogram created on the CUST_CITY_ID of the CUSTOMERS table in the SH sample schema and a query is issued that has a where clause predicate of CUST_CITY_ID = 51806. The Optimizer would first check to see how many buckets in the histogram have 51806 as their end point. In this case, the endpoint for bucket 136,137, 138 and 139 is 51806(info found in USER_HISTOGRAMS). Because the value is the endpoint of two or more buckets it is considered to be popular and the Optimizer will use the following formula to determine the cardinality estimate:

(Number of bucket endpoints / total number of buckets) X number of rows in the table

In this case 4/254 X 55500 = 874

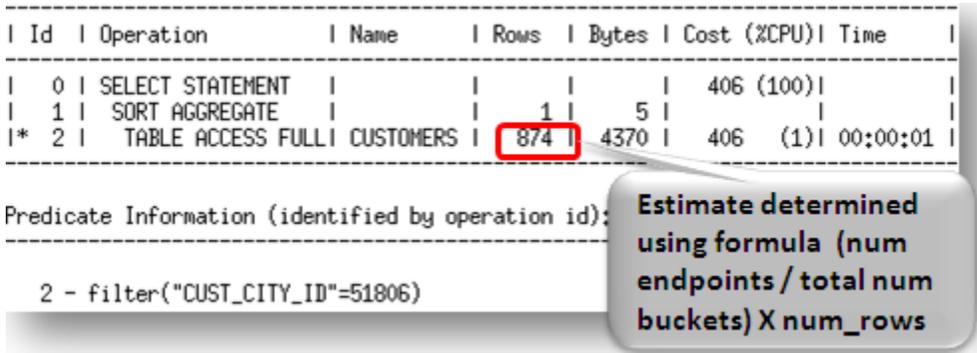


Figure 7. Height balanced histogram used for popular value cardinality estimate

However, if the predicate was CUST_CITY_ID = 52500, which is not the endpoint for any bucket then the Optimizer uses a different formula. For values that are the endpoint for only one bucket or are not an endpoint at all, the Optimizer uses the following formula:

DENSITY X number of rows in the table

where DENSITY is calculated 'on the fly' during optimization using an internal formula based on information in the histogram. The value for DENSITY seen in the dictionary view USER_TAB_COL_STATISTICS is not the value used by the Optimizer from Oracle Database 10.2.0.4 onwards. This value is recorded for backward compatibility, as this is the value used in Oracle Database 9i and earlier releases of 10g. Furthermore, if the parameter OPTIMIZER_FEATURES_ENABLE is set to version release earlier than 10.2.0.4, the value for DENSITY in the dictionary view will be used.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				405 (100)	
1	SORT AGGREGATE		1	5		
* 2	TABLE ACCESS FULL	CUSTOMERS	68	340	405 (1)	00:00:01

Predicate Information (identified by operation id):

2 - filter("CUST_CITY_ID"=52500)

Estimate determined using formula (density X num_rows)

Figure 8. Height balanced histogram used for non-popular value cardinality estimate

So, what happens for values that are nearly popular, the values that are the end point for just one bucket but almost span 2 buckets? These nearly popular values are classified as non-popular values and use the same formula as the non-popular values. For example, if the predicate was `CUST_CITY_ID = 52114`, it would use the cardinality estimates of 68 rows, the same as the non-popular value 52500. But 227 rows actually have `CUST_CITY_ID = 52114`.

The only way to make the Optimizer aware of these near popular values is to use dynamic sampling⁵. Dynamic sampling collects additional statement-specific object statistics during the optimization of a SQL statement. In this example the dynamic sampling hint is added to the query and the Optimizer get a much more accurate cardinality estimate.

```
SQL> SELECT /*+ dynamic_sampling(customers 2) */ count(CUST_ID)
2 FROM customers
3 WHERE cust_city_id =52114;
```

COUNT(CUST_ID)

227

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				405 (100)	
1	SORT AGGREGATE		1	5		
* 2	TABLE ACCESS STORAGE FULL	CUSTOMERS	248	1240	405 (1)	00:00:01

Note

- dynamic sampling used for this statement (level=2)

Figure 9. Use dynamic sampling to improve cardinality estimates for non-popular value in height balanced histogram

⁵ More information on dynamic sampling and how it works can be found in part one of this series, [Understanding Optimizer Statistics](#).

In the above sections, we have discussed the caveats around using histograms and the possible solutions in Oracle Database 10g. The recommendation from Oracle Database 11g onwards is to let `METHOD_OPT` default and to take advantage of Adaptive Cursor Sharing.

If you plan to manually set the `METHOD_OPT` parameter to a non-default value ensure you specify only the columns that really need a histogram. Setting `METHOD_OPT` to `FOR ALL COLUMNS SIZE 254` will cause Oracle to gather a histogram on every column. This will unnecessarily extend the elapsed time and the system resources needed for statistics gathering, as well as increasing the amount of space required to store the statistics.

You should also refrain from setting `METHOD_OPT` to `FOR ALL INDEX COLUMNS SIZE 254` as this will cause Oracle to gather histograms on every column used in an index, which again could waste system resources. This setting also has a nasty side effect of preventing Oracle from collecting basic column statistics for non-index columns.

Please refer to part one of this series, [Understanding Optimizer Statistics](#) to determine what type of histogram you should manually create.

Pending Statistics

When making changes to the default values of the parameter in the `DBMS_STATS.GATHER_*_STATS` procedures, it is highly recommended that you validate those changes before making the change in a production environment. If you don't have a full scale test environment you should take advantage of pending statistics. With pending statistics, instead of going into the usual dictionary tables, the statistics are stored in pending tables so that they can be enabled and tested in a controlled fashion before they are published and used system-wide. To activate pending statistics collection, you need to use one of the `DBMS_STATS.SET_*_PREFS` procedures to change value of the parameter `PUBLISH` from `TRUE` (default) to `FALSE` for the object(s) you wish to create pending statistics for. In the example below, pending statistics are enabled on the `SALES` table in the `SH` schema and then statistics are gathered on the `SALES` table.

```
SQL> BEGIN
  2  DBMS_STATS.SET_TABLE_PREFS('SH','SALES','PUBLISH','FALSE');
  3  END;
  4  /

PL/SQL procedure successfully completed.
```

Figure 10. Enable pending statistics by setting the publish preference to `FALSE`

Gather statistics on the object(s) as normal.

```
SQL> BEGIN
  2  dbms_stats.gather_table_stats('SH','SALES');
  3  END;
  4  /

PL/SQL procedure successfully completed.
```

Figure 11. Using the `DBMS_STATS.GATHER_TABLE_STATS` procedure

The statistics gathered for these objects can be displayed using the dictionary views called `USER_*_PENDING_STATS`.

You can enable the usage of pending statistics by issuing an alter session command to set the initialization parameter `OPTIMIZER_USE_PENDING_STATS` to `TRUE`. After enabling pending statistics, any SQL workload run in this session will use the new non-published statistics. For tables accessed in the workload that do not have pending statistics the Optimizer will use the current statistics in the standard data dictionary tables. Once you have validated the pending statistics, you can publish them using the procedure `DBMS_STATS.PUBLISH_PENDING_STATS`.

```
SQL> BEGIN
  2  DBMS_STATS.PUBLISH_PENDING_STATS('SH', 'SALES');
  3  END;
  4  /

PL/SQL procedure successfully completed.
```

Figure 12. Publishing pending statistics

When to gather statistics

In order to select an optimal execution plan the Optimizer must have representative statistics. Representative statistics are not necessarily up to the minute statistics but a set of statistics that help the Optimizer to determine the correct number of rows it should expect from each operation in the execution plan.

Automatic statistics gathering job

Oracle automatically collects statistics for all database objects, which are missing statistics or have stale statistics during a predefined maintenance window (10pm to 2am weekdays and 6am to 2am at the weekends).

You can change the maintenance window that the job will run in via Enterprise Manager or using the `DBMS_SCHEDULER` and `DBMS_AUTO_TASK_ADMIN` packages.

ORACLE Enterprise Manager 11g
Database Control

Cluster Database: DBM > Automated Maintenance Tasks >

Automated Maintenance Tasks Configuration

Global Status Enabled Disabled

Task Settings

Optimizer Statistics Gathering Enabled Disabled [Configure](#)

Segment Advisor Enabled Disabled

Automatic SQL Tuning Enabled Disabled [Configure](#)

Maintenance Window Group Assignment

[Edit Window Group](#)

Window	Optimizer Statistics Gathering		Segment Advisor		Automatic SQL Tuning	
	Select All	Select None	Select All	Select None	Select All	Select None
THURSDAY_WINDOW	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
FRIDAY_WINDOW	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SATURDAY_WINDOW	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SUNDAY_WINDOW	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
MONDAY_WINDOW	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TUESDAY_WINDOW	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
WEDNESDAY_WINDOW	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 13. Changing the maintenance window during which the auto stats gathering job runs

If you already have a well-established statistics gathering procedure or if for some other reason you want to disable automatic statistics gathering you can disable the task altogether:

```
SQL> BEGIN
 2  DBMS_AUTO_TASK_ADMIN.DISABLE(
 3  client_name => 'auto optimizer stats collection',
 4  operation => NULL,
 5  window_name => NULL);
 6  END;
 7  /

PL/SQL procedure successfully completed.
```

Figure 14. Disabling the automatic statistics gather job

Manual statistics collection

If you plan to manually maintain Optimizer statistics you will need to determine when statistics should be gathered.

You can determine when statistics should be gathered based on staleness, as it is for the automatic job, or based on when new data is loaded in your environment. It is not recommended to continually re-gather statistics if the underlying data has not changed significantly as this will unnecessarily waste system resources.

If data is only loaded into your environment during a pre-defined ETL or ELT job then the statistics gathering operations can be scheduled as part of this process. Note if you are using partition exchange loads and wish to take advantage of incremental statistics⁶, you will need to gather statistics after the exchange procedure and not before as in previous releases.

However, if your environment has more trickle feeds or online transactions that only insert a small number of rows but these operations occur throughout the day, you will need to determine when your statistics are stale and then trigger the statistics gathering job. If you plan to rely on the `STALE_STATS` column in `USER_TAB_STATISTICS` to determine if statistics are stale you should be aware that this information is updated on a daily basis only. If you need more timely information on what DML has occurred on your tables you will need to look in `USER_TAB_MODIFICATIONS`, which lists the number of `INSERTS`, `UPDATES`, and `DELETES` that occurs on each table, whether the table has been truncated (`TRUNCATED` column) and calculate staleness yourself. Again, you should note this information is automatically updated, from memory, periodically. If you need the latest information you will need to manually flush the information using the `DBMS_STATS.FLUSH_DATABASE_MONITORING_INFO` function.

Preventing "out of range" condition

Regardless of whether you use the automatic statistics gathering job or you manually gather statistics, if end-users start to query newly inserted data before statistics have been gathered, it is possible to get a suboptimal execution plan due to stale statistics, even if less than 10% of the rows have changed in the table. One of the most common cases of this occurs when the value supplied in a where clause predicate is outside the domain of values represented by the [minimum, maximum] column statistics. This is commonly known as an 'out-of-range' error. In this case, the Optimizer prorates the selectivity based on the distance between the predicate value, and the maximum value (assuming the value is higher than the max), that is, the farther the value is from the maximum or minimum value, the lower the selectivity will be.

This scenario is very common with range partitioned tables. A new partition is added to an existing range partitioned table, and rows are inserted into just that partition. End-users begin to query this new data before statistics have been gathered on this new partition. For partitioned tables, you can use the `DBMS_STATS.COPY_TABLE_STATS`⁷ procedure (available from Oracle Database 10.2.0.4 onwards) to prevent "Out of Range" conditions. This procedure copies the statistics of a representative source [sub]

⁶ More information on Incremental Statistics can be found in part one of this series, [Understanding Optimizer Statistics](#).

⁷ More information on `DBMS_STATS.COPY_TABLE_STATS` can be found in part one of this series, [Understanding Optimizer Statistics](#).

partition to the newly created and empty destination [sub] partition. It also copies the statistics of the dependent objects: columns, local (partitioned) indexes, etc. and sets the high bound partitioning value as the max value of the partitioning column and high bound partitioning value of the previous partition as the min value of the partitioning column. The copied statistics should only be considered as temporary solution until it is possible to gather accurate statistics for the partition. Copying statistics should not be used as a substitute for actually gathering statistics.

Note by default, `DBMS_STATS.COPY_TABLE_STATS` only adjust partition statistics and not global or table level statistics. If you want the global level statistics to be updated for the partition column as part of the copy you need to set the `flags` parameter of the `DBMS_STATS.COPY_TABLE_STATS` to 8.

For non-partitioned tables you can manually set the max value for a column using the `DBMS_STATS.SET_COLUMN_STATS` procedure. This approach is not recommended in general and is not a substitute for actually gathering statistics.

Improving the efficiency of gathering statistics

As data volumes grow and maintenance windows shrink, it is more important than ever to gather statistics in a timely manner. Oracle offers a variety of ways to speed up the statistics collection, from parallelizing the statistics gathering operations to generating statistics rather than collecting them.

Using parallelism

Parallelism can be leveraged in several ways for statistics collection

- Intra object parallelism
- Inter object parallelism
- A combination of both intra and inter object parallelism

Intra object parallelism

Intra object parallelism is controlled by the `DEGREE` parameter in the `DBMS_STATS.GATHER_*_STATS` procedures. The `DEGREE` parameter controls the number of parallel server processes that will be used to gather the statistics.

By default Oracle uses the same number of parallel server processes specified as an attribute of the table in the data dictionary (Degree of Parallelism). All tables in an Oracle database have this attribute set to 1 by default. It may be useful to explicitly set this parameter for the statistics collection on a large table to speed up statistics collection.

Alternatively you can set `DEGREE` to `AUTO_DEGREE`; Oracle will automatically determine the appropriate number of parallel server processes that should be used to gather statistics, based on the size of an object. The value can be between 1 (serial execution) for small objects to `DEFAULT_DEGREE (PARALLEL_THREADS_PER_CPU X CPU_COUNT)` for larger objects.

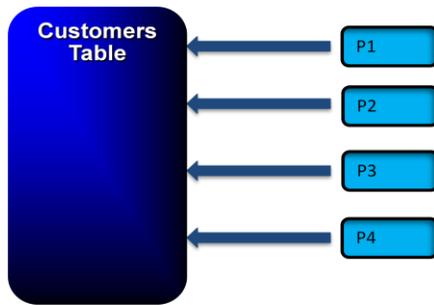


Figure 15. Use Intra object parallelism via the `DEGREE` parameter of the `DBMS_STATS.GATHER_*_STATS` procedures

You should note that setting the `DEGREE` for a partitioned table means that multiple parallel server processes will be used to gather statistics on each partition but the statistics will not be gathered concurrently on the different partitions. Statistics will be gathered on each partition one after the other.

Inter object parallelism

In Oracle Database 11.2.0.2, inter object parallelism was introduced and is controlled by the global statistics gathering preference `CONCURRENT`⁸. When `CONCURRENT` is set to `TRUE`, Oracle employs the Oracle Job Scheduler and Advanced Queuing components to create and manage multiple statistics gathering jobs concurrently. Gathering statistics on multiple tables and (sub)partitions concurrently can reduce the overall time it takes to gather statistics by allowing Oracle to fully utilize a multi-processor environment.

The maximum number of active concurrent statistics gathering jobs is controlled by the `JOB_QUEUE_PROCESSES` parameter. By default the `JOB_QUEUE_PROCESSES` is set to 1000. Typically this is too high for a `CONCURRENT` statistics gathering operation especially if parallel execution will also be employed. A more appropriate value would be 2 X total number of CPU cores (this is a per node parameter in a RAC environment). You need to make sure that you set this parameter system-wide (`ALTER SYSTEM ...` or in `init.ora` file) rather than at the session level (`ALTER SESSION`).

Combining Intra and Inter parallelism

Each of the statistics gathering jobs in a concurrent statistics gather operation can execute in parallel. Combining concurrent statistics gathering and parallel execution can greatly reduce the time it takes to gather statistics.

⁸ More information on concurrent statistics gathering can be found in part one of this series, [Understanding Optimizer Statistics](#).

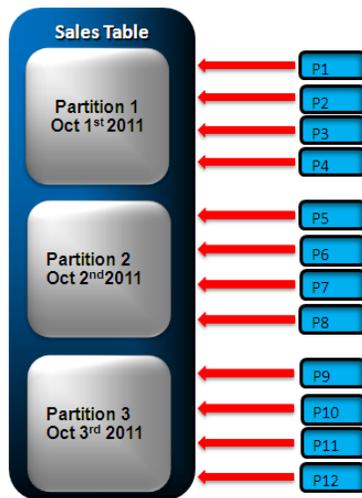


Figure 16. Use Inter and Intra object parallelism to speed up a `DBMS_STATS.GATHER_TABLE_STATS` on a partitioned table.

When using parallel execution as part of a concurrent statistics gathering you should disable the `PARALLEL_ADAPTIVE_MULTI_USER` initialization parameter to prevent the parallel jobs from being down graded to serial. Again this should be done at a system level and not at a session level. That is;

```
SQL> ALTER SYSTEM SET parallel_adaptive_multi_user=false;
System altered.
```

Figure 17. Disable `parallel_adaptive_multi_user` parameter

Incremental statistics

Gathering statistics on partitioned tables consists of gathering statistics at both the table level (global statistics) and (sub)partition level. If the `INCREMENTAL`⁹ preference for a partitioned table is set to `TRUE`, the `DBMS_STATS.GATHER_*_STATS` parameter `GRANULARITY` includes `GLOBAL`, and `ESTIMATE_PERCENT` is set to `AUTO_SAMPLE_SIZE`, Oracle will accurately derive all global level statistics by scanning only those partitions that have been added or modified, and not the entire table. Incremental global statistics works by storing a *synopsis* for each partition in the table. A synopsis is statistical metadata for that partition and the columns in the partition. Aggregating the partition level statistics and the synopses from each partition will accurately generate global level statistics, thus eliminating the need to scan the entire table. When a new partition is added to the table, you only need

⁹ More information on Incremental Statistics can be found in part one of this series, [Understanding Optimizer Statistics](#).

to gather statistics for the new partition. The table level statistics will be automatically and accurately calculated using the new partition synopsis and the existing partitions' synopses.

Note partition statistics are not aggregated from subpartition statistics when incremental statistics are enabled.

When not to gather statistics

Although the Optimizer needs accurate statistics to select an optimal plan, there are scenarios where gathering statistics can be difficult, too costly, or cannot be accomplished in a timely manner and an alternative strategy is required.

Volatile tables

A volatile table is one where the volume of data changes dramatically over time. For example, an orders queue table, which at the start of the day the table is empty. As the day progresses and orders are placed the table begins to fill up. Once each order is processed it is deleted from the tables, so by the end of the day the table is empty again.

If you relied on the automatic statistics gather job to maintain statistics on such tables then the statistics would always show the table was empty because it was empty over night when the job ran. However, during the course of the day the table may have hundreds of thousands of rows in it.

In such cases it is better to gather a representative set of statistics for the table during the day when the table is populated and then lock them. Locking the statistics will prevent the automatic statistics gathering job from over writing them. Alternatively, you could rely on dynamic sampling to gather statistics on these tables. The Optimizer uses dynamic sampling during the compilation of a SQL statement to gather basic statistics on the tables before optimizing the statement. Although the statistics gathered by dynamic sampling are not as high a quality or as complete as the statistics gathered using the `DBMS_STATS` package, they should be good enough in most cases.

Global Temporary Tables

Global temporary tables are often used to store intermediate results in an application context. A global temporary table shares its definition system-wide with all users with appropriate privileges, but the data content is always session-private. No physical storage is allocated unless data is inserted into the table. A global temporary table can be transaction specific (delete rows on commit) or session-specific (preserves rows on commit). Gathering statistics on transaction specific tables leads to the truncation of the table. In contrast, it is possible to gather statistics on a global temporary table (that persist rows); however the statistics gathered will only be based on the session-private data content of the user collecting the statistics but the statistics will be used by all sessions accessing that table.

If you have a global temporary table that persists rows, and each session will have approximately the same amount of data with roughly the same values then you should gather a representative set of

statistics for the table in one session and lock them. Thus preventing other sessions from over-riding them. Note the automatic statistics gathering job does not collect statistics on global temporary tables.

However, if each session could have drastically different data or the global temporary table does delete on commit, you should use dynamic sampling.

Intermediate work tables

Intermediate work tables are typically seen as part of an ETL process or a complex transaction. These tables are written to only once, read once, and then truncated or deleted. In such cases the cost of gathering statistics outweighs the benefit, since the statistics will be used just once. Instead dynamic sampling should be used in these cases. It is recommended you lock statistics on intermediate work tables that are persistent to prevent the automatic statistics gathering job from attempting to gather statistics on them.

Gathering other types of statistics

Since the Cost Based Optimizer is now the only supported optimizer, all tables in the database need to have statistics, including all of the dictionary tables (tables owned by 'SYS', SYSTEM, etc, and residing in the system and SYSAUX tablespace) and the x\$ tables used by the dynamic v\$ performance views.

Dictionary statistics

Statistics on the dictionary tables are maintained via the automatic statistics gathering job run during the nightly maintenance window. It is highly recommended that you allow Oracle to automatic statistics gather job to maintain dictionary statistics even if you choose to switch off the automatic statistics gathering job for your main application schema. You can do this by changing the value of AUTOSTATS_TARGET to ORACLE instead of AUTO using the procedure DBMS_STATS.SET_GLOBAL_PREFS.

```
SQL> BEGIN
  2  DBMS_STATS.SET_GLOBAL_PREFS('AUTOSTATS_TARGET', 'ORACLE');
  3  END;
  4  /

PL/SQL procedure successfully completed.
```

Figure 18. Modify the automatic statistics gather job to gather dictionary statistics only

Fixed object statistics

The automatic statistics gathering job does not gather fixed object statistics. And unlike other database tables, dynamic sampling is not automatically used for SQL statements involving X\$ tables when optimizer statistics are missing. The Optimizer uses predefined default values for the statistics if they are missing. These defaults may not be representative and could potentially lead to a suboptimal execution plan, which could cause severe performance problems in your system. It is for this reason that we strongly recommend you manually gather fixed objects statistics.

You can collect statistics on fixed objects using `DBMS_STATS.GATHER_FIXED_OBJECTS_STATS` procedure. Because of the transient nature of the x\$ tables it is important that you gather fixed object statistics when there is a representative workload on the system. This may not always be feasible on large systems due to additional resources needed to gather the statistics. If you can't do it during peak load you should do it after the system has warmed up and the three key types of fixed object tables have been populated:

- Structural data - for example, views covering datafiles, controlfile contents, etc.
- Session based data - for example, `v$session`, `v$access`, etc.
- Workload data - for example, `v$sql`, `v$sql_plan`, etc

It is recommended that you re-gather fixed object statistics if you do a major database or application upgrade, implement a new module, or make changes to the database configuration. For example if you increase the SGA size then all of the x\$ tables that contain information about the buffer cache and shared pool may change significantly, such as x\$ tables used in `v$buffer_pool` or `v$shared_pool_advice`.

System statistics

System statistics enable the Optimizer to more accurately cost each operation in an execution plan by using information about the actual system hardware executing the statement, such as CPU speed and IO performance. System statistics are enabled by default, and are automatically initialized with default values; these values are representative for most systems.

Conclusion

In order for the Optimizer to accurately determine the cost for an execution plan it must have accurate statistics about all of the objects (table and indexes) accessed in the SQL statement and information about the system on which the SQL statement will be run. This two part white paper series explains in detail what Optimizer statistics are necessary, how they are used, and the different statistics collection method available to you.

By using a combination of the automatic statistics gathering job and the other techniques described in this paper a DBA can maintain an accurate set of statistics for their environment and ensure the Optimizer always has the necessary information in order to select the most optimal plan. Once a statistics gathering strategy has been put in place, changes to the strategy should be done in a controlled manner and take advantage of key features such as pending statistics to ensure they do not have an adverse effect on application performance.



Best Practices for Gathering Optimizer
Statistics

April 2012

Author: Maria Colgan

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2011, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd. 1010

Hardware and Software, Engineered to Work Together