Best Practices for Gathering Optimizer Statistics with Oracle Database 12c Release 2

ORACLE WHITE PAPER | JUNE 2017

Table of Contents

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 part one of this series, *Understanding Optimizer Statistics with Oracle Database 12c*, 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 quality of statistics
- Gathering statistics more quickly
- When not to gather statistics
- Gathering other types of statistics

How to Gather Statistics

Strategy

The preferred method for gathering statistics in Oracle is to use the automatic statistics gathering. If you already have a well-established, manual statistics gathering procedure then you might prefer to use that instead. Whatever method you choose to use, start by considering whether the default global preferences meet your needs. In most cases they will, but if you want to change anything then you can do that with `SET_GLOBAL_PREFS`. Once you have done that, you can override global defaults where necessary using the `DBMS_STATS` "set preference" procedures. For example, use `SET_TABLE_PREFS` on tables that require incremental statistics or a specific set of histograms. In this way, you will have *declared* how statistics are to be gathered, and there will be no need to tailor parameters for individual "gather stats" operations. You will be free to use default parameters for gather table/schema/database stats and be confident that the statistics policy you have chosen will be followed. What's more, you will be able to switch freely between using auto and manual statistics gathering.

This section covers how to implement this strategy.

Automatic Statistics Gathering

The Oracle database collects statistics for database objects that are missing statistics or have "stale" (out of date) statistics. This is done by an automatic task that executes 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 basis. 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. For example, here's how you can change STALE_PRECENT to 5% on the SALES table:

```
exec dbms_stats.set_table_prefs(user,'SALES','STALE_PERCENT','5')
```

To check what preferences have been set, you can use the DBMS_STATS.GET_PREFS function. It takes three arguments; the name of the parameter, the schema name, and the table name:

```
select dbms_stats.get_prefs('STALE_PERCENT',user,'SALES') stale_percent
from   dual;

STALE_PERCENT
-------------
5
```

Setting DBMS_STATS Preferences

As indicated above, it is possible to set DBMS_STATS preferences to target specific objects and schemas to modify the behavior of auto statistics gathering where necessary. You can specify a particular non-default parameter value for an individual DBMS_STATS.GATHER_*_STATS command, but the recommended approach is to override the defaults where necessary using "targeted" DBMS_STATS.SET_*_PREFS procedures.

A parameter override can be specified at a table, schema, database, or global level using one of the following procedures (noting that AUTOSTATS_TARGET and CONCURRENT can only be modified at the global level):

```
SET_TABLE_PREFS
SET_SCHEMA_PREFS
SET_DATABASE_PREFS
SET_GLOBAL_PREFS
```

Traditionally, the most commonly overridden preferences have been ESTIMATE_PERCENT (to control the percentage of rows sampled) and METHOD_OPT (to control histogram creation), but estimate percent is now better left at its default value for reasons covered later in this section.

The SET_TABLE_PREFS procedure allows you to change the default values of the parameters used by the DBMS_STATS.GATHER_*_STATS procedures for the specified table only.

The SET_SCHEMA_PREFS procedure allows you to change the default values of the parameters used by the DBMS_STATS.GATHER_*_STATS procedures for all of the existing tables in the specified schema. This procedure actually calls the SET_TABLE_PREFS procedure for each of the tables in the specified schema. Since it uses SET_TABLE_PREFS, calling this procedure will not affect any new objects created after it has been run. New objects will pick up the GLOBAL preference values for all parameters.

The SET_DATABASE_PREFS procedure allows you to change the default values of the parameters used by the DBMS_STATS.GATHER_*_STATS procedures for all of the user-defined schemas in the database. This procedure actually calls the SET_TABLE_PREFS procedure for each table in each user-defined schema. Since it uses SET_TABLE_PREFS, this procedure will not affect any new objects created after it has been run. New objects will pick up the GLOBAL preference values for all parameters. It is also possible to include the Oracle owned schemas (sys, system, etc) by setting the ADD_SYS parameter to TRUE.

The SET_GLOBAL_PREFS procedure allows you to change the default values of the parameters used by the DBMS_STATS.GATHER_*_STATS procedures for any object in the database that does not have an existing table preference. All parameters default to the global setting unless there is a table preference set, or the parameter is explicitly set in the GATHER_*_STATS command. Changes made by this procedure will affect any new objects created after it has been run. New objects will pick up the GLOBAL_PREFS values for all parameters.

The DBMS_STATS.GATHER_*_STATS procedures and the automated statistics gathering task obeys the following hierarchy for parameter values; parameter values explicitly set in the command overrule everything else. If the parameter has not been set in the command, we check for a table level preference. If there is no table preference set, we use the GLOBAL preference.

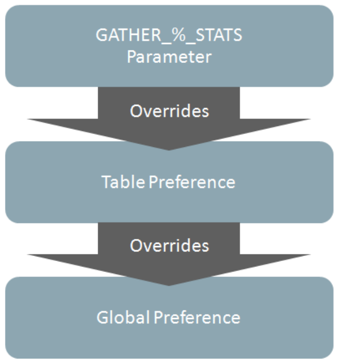


Figure 1: DBMS_STATS.GATHER_*_STATS hierarchy for parameter values

Oracle Database 12 Release 2 includes a new DBMS_STATS preference called PREFERENCE_OVERRIDES_PARAMETER. Its effect is illustrated in Figure 2. When this preference is set to TRUE, it allows preference settings to override DBMS_STATS parameter values. For example, if the global preference ESTIMATE_PERCENT is set to DBMS_STATS.AUTO_SAMPLE_SIZE, it means that this best-practice setting will be used even if existing manual statistics gathering procedures use a different parameter setting (for example, a fixed percentage sample size such as 10%).

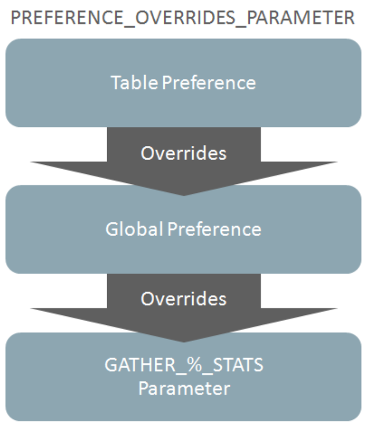


Figure 2: Using DBMS_STATS preference PREFERENCE_OVERRIDES_PARAMETER

ESTIMATE_PERCENT

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. a 100% sample), often referred to as computed statistics. Oracle Database 11g introduced a new sampling algorithm that is hash based and provides deterministic statistics. This new approach has an accuracy close to a 100% sample but with the cost of, at most, a 10% sample. The new algorithm is used when ESTIMATE_PERCENT is set to AUTO_SAMPLE_SIZE (the default) in any of the DBMS_STATS.GATHER_*_STATS procedures. Prior to Oracle Database 11g, DBAs often set the ESTIMATE_PRECENT parameter to a low value to ensure that the statistics would be gathered quickly. However, without detailed testing, it is difficult to know which sample size to use to get accurate statistics. It is highly recommended that from Oracle Database 11g onwards that the default AUTO_SAMPLE_SIZE is used for ESTIMATE_PRECENT. This is especially important because the new Oracle Database 12c histogram types, HYBRID and Top-Frequency, can only be created if an auto sample size is used.

Many systems still include old statistics gathering scripts that manually set estimate percent, so when upgrading to Oracle Database 12c Release 2, consider using the PREFERENCE_OVERRIDES_PARAMETER preference (see above) to enforce the use of auto sample size.

METHOD_OPT

The METHOD_OPT parameter controls the creation of histograms[1] 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.

The default and recommended value for METHOD_OPT is 'FOR ALL COLUMNS SIZE AUTO', which means that histograms will be created for columns that are likely to benefit from having [2]them. A column is a candidate for a histogram if it is used in equality or range predicates such as WHERE col1= 'X' or WHERE col1 BETWEEN 'A' and 'B' and, in particular, if it has a skew in the distribution of column values. The optimizer knows which columns are used in query predicates because this information is tracked and stored in the dictionary table SYS.COL_USAGE$.

Some DBAs prefer to tightly control when and what histograms are created. The recommended approach to achieve is to use SET_TABLE_PREFS to specify which histograms to create on a table-by-table basis. For example, here is how you can specify that SALES must have histograms on col1 and col2 *only*:

```
begin
   dbms_stats.set_table_prefs(
       user,
       'SALES',
       'method_opt',
       'for all columns size 1 for columns size 254 col1 col2');
end;
/
```

1 More information on the creation of histograms can be found in part one of this white paper series: *Understanding Optimizer Statistics Oracle Database 12c Release 2*.

It is possible to specify columns that *must* have histograms (col1 and col2) and, in addition, allow the optimizer to decide if additional histograms are useful:

```
begin
   dbms_stats.set_table_prefs(
       user,
       'SALES',
       'method_opt',
       'for all columns size auto for columns size 254 col1 col2');
end;
/
```

Histogram creation is disabled if `METHOD_OPT` is set to `'FOR ALL COLUMNS SIZE 1'`. For example, you can change the `DBMS_STATS` global preference for `METHOD_OPT` so that histograms are not created by default:

```
begin
   dbms_stats.set_global_prefs(
       'method_opt',
       'for all columns size 1');
end;
/
```

Unwanted histograms can be dropped without dropping all column statistics by using `DBMS_STATS.DELETE_COLUMN_STATS` and setting the *col_stat_type* to ‘HISTOGRAM’.

Manual 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.

```
exec dbms_stats.set_global_prefs('autostats_target','oracle')
```

To manually gather statistics you should use the PL/SQL DBMS_STATS package. The obsolete, ANALYZE command should not be used. The package DBMS_STATS provides multiple DBMS_STATS.GATHER_*_STATS procedures to gather statistics on 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:

```
exec dbms_stats.gather_table_stats('sh','sales')
```

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 bases.

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:

```
exec dbms_stats.set_table_prefs('sh','sales','publish','false')
```

Gather statistics on the object(s) as normal:

```
exec dbms_stats.gather_table_stats('sh','sales')
```

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.

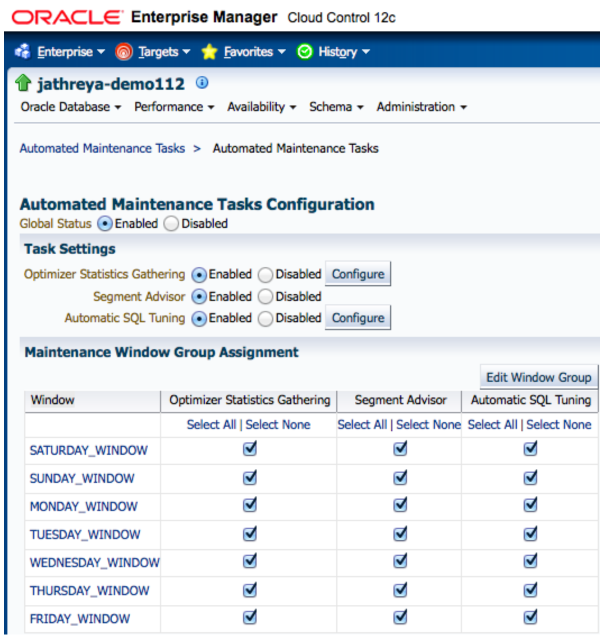
```
exec dbms_stats.publish_pending_stats('sh','sales')
```

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 Task

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.



Automated Maintenance Tasks Configuration

Global Status [✓ Enabled] Disabled

Task Settings

Optimizer Statistics Gathering [✓ Enabled] Disabled

Segment Advisor [✓ Enabled] Disabled

Automatic SQL Tuning [✓ Enabled] Disabled

Maintenance Window Group Assignment

Window	Optimizer Statistics Gathering	Segment Advisor	Automatic SQL Tuning
	Select All | Select None	Select All | Select None	Select All | Select None
SATURDAY_WINDOW	✓	✓	✓
SUNDAY_WINDOW	✓	✓	✓
MONDAY_WINDOW	✓	✓	✓
TUESDAY_WINDOW	✓	✓	✓
WEDNESDAY_WINDOW	✓	✓	✓
THURSDAY_WINDOW	✓	✓	✓
FRIDAY_WINDOW	✓	✓	✓

Figure 3: 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:

```
begin
   dbms_auto_task_admin.disable(
      client_name=>'auto optimizer stats collection',
      operation=>null,
      window_name=>null);
end;
/
```

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. You should try and take advantage of online statistics gathering and incremental statistics as part of your statistics maintenance strategy.

Online Statistics Gathering

In Oracle Database 12*c,* online statistics gathering “piggybacks” statistics gather as part of a direct-path data loading operation such as, create table as select (CTAS) and insert as select (IAS) operations. Gathering statistics as part of the data loading operation means no additional full data scan is required to have statistics available immediately after the data is loaded.

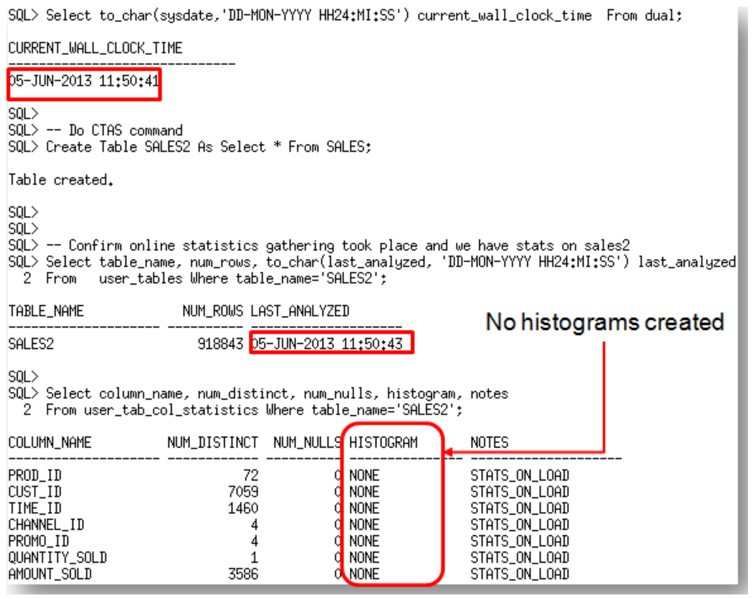
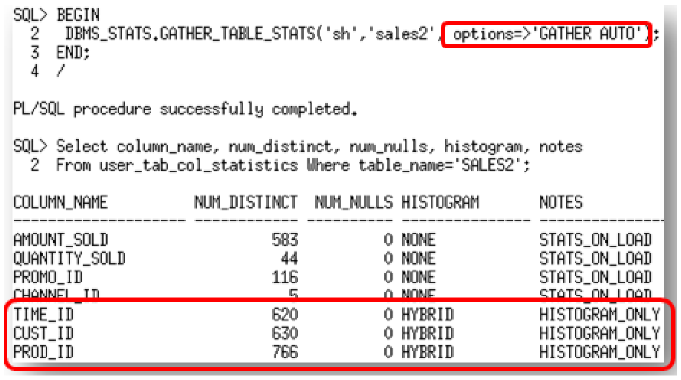

```
SQL> Select to_char(sysdate,'DD-MON-YYYY HH24:MI:SS') current_wall_clock_time  From dual;

CURRENT_WALL_CLOCK_TIME
-----------------------------
05-JUN-2013 11:50:41

SQL>
SQL> -- Do CTAS command
SQL> Create Table SALES2 As Select * From SALES;

Table created.

SQL>
SQL>
SQL> -- Confirm online statistics gathering took place and we have stats on sales2
SQL> Select table_name, num_rows, to_char(last_analyzed, 'DD-MON-YYYY HH24:MI:SS') last_analyzed
  2  From   user_tables Where table_name='SALES2';

TABLE_NAME             NUM_ROWS LAST_ANALYZED
-------------------- ---------- --------------------
SALES2                   918843 05-JUN-2013 11:50:43

SQL>
SQL> Select column_name, num_distinct, num_nulls, histogram, notes
  2  From user_tab_col_statistics Where table_name='SALES2';

COLUMN_NAME          NUM_DISTINCT  NUM_NULLS HISTOGRAM       NOTES
-------------------- ------------ ---------- --------------- --------------------
PROD_ID                        72          0 NONE            STATS_ON_LOAD
CUST_ID                      7059          0 NONE            STATS_ON_LOAD
TIME_ID                      1460          0 NONE            STATS_ON_LOAD
CHANNEL_ID                      4          0 NONE            STATS_ON_LOAD
PROMO_ID                        4          0 NONE            STATS_ON_LOAD
QUANTITY_SOLD                   1          0 NONE            STATS_ON_LOAD
AMOUNT_SOLD                  3586          0 NONE            STATS_ON_LOAD
```

Figure 4: Online statistic gathering provides both table and column statistics for newly created SALES2 table

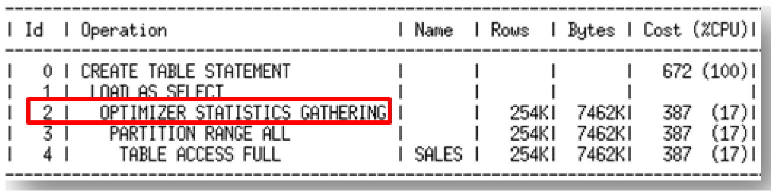
Online statistics gathering does not gather histograms or index statistics, as these types of statistics require additional data scans, which could have a large impact on the performance of the data load. To gather the necessary histogram and index statistics without re-gathering the base column statistics use the `DBMS_STATS.GATHER_TABLE_STATS` procedure with the new options parameter set to `GATHER AUTO`. Note that for performance reasons, `GATHER AUTO` builds histogram using a *sample* of rows rather than *all* rows in the table.



```
SQL> BEGIN
  2   DBMS_STATS.GATHER_TABLE_STATS('sh','sales2', options=>'GATHER AUTO');
  3  END;
  4  /

PL/SQL procedure successfully completed.

SQL> Select column_name, num_distinct, num_nulls, histogram, notes
  2  From user_tab_col_statistics Where table_name='SALES2';

COLUMN_NAME          NUM_DISTINCT  NUM_NULLS HISTOGRAM       NOTES
-------------------- ------------ ---------- --------------- --------------
AMOUNT_SOLD                   583          0 NONE            STATS_ON_LOAD
QUANTITY_SOLD                  44          0 NONE            STATS_ON_LOAD
PROMO_ID                      116          0 NONE            STATS_ON_LOAD
CHANNEL_ID                      5          0 NONE            STATS_ON_LOAD
TIME_ID                       620          0 HYBRID          HISTOGRAM_ONLY
CUST_ID                       630          0 HYBRID          HISTOGRAM_ONLY
PROD_ID                       766          0 HYBRID          HISTOGRAM_ONLY
```

Figure 5: Set options to GATHER AUTO creates histograms on SALES2 table without regarding the base statistics

The notes column "HISTOGRAM_ONLY" indicates that histograms were gathered without re-gathering basic column statistics. There are two ways to confirm online statistics gathering has occurred: check the execution plan to see if the new row source `OPTIMIZER STATISTICS GATHERING` appears in the plan or look in the new notes column of the `USER_TAB_COL_STATISTICS` table for the status `STATS_ON_LOAD`.



```
----------------------------------------------------------------------------------
| Id  | Operation                        | Name  | Rows  | Bytes | Cost (%CPU)|
----------------------------------------------------------------------------------
|   0 | CREATE TABLE STATEMENT           |       |       |       |   672 (100)|
|   1 |  LOAD AS SELECT                  |       |       |       |            |
|   2 |   OPTIMIZER STATISTICS GATHERING |       |  254K|  7462K|   387   (17)|
|   3 |    PARTITION RANGE ALL           |       |  254K|  7462K|   387   (17)|
|   4 |     TABLE ACCESS FULL            | SALES |  254K|  7462K|   387   (17)|
----------------------------------------------------------------------------------
```

Figure 6: Execution plan for an on-line statistics gathering operation

Since online statistics gathering was designed to have a minimal impact on the performance of a direct path load operation it can only occur when data is being loaded into an empty object. To ensure online statistics gathering kicks in when loading into a new partition of an existing table, use extended syntax to specify the partition explicitly. In this case partition level statistics will be created but global level (table level) statistics will not be updated. If incremental statistics have been enabled on the partitioned table a synopsis will be created as part of the data load operation.

Online statistics gathering can be disabled for individual SQL statements using the `NO_GATHER_OPTIMIZER_STATISTICS` hint.

Incremental Statistics and Partition Exchange Data Loading

Gathering statistics on partitioned tables consists of gathering statistics at both the table level (global statistics) and (sub)partition level. If the `INCREMENTAL`[3] 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

3 More information can be found in part one of this white paper series, *Understanding Optimizer Statistics With Oracle Database 12c Release 2*.

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

If you are using partition exchange loads and wish to take advantage of incremental statistics, you will need to set the `DBMS_STATS` table preference `INCREMENTAL_LEVEL` on the non-partitioned table to identify that it will be used in partition exchange load. By setting the `INCREMENTAL_LEVEL` to `TABLE` (default is `PARTITION`), Oracle will automatically create a synopsis for the table when statistics are gathered on it. This table level synopsis will then become the partition level synopsis after the exchange.

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 automated statistics gathering task. 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 manual flush the information using the `DBMS_STATS.FLUSH_DATABASE_MONITORING_INFO` function.

Preventing "Out of Range" Condition

Regardless of whether you use the automated statistics gathering task 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`[4] 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] 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

4 More information can be found in part one of this white paper series, *Understanding Optimizer Statistics With Oracle Database 12c Release 2*.

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.

Assuring the Quality of Optimizer Statistics

Good quality statistics are essential to be able to generate optimal SQL execution plans, but sometimes statistics can be of poor quality and this fact could remain unnoticed. For example, older "inherited" systems might use scripts that are no longer understood by the database administrators and, understandably, there is a reluctance to change them. However, because Oracle continuously enhances statistics gathering features it is possible that best practice recommendations will be neglected.

For these reasons, Oracle Database 12c Release 2 includes a new advisor called the *Optimizer Statistics Advisor* to help you to improve the quality of statistics in the database. This diagnostic software analyzes information in the data dictionary, assesses the quality of statistics and discovers how statistics are being gathered. It will report on poor and missing statistics and generate recommendations to resolve these problems.

The principle behind its operation is to apply best-practice *Rules* to uncover potential problems. These problems are reported as a series of *Findings*, which in turn can lead to specific *Recommendations*. Recommendations can be implemented automatically using *Actions* (either immediately or via an auto-generated script to be executed by the database administrator).

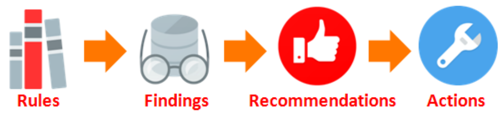


Figure 7: The Optimizer Statistics Advisor

The advisor task runs automatically in the maintenance window, but it can also be run on demand. The HTML or text report generated by the advisor can be viewed at any time and the actions can be implemented at any time.

Figure 8 illustrates an example of a specific rule that leads to a finding, a recommendation and then an action to resolve the problem:

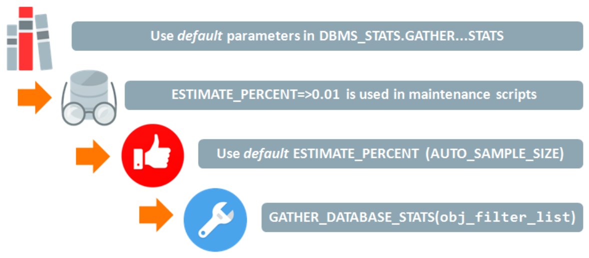

Use *default* parameters in DBMS_STATS.GATHER...STATS
ESTIMATE_PERCENT=>0.01 is used in maintenance scripts
Use *default* ESTIMATE_PERCENT (AUTO_SAMPLE_SIZE)
GATHER_DATABASE_STATS(obj_filter_list)

Figure 8: Example of a rule, finding, recommendation and action.

The advisor task gathers and stores data in the data dictionary. It is a low performance overhead operation because it performs an analysis of optimizer statistics and statistics gathering information (that's already held in the data dictionary). It does not perform a secondary analysis of data stored in application schema objects.

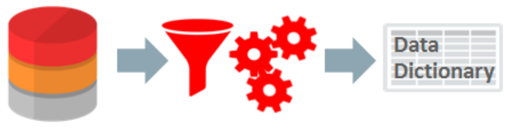


Figure 9: Reading the data dictionary, executing the task via a filter and storing the results.

Once the task is complete, the report can be generated in HTML or text format and an action (SQL) script can be created too.

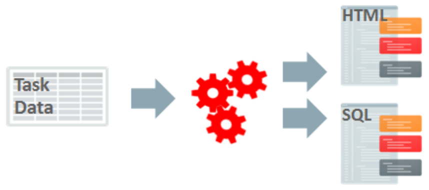


Figure 10: Reporting on the advisor task and generating the action SQL script.

It is a simple matter to view the report generated by the automated task:

```
select dbms_stats.report_advisor_task('auto_stats_advisor_task') as report from dual;
```

Alternatively, users with the ADVISOR privilege can execute the task manually and report on the results using the following three-step process:

```
DECLARE
   tname   VARCHAR2(32767) := 'demo';   -- task name
BEGIN
   tname := dbms_stats.create_advisor_task(tname);
END;
/
DECLARE
   tname   VARCHAR2(32767) := 'demo';   -- task name
   ename   VARCHAR2(32767) := NULL;     -- execute name
BEGIN
   ename := dbms_stats.execute_advisor_task(tname);
END;
/
SELECT dbms_stats.report_advisor_task('demo') AS report
FROM dual;
```

The actions generated by the advisor can be implemented immediately:

```
DECLARE
    tname            VARCHAR2 (32767) := 'demo'; -- task name
    impl_result      CLOB;                       -- report of
                                                    implementation
BEGIN
    impl_result := dbms_stats.implement_advisor_task(tname);
END;
/
```

In addition, Oracle Database 12c Real Application Testing includes useful performance assurance features such as SQL Performance Advisor Quick Check. See the Oracle white paper, *Database 12c Real Application Testing Overview* for more details (see Reference 3 on Page 21).

Gathering Statistics More Quickly

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

» Using the DEGREE parameter

» Concurrent statistics gathering

» A combination of both DEGREE and concurrent gathering

Using the DEGREE Parameter

The DBMS_STATS, `DEGREE` parameter controls the number of parallel execution 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.

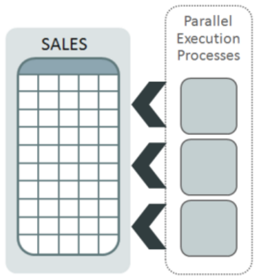


Figure 11: Using parallelism via the DEGREE parameter

You should note that setting the `DEGREE` for a partitioned table means that multiple parallel sever 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.

Concurrent Statistic Gathering

Concurrent statistics gathering enables statistics to be gathered on multiple tables in a schema (or database), and multiple (sub)partitions within a table 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.

Concurrent statistics gathering is controlled by the global preference, `CONCURRENT`, which can be set to `MANUAL`, `AUTOMATIC`, `ALL`, `OFF`. By default it is set to `OFF`. When `CONCURRENT` is enabled, Oracle employs Oracle Job Scheduler and Advanced Queuing components to create and manage multiple statistics gathering jobs concurrently.

Calling `DBMS_STATS.GATHER_TABLE_STATS` on a partitioned table when `CONCURRENT` is set to `MANUAL or ALL`, causes Oracle to create a separate statistics gathering job for each (sub)partition in the table. How many of these jobs will execute concurrently, and how many will be queued is based on the number of available job queue processes (`JOB_QUEUE_PROCESSES` initialization parameter, per node on a RAC environment) and the available system resources. As the currently running jobs complete, more jobs will be dequeued and executed until all of the (sub)partitions have had their statistics gathered.

If you gather statistics using `DBMS_STATS.GATHER_DATABASE_STATS`, `DBMS_STATS.GATHER_SCHEMA_STATS`, or `DBMS_STATS.GATHER_DICTIONARY_STATS`, then Oracle will create a separate statistics gathering job for each non-partitioned table, and each (sub)partition for the partitioned tables. Each partitioned table will also have a coordinator job that manages its (sub)partition jobs. The database will then run as many concurrent jobs as possible, and queue the remaining jobs until the executing jobs complete. However, to prevent possible deadlock scenarios multiple partitioned tables cannot be processed simultaneously. Hence, if there are some jobs running for a partitioned table, other partitioned tables in a schema (or database or dictionary) will be queued until the current one completes. There is no such restriction for non-partitioned tables.

Figure 12 illustrates the creation of jobs at different levels, when a `DBMS_STATS.GATHER_SCHEMA_STATS` command has been issued on the `SH` schema. Oracle will create a statistics gathering job (Level 1 in Figure 12) for each of the non-partitioned tables;

» CHANNELS

» COUNTRIES

» TIMES

Oracle will create a coordinator job for each partitioned table: `SALES` and `COSTS`, which in turn creates a statistics gathering job for each of partition in `SALES` and `COSTS` tables, respectively (Level 2 in Figure 12).

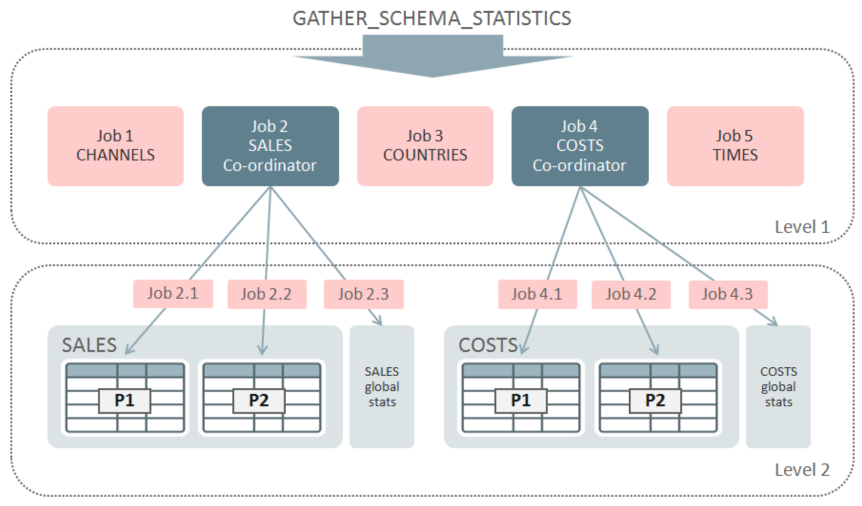


Figure 12: List of the statistics gathering job created when Concurrent Statistics Gathering occurs on the SH schema

Each of the individual statistics gathering jobs can also take advantage of parallel execution if the DEGREE parameter is specified.

If a table, partition, or sub-partition is very small or empty, the database may automatically batch the object with other small objects into a single job to reduce the overhead of job maintenance.

Configuring Concurrent Statistics Gathering

The concurrency setting for statistics gathering is turned off by default. It can be turned on as follows:

```
exec dbms_stats.set_global_prefs('concurrent', 'all')
```

You will also need some additional privileges above and beyond the regular privileges required to gather statistics. The user must have the following Job Scheduler and AQ privileges:

» CREATE JOB

» MANAGE SCHEDULER

» MANAGE ANY QUEUE

The `SYSAUX` tablespace should be online, as the Job Scheduler stores its internal tables and views in `SYSAUX` tablespace. Finally, the `JOB_QUEUE_PROCESSES` parameter should be set to fully utilize all of the system resources available (or allocated) for the statistics gathering process. If you don't plan to use parallel execution you should set the `JOB_QUEUE_PROCESSES` to 2 X total number of CPU cores (this is a per node parameter in a RAC environment). Please make sure that you set this parameter system-wise (`ALTER SYSTEM` ... or in init.ora file) rather than at the session level (`ALTER SESSION`).

If you are going to use parallel execution as part of concurrent statistics gathering you should disable the parallel adaptive multi user:

```
ALTER SYSTEM SET parallel_adaptive_multi_user=false;
```

Resource manager must be activated using, for example:

```
ALTER SYSTEM SET resource_manager_plan = 'DEFAULT_PLAN';
```

It is also recommended that you enable parallel statement queuing. This requires resource manager to be activated, and the creation of a temporary resource plan where the consumer group "`OTHER_GROUPS`" should have queuing enabled. By default, Resource Manager is activated only during the maintenance windows. The following script illustrates one way of creating a temporary resource plan (pqq_test), and enabling the Resource Manager with this plan.

```
-- connect as a user with dba privileges
begin
  dbms_resource_manager.create_pending_area();
  dbms_resource_manager.create_plan('pqq_test', 'pqq_test');
  dbms_resource_manager.create_plan_directive(
        'pqq_test',
        'OTHER_GROUPS',
        'OTHER_GROUPS directive for pqq',
        parallel_target_percentage => 90);
  dbms_resource_manager.submit_pending_area();
end;
/
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'pqq_test' SID='*';
```

If you want the automated statistics gathering task to take advantage of concurrency, set `CONCURRENT` to either `AUTOMATIC` or `ALL`. A new `ORA$AUTOTASK` consumer group has been added to the Resource Manager plan used during the maintenance window, to ensure concurrent statistics gathering does not use too much of the system resources.

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.

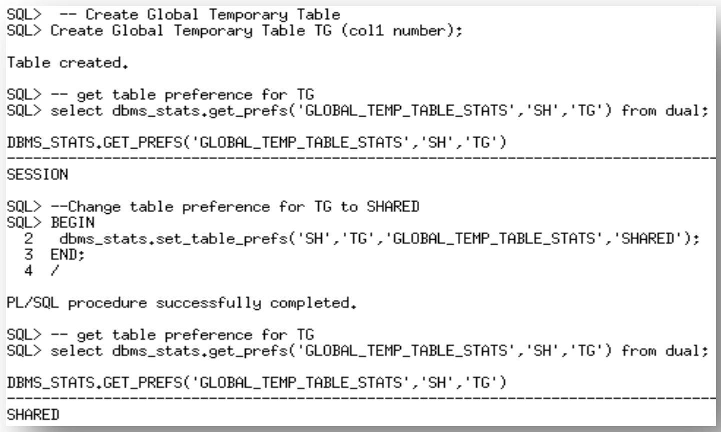
If you relied on the automatic statistics gather job to maintain statistics on such tables then the statistics would always show the table was empty because it was empty over night when the job ran. However, during the course of the day the table may have hundreds of thousands of rows in it.

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 automated statistics gathering task 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) but in previous releases its wasn't always a good idea as all sessions using the GTT had to share a single set of statistics so a lot of systems relied on dynamic statistics.

However, in Oracle Database 12*c*, it is now possible to have a separate set of statistics for every session using a GTT. Statistics sharing on a GTT is controlled using a new `DBMS_STATS` preference `GLOBAL_TEMP_TABLE_STATS`. By default the preference is set to `SESSION`, meaning each session accessing the GTT will have its own set of statistics. The optimizer will try to use session statistics first but if session statistics do not exist, then optimizer will use shared statistics.

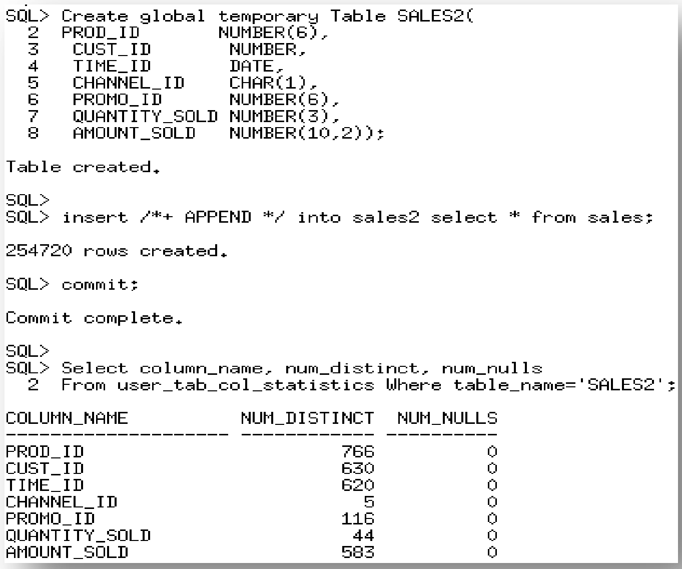


```
SQL>  -- Create Global Temporary Table
SQL> Create Global Temporary Table TG (col1 number);

Table created.

SQL> -- get table preference for TG
SQL> select dbms_stats.get_prefs('GLOBAL_TEMP_TABLE_STATS','SH','TG') from dual;

DBMS_STATS.GET_PREFS('GLOBAL_TEMP_TABLE_STATS','SH','TG')
--------------------------------------------------------------------------------
SESSION

SQL> --Change table preference for TG to SHARED
SQL> BEGIN
  2   dbms_stats.set_table_prefs('SH','TG','GLOBAL_TEMP_TABLE_STATS','SHARED');
  3  END;
  4  /

PL/SQL procedure successfully completed.

SQL> -- get table preference for TG
SQL> select dbms_stats.get_prefs('GLOBAL_TEMP_TABLE_STATS','SH','TG') from dual;

DBMS_STATS.GET_PREFS('GLOBAL_TEMP_TABLE_STATS','SH','TG')
--------------------------------------------------------------------------------
SHARED
```

Figure 13: Changing the default behavior of not sharing statistics on a GTT to forcing statistics sharing

If you have upgraded from Oracle Database 11g and if database applications have not been modified to take advantage of `SESSION` statistics for GTTs, you may want to keep GTT behavior consistent with the pre-upgrade environment by setting the `DBMS_STATS` preference `GLOBAL_TEMP_TABLE_STATS` to `SHARED` (at least until applications have been updated).

When populating a GTT (that persists rows on commit) using a direct path operation, session level statistics will be automatically created due to online statistics gathering, which will remove the necessity to run additional statistics gathering command and will not impact the statistics used by any other session.



```
SQL> Create global temporary Table SALES2(
  2  PROD_ID       NUMBER(6),
  3   CUST_ID       NUMBER,
  4   TIME_ID       DATE,
  5   CHANNEL_ID    CHAR(1),
  6   PROMO_ID      NUMBER(6),
  7   QUANTITY_SOLD NUMBER(3),
  8   AMOUNT_SOLD   NUMBER(10,2));

Table created.

SQL>
SQL> insert /*+ APPEND */ into sales2 select * from sales;

254720 rows created.

SQL> commit;

Commit complete.

SQL>
SQL> Select column_name, num_distinct, num_nulls
  2  From user_tab_col_statistics Where table_name='SALES2';

COLUMN_NAME          NUM_DISTINCT  NUM_NULLS
-------------------- ------------ ----------
PROD_ID                       766          0
CUST_ID                       630          0
TIME_ID                       620          0
CHANNEL_ID                      5          0
PROMO_ID                      116          0
QUANTITY_SOLD                  44          0
AMOUNT_SOLD                   583          0
```

Figure 14: Populating a GTT using a direct path operation results in session level statistics being automatically gathered

Intermediate Work Tables

Intermediate work tables are typically seen as part of an ELT 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 automated statistics gathering task 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 automated statistics gathering task run during the nightly maintenance window. It is highly recommended that you allow the automated statistics gather task 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`.

```
exec dbms_stats.set_global_prefs('autostats_target','oracle')
```

Fixed Object Statistics

Beginning with Oracle Database 12c, fixed object statistics are collected by the automated statistics gathering task if they have not been previously collected. Beyond that, the database does not gather fixed object statistics. Unlike other database tables, dynamic sampling is not automatically used for SQL statements involving X$ tables when optimizer statistics are missing so 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 import 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 Oracle 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 automated statistics gathering task 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.

References

1. Oracle white paper: *Understanding Optimizer Statistics with Oracle Database 12c Release 2*

2. Oracle white paper: *Optimizer with Oracle Database 12c Release 2*

3. Oracle white paper: *Database 12c Real Application Testing Overview*

ORACLE

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle

facebook.com/oracle

twitter.com/oracle

oracle.com

Hardware and Software, Engineered to Work Together