22

Managing Performance: SQL Tuning

ORACLE

Objectives

After completing this lesson, you should be able to:

- Manage optimizer statistics
- Use the SQL Tuning Advisor to:
  - Identify SQL statements that are using the most resources
  - Tune SQL statements that are using the most resources
- Use the SQL Access Advisor to tune a workload

SQL Tuning

SQL tuning process

- Identify poorly tuned SQL statements.
- Tune the individual statements.
- Tune the application as a whole.

Generally, the tuning effort that yields the most benefit is SQL tuning. Poorly tuned SQL uses more resources than required. This inefficiency prevents scalability, uses more OS and database resources, and increases response time. To tune poorly tuned SQL statements, they must be identified, and then tuned. SQL statements can be tuned individually, but often the solution that optimizes one statement can hurt the performance of several others.

The SQL statements that use the most resources are by definition the statements in need of tuning. These are statements that have the longest elapsed time, use the most CPU, or do the most physical or logical reads.

Tune the individual statements by checking the optimizer statistics, check the explain plan for the most efficient access path, test alternate SQL constructions, and test possible new indexes, materialized views, and partitioning.

Test the application as a whole, using the tuned SQL statements. Is the overall performance better?

The methodology is sound, but tedious. Tuning an individual statement is not difficult. Testing the overall impact of the individual statement tuning on an application can be very difficult.

In Oracle Database, a set of SQL advisors are available to identify and tune statements, individually, or as a set.

Oracle Optimizer: Overview

The Oracle optimizer determines the most efficient execution plan and is the most important step in the processing of any SQL statement.

The optimizer:

- Evaluates expressions and conditions
- Uses object and system statistics
- Decides how to access the data
- Decides how to join tables
- Determines the most efficient path

The optimizer is the part of the Oracle Database server that creates the execution plan for a SQL statement. The determination of the execution plan is an important step in the processing of any SQL statement and can greatly affect execution time.

The execution plan is a series of operations that are performed in sequence to execute the statement. The optimizer considers many factors related to the referenced objects and the conditions specified in the query. The information necessary to the optimizer includes:

- Statistics gathered for the system (I/O, CPU, and so on) as well as schema objects (number of rows, index, and so on)
- Information in the dictionary
- `WHERE` clause qualifiers
- Hints supplied by the developer

When you use diagnostic tools, such as Enterprise Manager, `EXPLAIN PLAN`, and SQL*Plus `AUTOTRACE`, you can see the execution plan that the optimizer chooses.

Note: The Oracle optimizer has two names based on its functionality: the *query optimizer* and the *Automatic Tuning Optimizer*.

Optimizer Statistics

Optimizer statistics are:

- A snapshot at a point in time
- Persistent across instance restarts
- Collected automatically

```
SQL> SELECT COUNT(*) FROM hr.employees;
  COUNT(*)
----------
       214
SQL> SELECT num_rows FROM dba_tables
  2 WHERE owner='HR' AND table_name = 'EMPLOYEES';
  NUM_ROWS
----------
       107
```


Optimizer statistics include table, column, index, and system statistics. Statistics for tables and indexes are stored in the data dictionary. These statistics are not intended to provide real-time data. They provide the optimizer a *statistically* correct snapshot of data storage and distribution, which the optimizer uses to make decisions on how to access data.

The statistics that are collected include:

- Size of the table or index in database blocks
- Number of rows
- Average row size and chain count (tables only)
- Height and number of deleted leaf rows (indexes only)

As data is inserted, deleted, and modified, these facts change. Because the performance impact of maintaining real-time data distribution statistics is prohibitive, these statistics are updated by periodically gathering statistics on tables and indexes.

Optimizer statistics are collected automatically by an automatic maintenance job that runs during predefined maintenance windows once daily by default. System statistics are operating system characteristics that are used by the optimizer. These statistics are not collected automatically. For details about collecting system statistics, see the *Oracle Database Performance Tuning Guide*.

Optimizer statistics are not the same as the database performance statistics that are gathered in the AWR snapshot.

Optimizer Statistics Collection

- SQL performance tuning: Depends on collection of accurate statistics
- Optimizer statistics:
  - Object statistics
  - Operating system statistics
- Ways to collect statistics:
  - Automatically: Automatic Maintenance Tasks
  - Manually: `DBMS_STATS` package
  - By setting database initialization parameters
  - By importing statistics from another database

Optimizer statistics are collections of data that are specific details about database objects. These statistics are essential for the query optimizer to choose the best execution plan for each SQL statement. These statistics are gathered periodically and do not change between gatherings.

The recommended approach to gathering optimizer statistics is to allow the Oracle Database server to automatically gather the statistics. The Automatic Maintenance Tasks can be created automatically at database creation time and is managed by the Scheduler. It gathers statistics on all objects in the database that have either missing or stale optimizer statistics by default. You can change the default configuration through the Automatic Maintenance Tasks page.

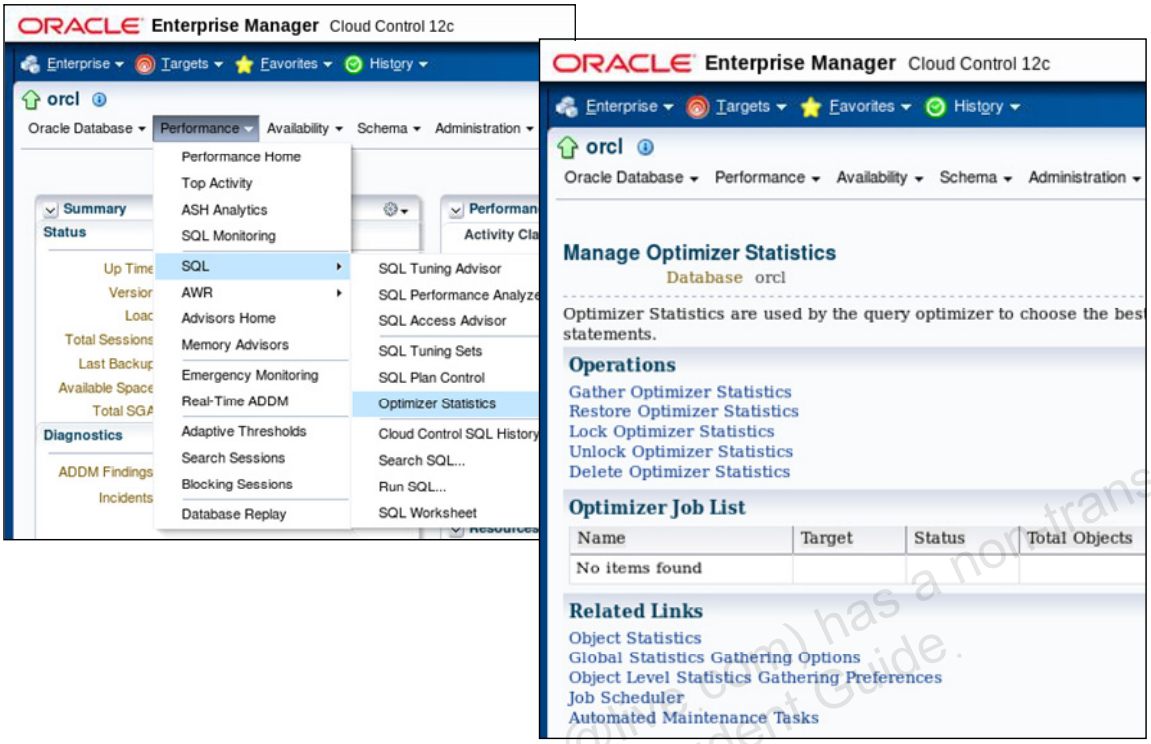
System statistics describe the system's hardware characteristics, such as I/O and CPU performance and utilization, to the query optimizer. When choosing an execution plan, the optimizer estimates the I/O and CPU resources required for each query. System statistics enable the query optimizer to more accurately estimate I/O and CPU costs, and thereby choose a better execution plan. System statistics are collected using the `DBMS_STATS.GATHER_SYSTEM_STATS` procedure. When the Oracle Database server gathers system statistics, it analyzes system activity in a specified period of time. System statistics are not automatically gathered. Oracle Corporation recommends that you use the `DBMS_STATS` package to gather system statistics.

If you choose not to use automatic statistics gathering, you must manually collect statistics in all schemas, including system schemas. If the data in your database changes regularly, you also need to gather statistics regularly to ensure that the statistics accurately represent characteristics of your database objects. To manually collect statistics, use the `DBMS_STATS` package. This PL/SQL package is also used to modify, view, export, import, and delete statistics.

You can also manage optimizer and system statistics collection through database initialization parameters. For example:

- The `OPTIMIZER_DYNAMIC_SAMPLING` parameter controls the level of dynamic sampling performed by the optimizer. You can use dynamic sampling to estimate statistics for tables and relevant indexes when they are not available or are too out of date to trust. Dynamic sampling also estimates single-table predicate selectivity when collected statistics cannot be used or are likely to lead to significant errors in estimation.
- The `STATISTICS_LEVEL` parameter controls all major statistics collections or advisories in the database and sets the statistics collection level for the database. The values for this parameter are `BASIC`, `TYPICAL`, and `ALL`. You can query the `V$STATISTICS_LEVEL` view to determine which parameters are affected by the `STATISTICAL_LEVEL` parameter.
  Note: Setting `STATISTICS_LEVEL` to `BASIC` disables many automatic features and is not recommended.

Using the Manage Optimizer Statistics Page



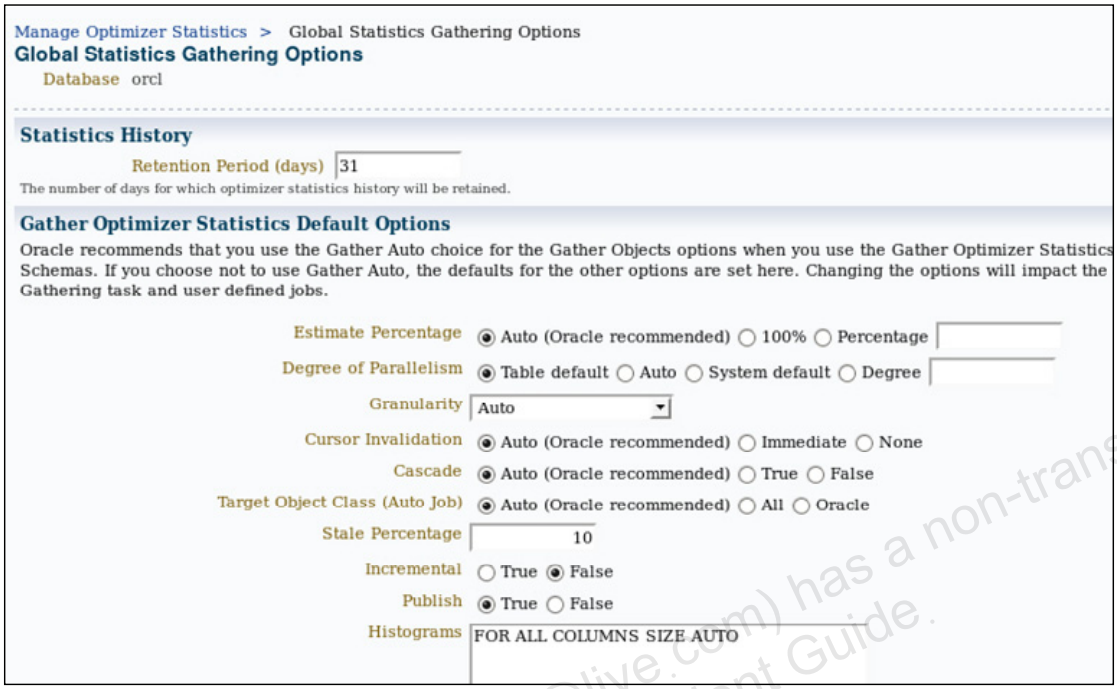
To manage optimizer statistics in Enterprise Manager, select Optimizer Statistics in the SQL submenu of the Performance menu. On the Manage Optimizer Statistics page, you can perform the following tasks:

- Gather optimizer statistics manually.
- Restore optimizer statistics to a point in the past. The chosen point in time must be within the optimizer statistics retention period, which defaults to 30 days.
- Lock optimizer statistics to guarantee that the statistics for certain objects are never overwritten. This is useful if statistics have been calculated for a certain table at a time when well-representative data is present, and you want to always have those statistics. No fluctuations in the table affect the statistics if they are locked.
- Unlock optimizer statistics to undo the previously done lock.
- Delete optimizer statistics to delete statistics.

Best-Practice Tip

Use the automatic maintenance tasks to gather optimizer statistics. To enable the task for gathering optimizer statistics gathering, you must ensure that the `STATISTICS_LEVEL` initialization parameter is set to `TYPICAL` or `ALL`.

Setting Global Preferences by Using Enterprise Manager Cloud Control

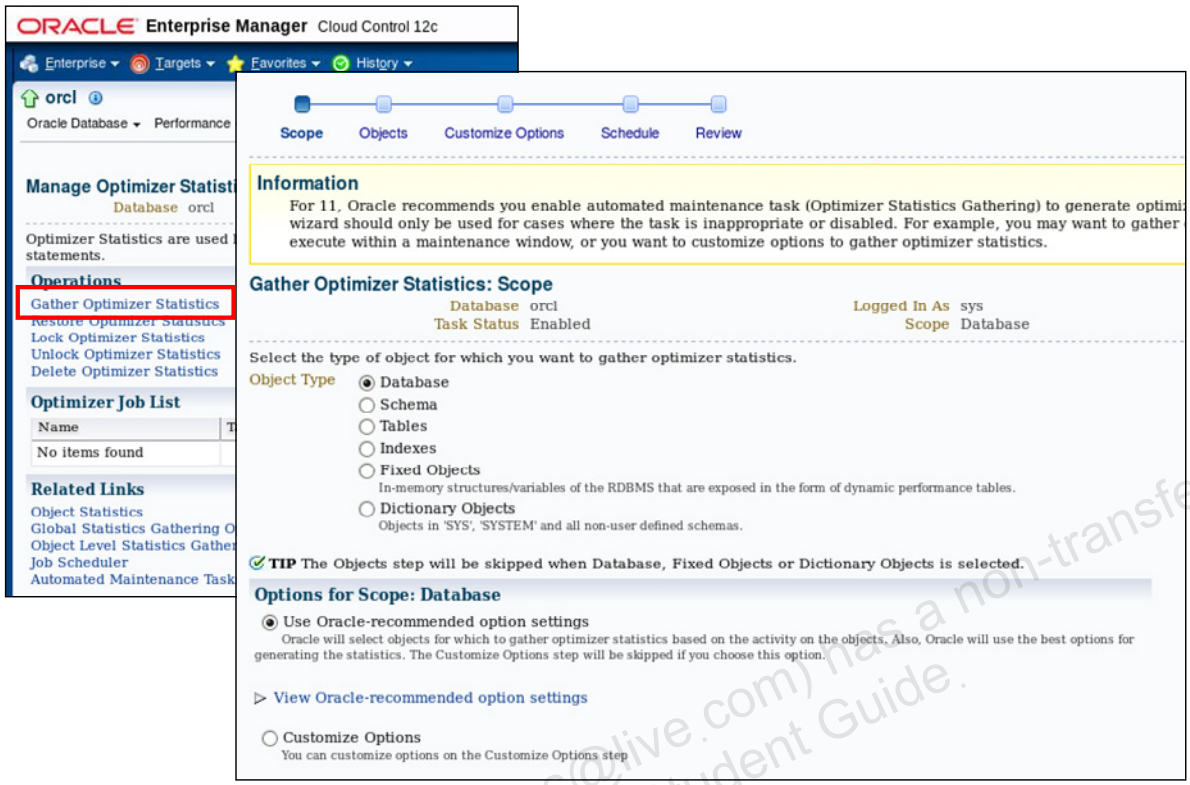


You can manage global preference settings by using Enterprise Manager. Click the Global Statistics Gathering Options link on the Manage Optimizer Statistics page.

On the Global Statistics Gathering Options page, change the global preferences in the Gather Optimizer Statistics Default Options section. When finished, click the Apply button.

Note: To change the statistics gathering options at the object level or schema level, click the Object Level Statistics Gathering Preferences link on the Manage Optimizer Statistics page.

Gathering Optimizer Statistics Manually



You may need to gather statistics manually at certain times, such as when the contents of a table have changed so much between automatic gathering jobs that the statistics no longer represent the table accurately. This is common for large tables that experience more than a 10 percent change in size in a 24-hour period.

Best-practice tip: Collect statistics often enough that the table never changes by more than about 10 percent between collection periods. This may require manual statistics collection or additional maintenance windows.

Statistics can be manually collected by using either Enterprise Manager or the `DBMS_STATS` package. System statistics can be gathered only by using the `DBMS_STATS` package. System statistics describe the system's hardware characteristics, such as I/O and CPU performance and utilization, to the query optimizer.

The Gather Optimizer Statistics menu selection starts a wizard that enables you to select the scope, objects, options, and schedule for a job that will gather optimizer statistics. The wizard submits a `DBMS_STATS.GATHER_*_STATS` job at the scope you specify: table, schema, or database. In this wizard, you set the preferences for the default values used by the `DBMS_STATS` package and you schedule this job to run at a time that you determine.

Gathering statistics manually for routine statistics collection is not recommended because the statistics are gathered more efficiently and with less impact on users during the maintenance windows. A manual job can also be submitted if the automatic job has failed or been disabled.

You can also gather optimizer statistics with the DBMS_STATS package directly:

```
SQL> EXEC dbms_stats.gather_table_stats('HR','EMPLOYEES');
SQL> SELECT num_rows FROM dba_tables
  2  WHERE owner='HR' AND table_name = 'EMPLOYEES';
  NUM_ROWS
----------
       214
```

Notice that the number of rows now correctly reflects what was in the table at the time that the statistics were gathered. DBMS_STATS also enables manual collection of statistics for an entire schema or even for the whole database.

System statistics do not change unless the workload significantly changes. As a result, system statistics do not need frequent adjustment. The DBMS_STATS.GATHER_SYSTEM_STATS procedure will collect system statistics over a specified period, or you can start the gathering of system statistics and make another call to stop gathering.

Best-practice tip: Use the following command when you create a database:

```
SQL> EXEC dbms_stats.gather_system_stats('NOWORKLOAD');
```

The NOWORKLOAD option takes a few minutes (depending on the size of the database) and captures estimates of I/O characteristics such as average read seek time and I/O transfer rate.

Setting Optimizer Statistics Preferences

- Optimizer statistics preferences set the default values of parameters used by:
  - Automatic statistics collection
  - DBMS_STATS.GATHER_*_STATS procedures
- Set preferences at levels:
  - Table, schema, database, or global
- Preferences: CASCADE, DEGREE, ESTIMATE_PERCENT, NO_INVALIDATE, METHOD_OPT, GRANULARITY, INCREMENTAL, PUBLISH, STALE_PERCENT
- Use DBMS_STATS.SET | GET | DELETE | EXPORT | IMPORT_*_PREFS to manage preferences

Optimizer statistics gathering task


```
EXEC DBMS_STATS.SET_TABLE_PREFS('SH','SALES','STALE_PERCENT','13');
```


The DBMS_STATS.GATHER_*_STATS procedures can be called at various levels to gather statistics for an entire database or for individual objects, such as tables. When the GATHER_*_STATS procedures are called, several of the parameters are often allowed to default. The supplied defaults work well for most of the objects in the database, but for some objects or schemas the defaults need to be changed. Instead of running manual jobs for each of these objects, Oracle Database enables you to set values (called preferences) for individual objects, schemas, or databases, or to change the default values with the global-level command.

The preferences specify the parameters that are given to the gather procedures. The SET_*_PREFS procedures create preference values for any object that is not owned by SYS or SYSTEM. The expected use is that the DBA will set the global preferences for any parameters that should be database-wide. These will be applied for any parameter that is allowed to default.

The SET_DATATBASE_PREFS procedure iterates over all the tables and schemas in the database setting the specified preference. SET_SCHEMA_PREFS iterates over the tables in the specified schema. SET_TABLE_PREFS sets the preference value for a single table.

All object preferences—whether set at the database, schema, or table level—are held in a single table. Changing the preferences at the schema level overwrites the preferences that were previously set at the table level.

When the various gather procedures execute, they retrieve the object-level preferences that were set for each object. You can view the object-level preferences in the DBA_TAB_STAT_PREFS view. Any preferences that are not set at the object level will be set to the global-level preferences. You can see the global preferences by calling the DBMS_STATS.GET_PREFS procedure for each preference.

You can set, get, delete, export, and import those preferences at the table, schema, database, and global levels. The preference values are expected to be set from global to table levels, applying the preferences to the smallest group last.

Preferences include:

- CASCADE – Determines whether index statistics are collected as part of gathering table statistics
- DEGREE – Sets the degree of parallelism that is used for gathering statistics
- PUBLISH – Is used to decide whether to publish the statistics to the dictionary or store them in a private area. This enables the DBA to validate the statistics before publishing them to the data dictionary with the PUBLISH_PENDING_STATS procedure.
- STALE_PERCENT – Is used to determine the threshold level at which an object is considered to have stale statistics. The value is a percentage of the rows modified since the last statistics gathering. The example changes the 10 percent default to 13 percent for SH.SALES only.
- INCREMENTAL – Is used to gather global statistics on partitioned tables in an incremental way
- METHOD_OPT – Determines the columns and histogram parameters that are used to gather column statistics
- GRANULARITY – Determines the granularity of statistics to collect (which is pertinent only if the table is partitioned)
- NO_INVALIDATE – Is used to determine whether to invalidate cursors
- ESTIMATE_PERCENT – Is used to determine the number of rows to sample to obtain good statistics. It is a percentage of the number of rows in the table

Note: For details about these preferences, see the DBMS_STATS documentation in the *Oracle Database PL/SQL Packages and Types Reference*.

Preferences may be deleted with the DBMS_STATS.DELETE_*_PREFS procedures at the table, schema, and database levels. You can reset the global preferences to the recommended values with the DBMS_STATS.RESET_PARAM_DEFAULTS procedure.

Concurrent Statistics Gathering

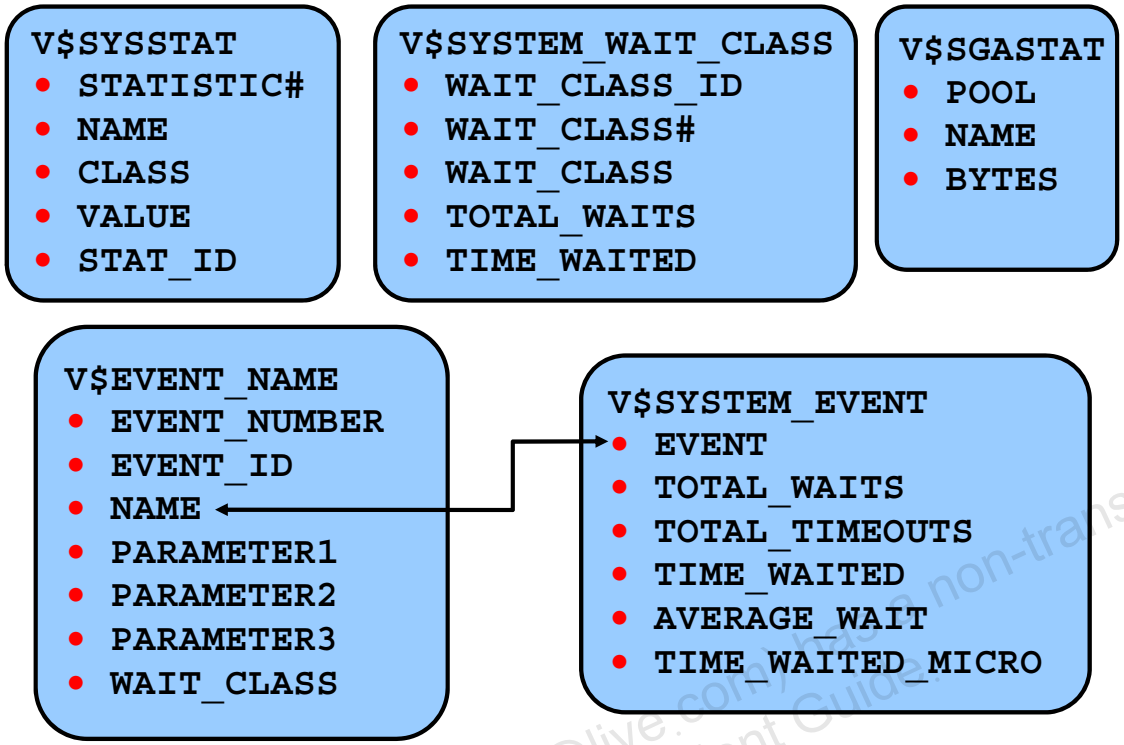
- The auto statistics gather job uses concurrency.
- Prevent concurrent statistics gathering to make the system overwhelmed through careful resource usage capping.
- Allow gathering index statistics concurrently.
- Allow gathering of statistics for multiple partitioned tables concurrently.

Before Oracle Database 12*c*, the automatic statistics gather job gathered statistics one segment at a time. It now uses concurrency.

In Oracle Database 12*c*, you can run multiple statistics gathering tasks concurrently which may provide considerable improvements in statistics gathering time by increasing the resource utilization rate, in particular, on multi-CPU systems.

Employ smart scheduling mechanisms to have maximum degree of concurrency (to the extent that resources are available), for example, allow gathering of statistics for multiple partitioned tables concurrently.

Viewing Statistics Information



To effectively diagnose performance problems, statistics must be available. The Oracle Database instance generates many types of cumulative statistics for the system, sessions, and individual SQL statements at the instance level. The Oracle Database server also tracks cumulative statistics on segments and services. When analyzing a performance problem in any of these scopes, you typically look at the change in statistics (delta value) over the period of time you are interested in.

Note: Instance statistics are dynamic and are reset at every instance startup. These statistics can be captured at a point in time and held in the database in the form of snapshots.

Wait Events Statistics

All the possible wait events are cataloged in the `V$EVENT_NAME` view.

Cumulative statistics for all sessions are stored in `V$SYSTEM_EVENT`, which shows the total waits for a particular event since instance startup.

When you are troubleshooting, you need to know whether a process has waited for any resource.

Systemwide Statistics

All the systemwide statistics are cataloged in the `V$STATNAME` view: Over 400 statistics are available in Oracle Database.

The server displays all calculated system statistics in the V$SYSSTAT view. You can query this view to find cumulative totals since the instance started.

Example:

```
SQL>  SELECT name, class, value FROM v$sysstat;
NAME                                CLASS      VALUE
------------------------------  ------ ----------
...
table scans (short tables)            64     135116
table scans (long tables)             64        250
table scans (rowid ranges)            64          0
table scans (cache partitions)        64          3
table scans (direct read)             64          0
table scan rows gotten                64   14789836
table scan blocks gotten              64     558542
...
```

Systemwide statistics are classified by the tuning topic and the debugging purpose. The classes include general instance activity, redo log buffer activity, locking, database buffer cache activity, and so on. Each of the system statistics can belong to more than one class, so you cannot do a simple join on V$SYSSTATS.CLASS and V$SYSTEM_WAIT_CLASS.WAIT_CLASS#.

You can also view all wait events for a particular wait class by querying V$SYSTEM_WAIT_CLASS, as in this example (with formatting applied):

```
SQL> SELECT * FROM V$SYSTEM_WAIT_CLASS
  2  WHERE wait_class LIKE '%I/O%';
CLASS_ID   CLASS# WAIT_CLASS   TOTAL_WAITS TIME_WAITED
---------- ------ ------------ ----------- -----------
1740759767      8 User I/O         1119152       39038
4108307767      9 System I/O        296959       27929
```

SGA Global Statistics

The server displays all calculated memory statistics in the V$SGASTAT view. You can query this view to find cumulative totals of detailed SGA usage since the instance started, as in the following example:

```
SQL>  SELECT * FROM v$sgastat;
POOL            NAME                           BYTES
------          ------------------------  ----------
                fixed_sga                     7780360
                buffer_cache                 25165824
                log_buffer                     262144
shared pool     sessions                      1284644
shared pool     sql area                     22376876
...
```

The results shown are only a partial display of the output.

When the STATISTICS_LEVEL parameter is set to BASIC, the value of the TIMED_STATISTICS parameter defaults to FALSE. Timing information is not collected for wait events and much of the performance-monitoring capability of the database is disabled. The explicit setting of TIMED_STATISTICS overrides the value derived from STATISTICS_LEVEL.

SQL Plan Directives

- A SQL plan directive is additional information and instructions that the optimizer can use to generate a better plan:
  - Collect missing statistics
  - Create column group statistics
  - Perform dynamic sampling
- Directives can be used for multiple statements:
  - Directives are collected on query expressions
- They are persisted to disk in the `SYSAUX` tablespace.
- Directives are automatically maintained:
  - Created as needed during compilation or execution:
    - Missing statistics, cardinality misestimates
  - Purged if not used after a year

In Oracle Database 12*c*, the database server can use a SQL plan directive, which is additional information and instructions that the optimizer can use to generate a more optimal plan. For example, a SQL plan directive might instruct the optimizer to collect missing statistics, create column group statistics, or perform dynamic sampling. During SQL compilation or execution, the database analyzes the query expressions that are missing statistics or that misestimate optimizer cardinality to create SQL plan directives. When the optimizer generates an execution plan, the directives give the optimizer additional information about objects that are referenced in the plan.

SQL plan directives are not tied to a specific SQL statement or `SQL ID`. The optimizer can use SQL plan directives for SQL statements that are nearly identical because SQL plan directives are defined on a query expression. For example, directives can help the optimizer with queries that use similar patterns, such as web-based queries that are the same except for a select list item. The database stores SQL plan directives persistently in the `SYSAUX` tablespace. When generating an execution plan, the optimizer can use SQL plan directives to obtain more information about the objects that are accessed in the plan.

Directives are automatically maintained, created as needed, purged if not used after a year.

Directives can be monitored in `DBA_SQL_PLAN_DIR_OBJECTS`. SQL plan directives improve plan accuracy by persisting both compilation and execution statistics in the `SYSAUX` tablespace, allowing them to be used by multiple SQL statements.

Adaptive Execution Plans

- A query plan changes during execution because runtime conditions indicate that optimizer estimates are inaccurate.
- All adaptive execution plans rely on statistics that are collected during query execution.
- The two adaptive plan techniques are:
  - Dynamic plans
  - Re-optimization
- The database uses adaptive execution plans when `OPTIMIZER_FEATURES_ENABLE` is set to `12.1.0.1` or higher, and `OPTIMIZER_ADAPTIVE_REPORTING_ONLY` is set to the default value of `FALSE`.

The Adaptive Execution Plans feature enables the optimizer to automatically adapt a poorly performing execution plan at run time and prevent a poor plan from being chosen on subsequent executions. The optimizer instruments its chosen plan so that at run time, it can be detected if the optimizer's estimates are not optimal. Then the plan can be automatically adapted to the actual conditions. An adaptive plan is a plan that changes after optimization when optimizer estimates prove inaccurate.

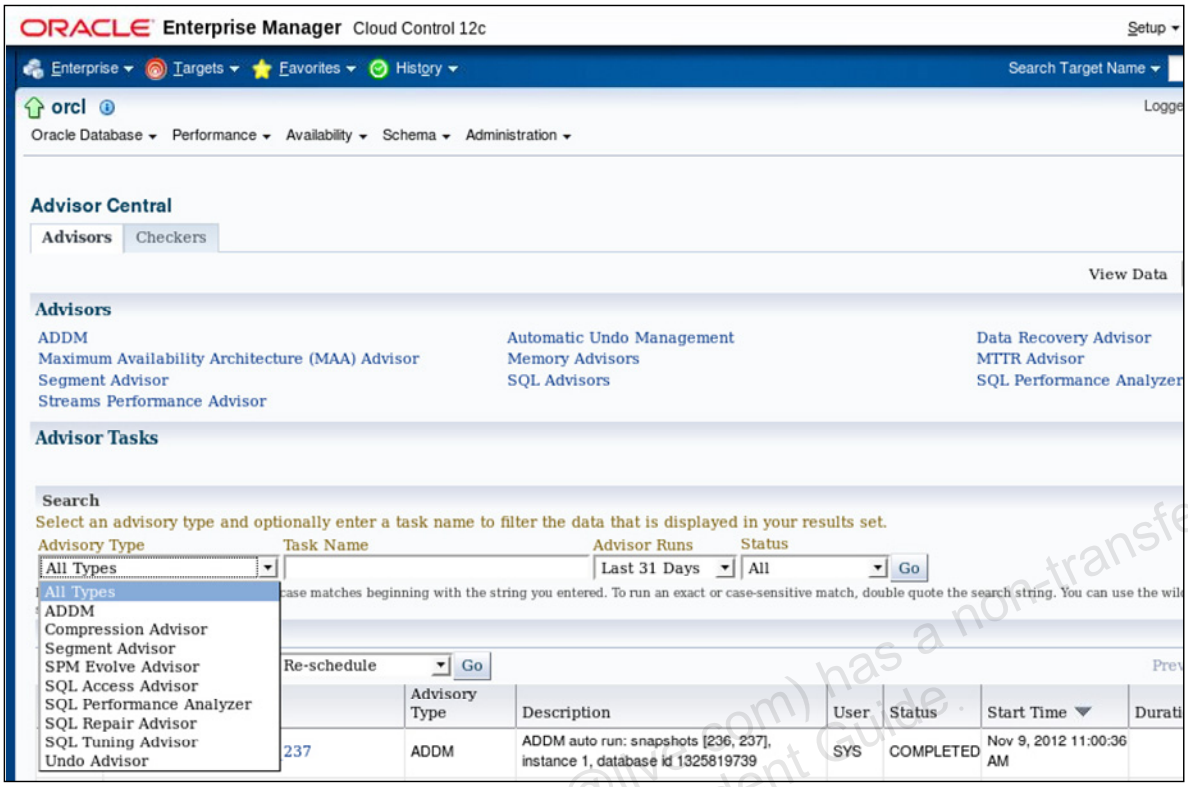
The optimizer can adapt plans based on statistics that are collected during statement execution. All adaptive mechanisms can execute a plan that differs from the plan which was originally determined during hard parse. This improves the ability of the query-processing engine (compilation and execution) to generate better execution plans.

The two Adaptive Plan techniques are:

- **Dynamic plans:** A dynamic plan chooses among subplans during statement execution. For dynamic plans, the optimizer must decide which subplans to include in a dynamic plan, which statistics to collect to choose a subplan, and thresholds for this choice.
- **Re-optimization:** In contrast, re-optimization changes a plan for executions after the current execution. For re-optimization, the optimizer must decide which statistics to collect at which points in a plan and when re-optimization is feasible.

Note: `OPTIMIZER_ADAPTIVE_REPORTING_ONLY` controls reporting-only mode for adaptive optimizations. When set to `TRUE`, adaptive optimizations run in reporting-only mode where the information required for an adaptive optimization is gathered, but no action is taken to change the plan.

Using the SQL Advisors



Oracle Database provides a set of advisors for SQL. The AWR identifies and records statistics about the recent high-load SQL statements.

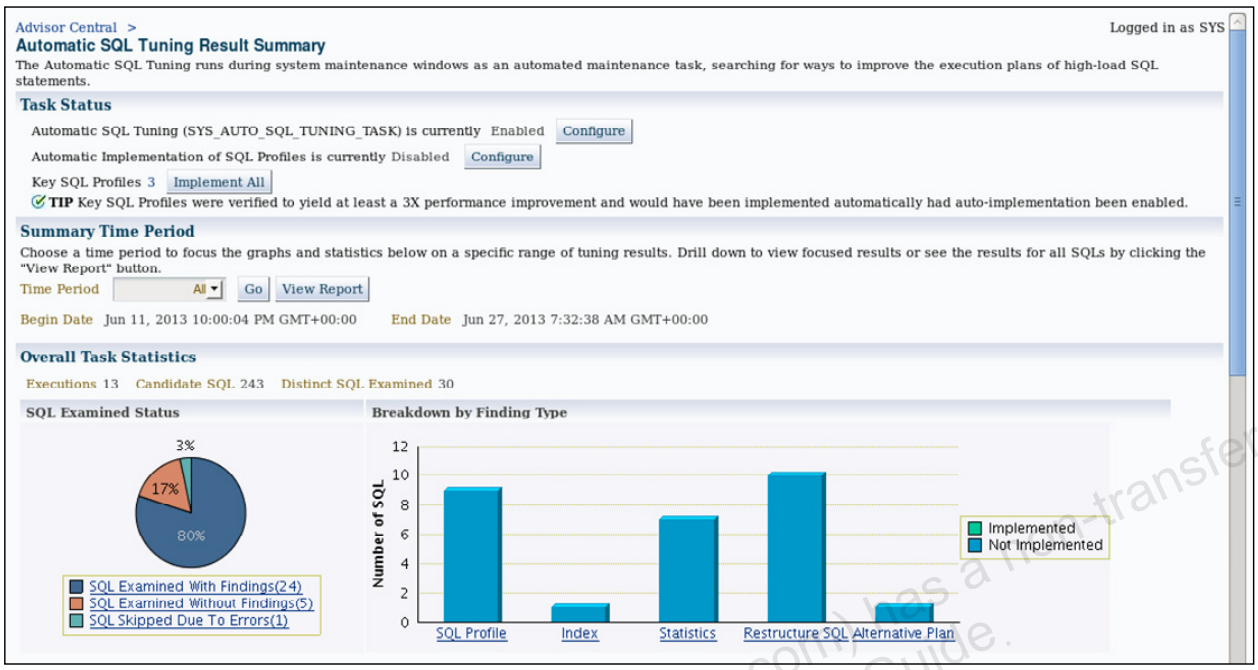
The SQL Access Advisor considers changes applied to a set of SQL statements and looks for a net gain in performance. This set can be a hypothetical set of SQL, a historical set, or a manually created set.

The SQL Performance Analyzer can be used to predict and prevent potential performance problems for any database environment change that affects the structure of the SQL execution plans.

The SQL Repair Advisor is run from the Support Workbench when a SQL statement fails with a critical error. A critical error also produces an incident. The repair advisor attempts to find and recommend a SQL patch. If no patch is found, you can continue in the Support Workbench to package the incident and submit the package to Oracle Support as a Service Request (SR).

The SQL Tuning Advisor analyzes one or more SQL statements one at a time. It examines statistics, SQL profiles, indexes, materialized views, and restructured SQL. The SQL Tuning Advisor can be run manually at any time, but it is run during every maintenance window against the recent high-load SQL statements. Click Automatic SQL Tuning Results to view and implement the recommendations. This automatic job can be configured to automatically implement recommended SQL profiles for the high-load statements.

Automatic SQL Tuning Results

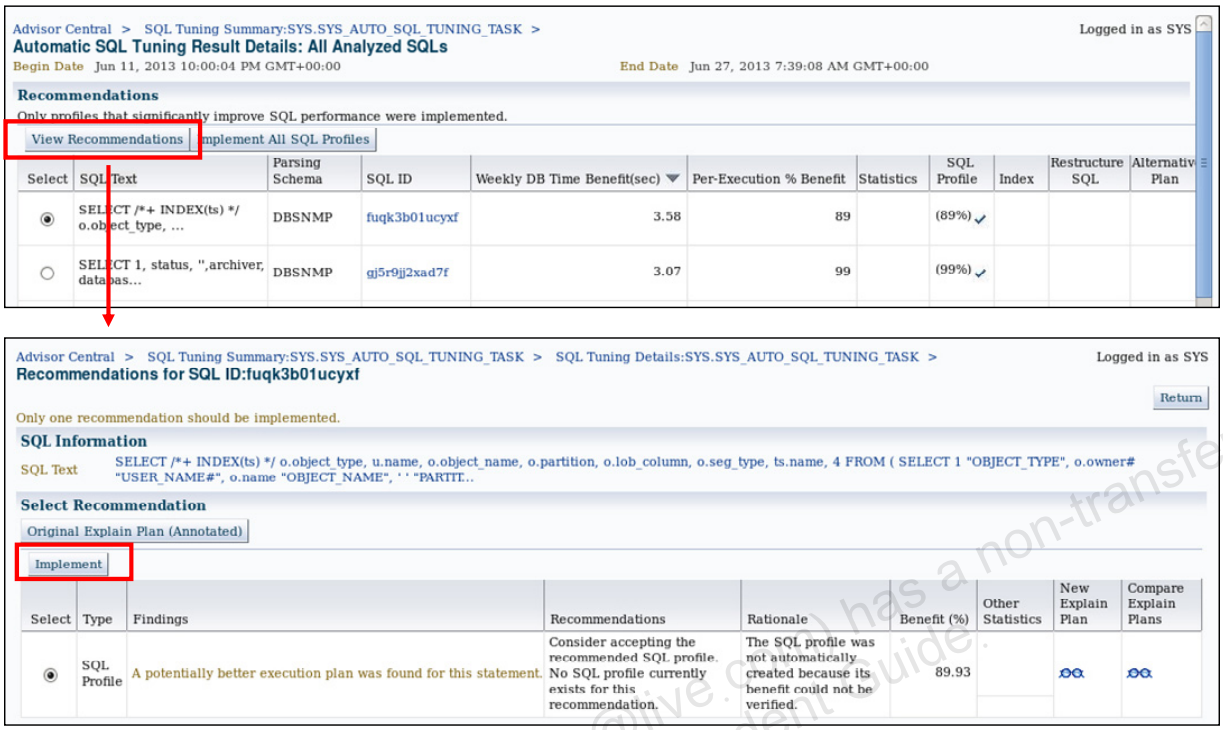


The Automatic SQL Tuning Task runs by default every night.

You can access the Automatic SQL Tuning Result Summary page by selecting Automated Maintenance Tasks in the Oracle Scheduler submenu of the Administration menu. Select the Automatic SQL Tuning link to view the result summary page.

Click Configure to display a page where you can change the defaults of the Automatic Tuning Task and enable automatic implementation of SQL profiles.

Implementing Automatic Tuning Recommendations



If you click View Report on the Automatic Tuning Results Summary page, you will see the Automatic SQL Tuning Result Details. You can implement all the recommendations or drill down to view or implement individual recommendations. On the Recommendations page, you can click the eyeglass icon on the right to see the differences that implementing a SQL profile will make in the explain plan.

SQL Tuning Advisor: Overview

SQL Tuning Advisor

Comprehensive SQL tuning

- **Detect stale or missing statistics**
- **Tune SQL plan (SQL profile)**
- **Add missing index**
- **Restructure SQL**

The SQL Tuning Advisor is the primary driver of the tuning process. It performs several types of analyses:

- **Statistics Analysis:** Checks each query object for missing or stale statistics, and makes recommendations to gather relevant statistics.
- **SQL Profiling:** The optimizer verifies its own estimates and collects auxiliary information to remove estimation errors. It builds a SQL profile using the auxiliary information and makes a recommendation to create it. When a SQL profile is created, it enables the query optimizer to generate a well-tuned plan.
- **Access Path Analysis:** New indexes are considered if they significantly improve access to each table in the query. When appropriate, recommendations to create such objects are made.
- **SQL Structure Analysis:** SQL statements that use bad plans are identified and relevant suggestions are made to restructure them. The suggested changes can be syntactic as well as semantic.

The SQL Tuning Advisor considers each SQL statement included in the advisor task independently. Creating a new index may help a query, but may hurt the response time of DML. So, a recommended index or other object should be checked with the SQL Access Advisor over a workload (a set of SQL statements) to determine whether there is a net gain in performance.

Using the SQL Tuning Advisor

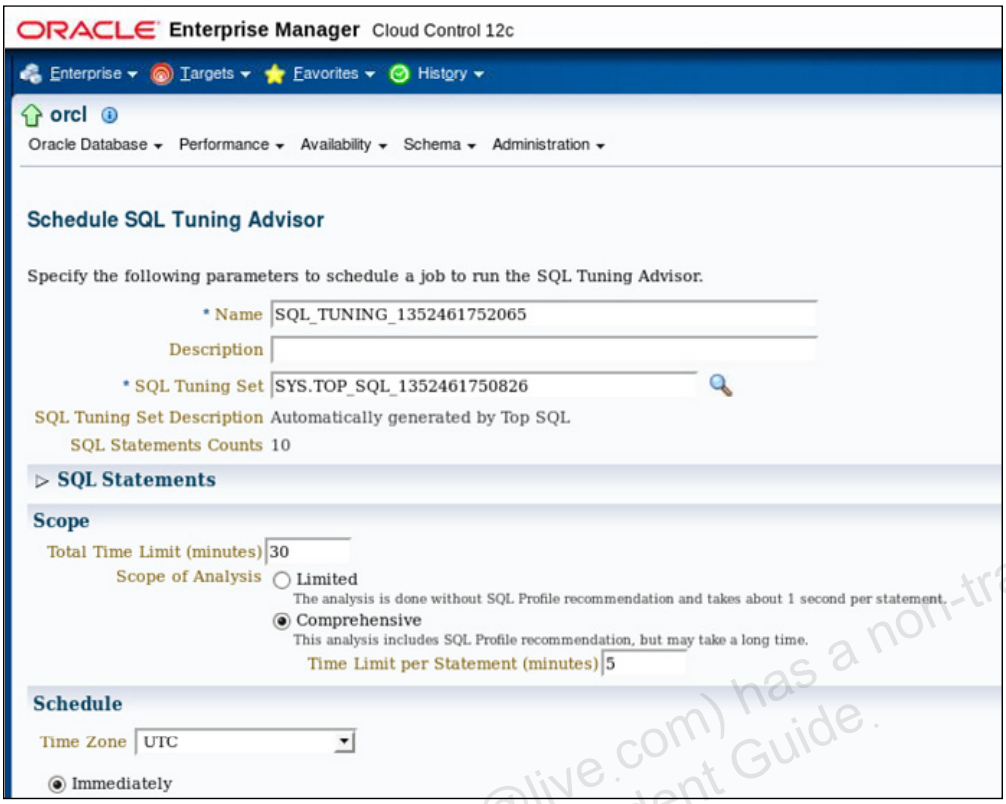
- Use the SQL Tuning Advisor to analyze SQL statements and obtain performance recommendations.
- Sources for SQL Tuning Advisor to analyze:
  - Top Activity: Analyzes the top SQL statements currently active
  - SQL Tuning Sets: Analyzes a set of SQL statements you provide
  - Historical SQL (AWR): Analyzes SQL statements from statements collected by AWR snapshots

The SQL Tuning Advisor runs automatically every night as the Automatic SQL Tuning Task. There may be times when a SQL statement needs immediate tuning action. You can use the SQL Tuning Advisor to analyze SQL statements and obtain performance recommendations at any time. Typically, you run this advisor as an ADDM performance-finding action.

Additionally, you can run the SQL Tuning Advisor when you want to analyze the top SQL statements consuming the most CPU time, I/O, and memory.

Even though you can submit multiple statements to be analyzed in a single task, each statement is analyzed independently. To obtain tuning recommendations that consider overall performance of a set of SQL, use the SQL Access Advisor.

Using the SQL Tuning Advisor

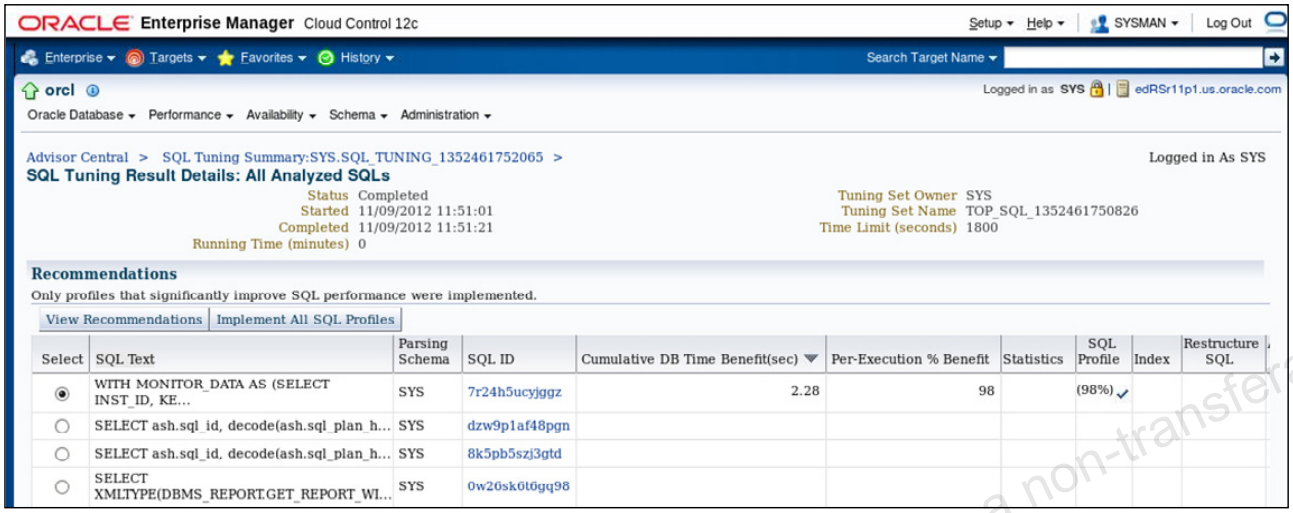


On the Schedule SQL Tuning Advisor page, you can choose the SQL statements to include and change the automatic defaults for a tuning task. You can set the source of the SQL statements, and if you have been granted the `ADVISOR` system privilege, you can submit the task. Enterprise Manager then creates a tuning task for the SQL Tuning Advisor.

The SQL statement options allow you to choose one or more SQL statements from recent Top Activity, choose Historical SQL stored in the AWR, or choose from a SQL Tuning Set that you have already created.

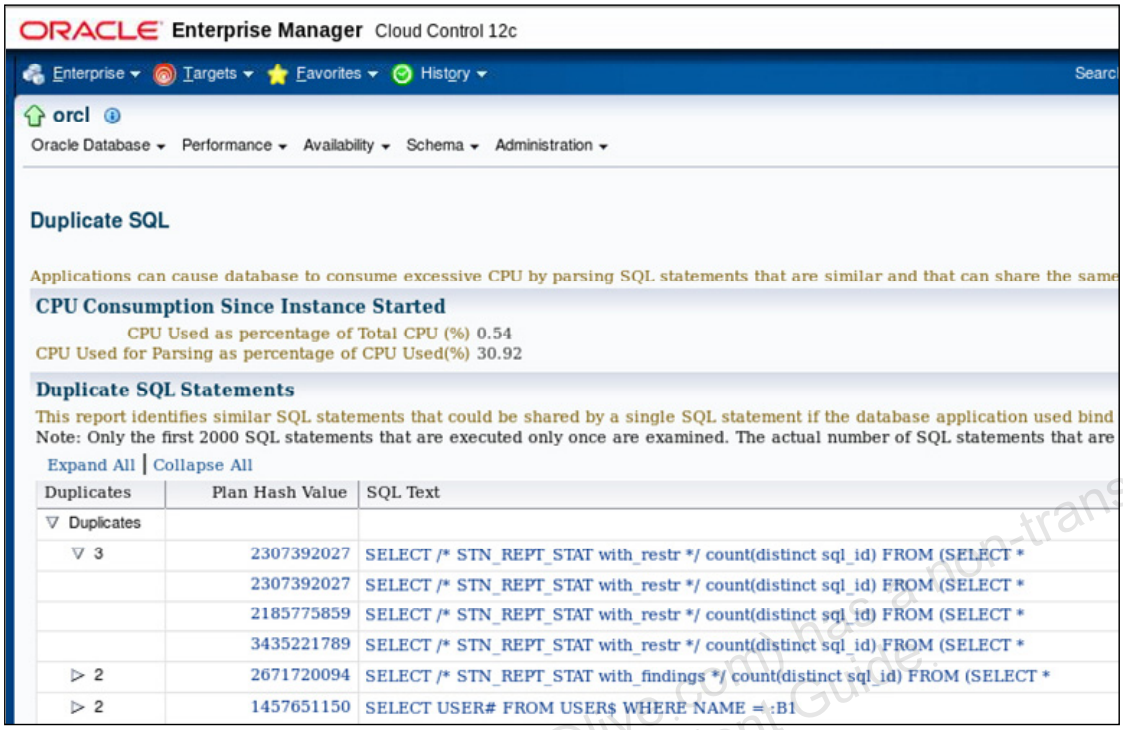
It is important to choose the appropriate scope for the tuning task. If you choose the Limited option, the SQL Tuning Advisor produces recommendations based on statistics check, access path analysis, and SQL structure analysis. The Limited option will not make a SQL profile recommendation. If you choose the Comprehensive option, the SQL Tuning Advisor produces all the recommendations that the Limited option produces, but it also invokes the optimizer under the SQL profiling mode to build a SQL profile. With the Comprehensive option, you can also specify a time limit for the tuning task, which by default is 30 minutes. After you select Run SQL Tuning Advisor, configure your tuning task using the SQL Tuning Options page.

SQL Tuning Advisor Recommendations



The SQL Tuning Results for the task are displayed as soon as the task completes and can also be accessed later from the Advisor Central page. A summary of the recommendations are displayed. You can review and implement individual recommendations. Select the statement and click view.

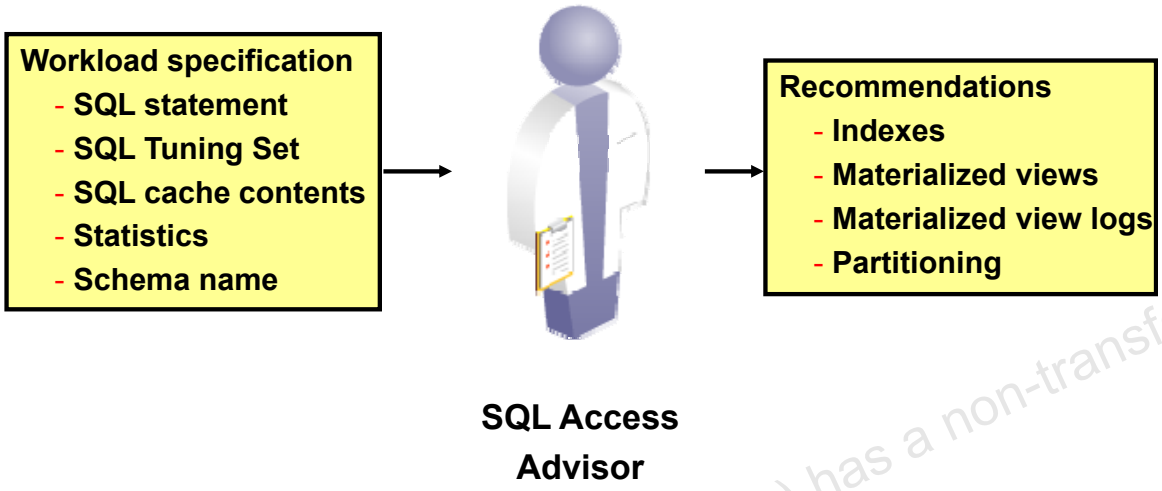
Duplicate SQL



Duplicate SQL statements are statements that are different only in the literal values they use or in their formatting. Statements that are different each get a separate cursor in the Library Cache. Statements that are duplicates can use the same cursor if the literals are replaced with bind variables, and the formatting is made to be the same.

Duplicate SQL statements can be identified by clicking Duplicate SQL in the Additional Monitoring Links section of the Performance Home page. SQL that is determined to be duplicate, except for formatting or literal differences, is listed together. This helps you determine which SQL in your application can be consolidated, thus lowering the requirements on the Library Cache and speeding up the execution of the statement.

SQL Access Advisor: Overview



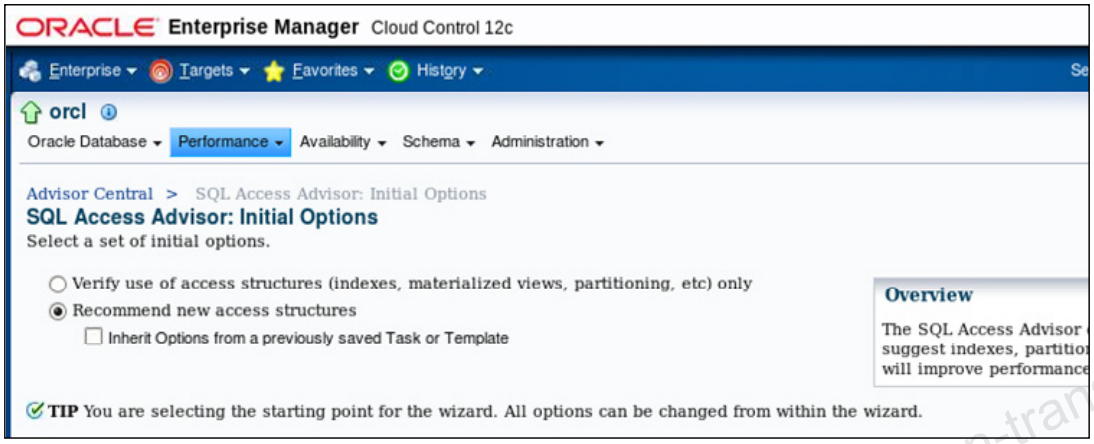
The SQL Access Advisor can recommend the proper set of materialized views, materialized view logs, partitioning, and indexes for a given workload. Understanding and using these structures is essential when optimizing SQL because they can result in significant performance improvements in data retrieval.

The SQL Access Advisor recommends bitmap, function-based, and B-tree indexes. A bitmap index offers a reduced response time for many types of ad hoc queries and reduced storage requirements compared to other indexing techniques. B-tree indexes are most commonly used in a data warehouse to index unique or near-unique keys.

Another component of the SQL Access Advisor also recommends how to optimize materialized views so that they can be fast refreshable and take advantage of general query rewrite.

Note: For more information about materialized views and query rewrite, see the *Oracle Database Performance Tuning Guide*.

Using the SQL Access Advisor

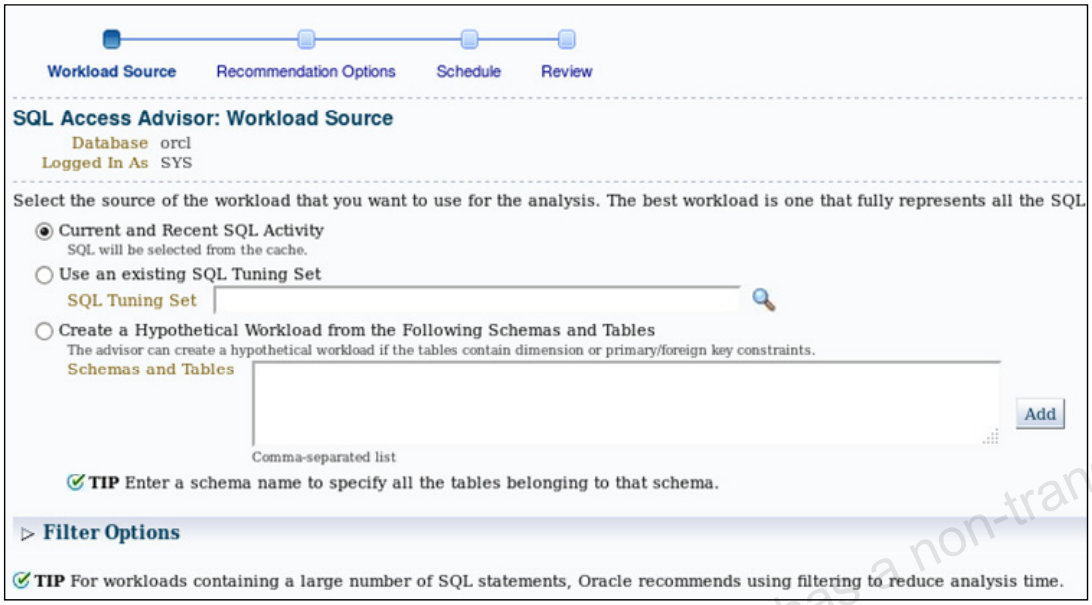


Access the SQL Access Advisor by selecting SQL Access Advisor in the SQL submenu of the Performance menu.

When starting a SQL Access Advisor session, you can select Use Default Options and start with a predefined set of advisor options that are recommended. Additionally, you can start a task and have it inherit a set of option values as defined by a template or task by selecting “Inherit Options from a Task or Template.” These include several generic templates designed for general-purpose environments, OLTP, and data warehousing databases. You can save custom templates from a previous task and reuse them when needed.

Click Continue to launch the SQL Access Advisor Wizard.

Workload Source



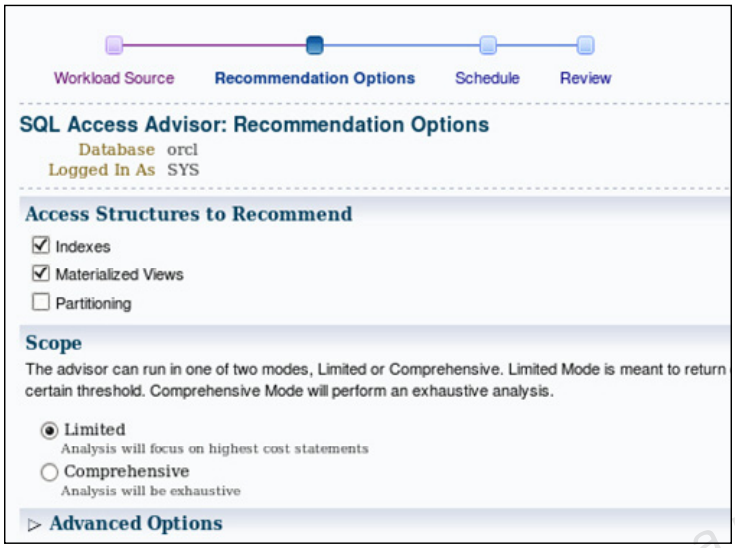
Use the SQL Access Advisor Wizard's Workload Source page to provide a defined workload that allows the Access Advisor to make recommendations. Supported workload sources are:

- **Current and Recent SQL Activity:** Uses the current SQL from the cache as the workload
- **Use an existing SQL Tuning Set:** Enables you to specify a previously created SQL Tuning Set as the workload source
- **Create a Hypothetical Workload from the Following Schemas and Tables:** Provides a schema that allows the advisor to search for dimension tables and produce a workload

The scope of the workload can be further reduced by applying filters that you can access in the Filter Options section. With these options, you can reduce the scope of the SQL statements that are present in the workload. The filters are applied to the workload by the advisor to focus the tuning effort. Possible filter options are:

- Top resource consuming SQL statements
- Users, module identifier, or actions
- Tables

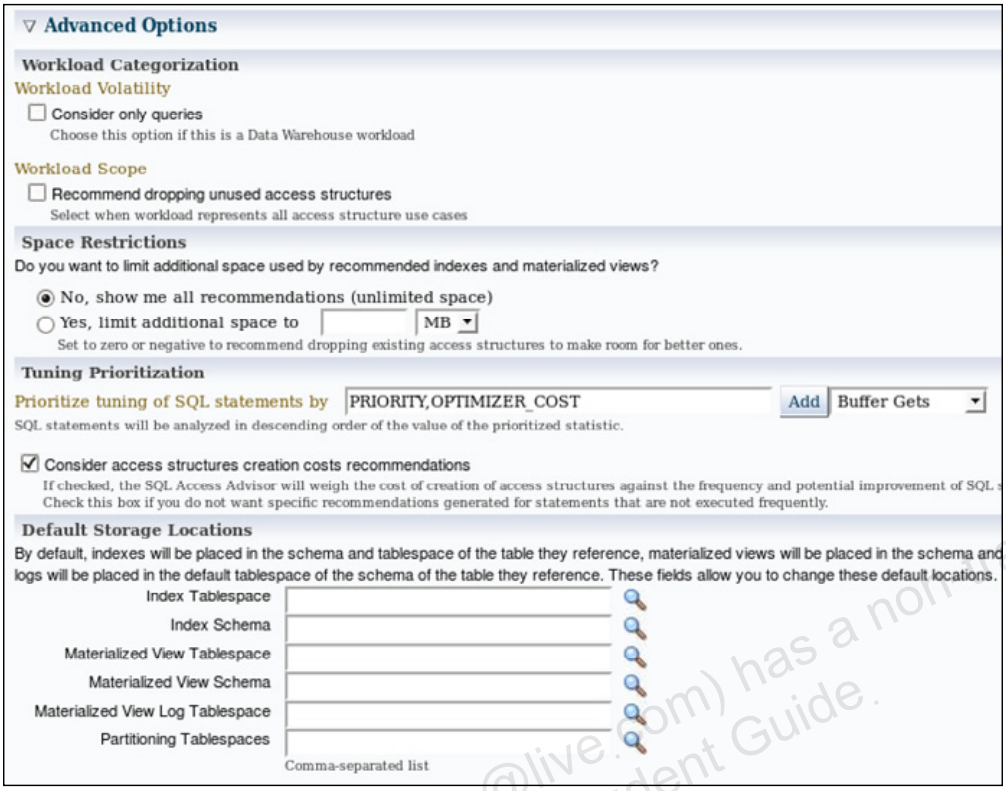
Recommendation Options



Use the Recommendations Options page to choose whether to limit the advisor to recommendations based on a single access method. Choose Indexes, Materialized Views, Partitioning, or a combination of those from the “Access Structures to Recommend” section. You can choose Evaluation Only to evaluate only existing access structures. In this mode, the advisor does not generate new recommendations but comments on the use of existing structures. This is useful to track the effectiveness of the current index, materialized view, and MV log usage over time.

You can use the Scope section to specify a mode of Limited or Comprehensive. These modes affect the quality of recommendations as well as the length of time required for processing. In the Comprehensive mode, the advisor searches a large pool of candidates resulting in recommendations of the highest quality. In the Limited mode, the advisor performs quickly, limiting the candidate recommendations.

Recommendation Options



You can choose Advanced Options to show or hide options that enable you to set space restrictions, tuning options, and default storage locations. Use the Workload Categorization section to set options for Workload Volatility and Workload Scope. You can choose to favor read-only operations or you can consider the volatility of referenced objects when forming recommendations. You can also select Partial Workload, which does not include recommendations, to drop unused access structures, or Complete Workload, which does include recommendations to drop unused access structures.

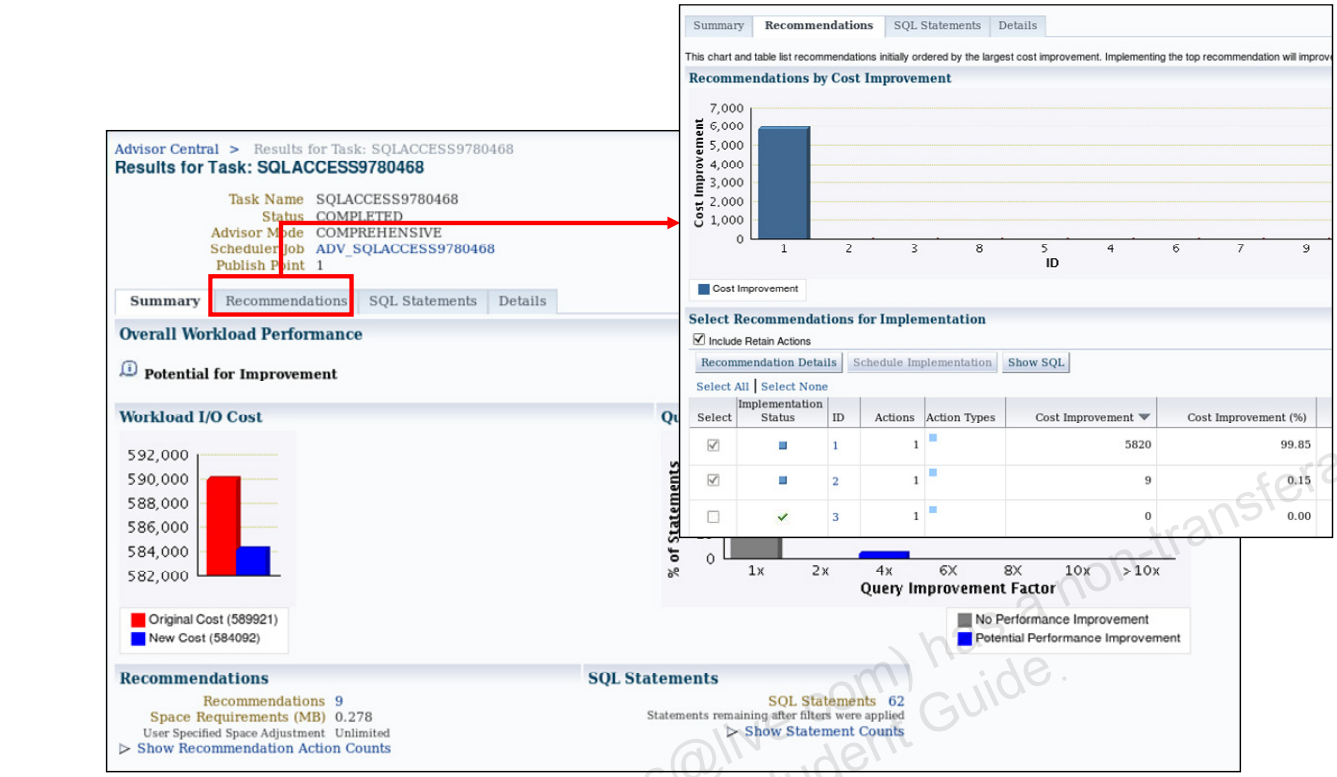
Use the Space Restrictions section to specify a hard space limit, which forces the advisor to produce recommendations only with total space requirements that do not exceed the specified limit.

Use the Tuning Options section to specify options that customize the recommendations made by the advisor. Use the “Prioritize Tuning of SQL Statements by” drop-down list to prioritize by Optimizer Cost, Buffer Gets, CPU Time, Disk Reads, Elapsed Time, and Execution Count.

Use the Default Storage Locations section to override the defaults defined for schema and tablespace locations. By default, indexes are placed in the schema and tablespace of the table they reference. Materialized views are placed in the schema and tablespace of the user who executed one of the queries that contributed to the materialized view recommendation.

After you define these parameters, you can schedule and review your tuning task.

Reviewing Recommendations



Using the Advisor Central page, you can list all the completed SQL Access Advisor tasks. Select the one for which you want to see the recommendations, and then click the View Result button. Use the Results for Task Summary page to get an overview of the advisor findings. The page presents charts and statistics that provide overall workload performance and query execution time potential improvement for the recommendations. You can use the page to show statement counts and recommendation action counts.

To see other aspects of the results for the advisor task, click one of the three other tabs on the page: Recommendations, SQL Statements, or Details.

The Recommendations page displays a chart and a table that show the top recommendations ordered by their percentage improvement to the total cost of the entire workload. The top recommendations have the biggest total performance improvement.

By clicking the Show SQL button, you can see the generated SQL script for the selected recommendations. You can click the corresponding recommendation identifier in the table to see the list of actions that need to be performed in order to implement the recommendation. On the Actions page, you can actually see all the corresponding SQL statements to execute in order to implement the action. For recommendations that you do not want to implement, keep those check boxes deselected. Then, click the Schedule Implementation button to implement the retained actions. This step is executed in the form of a Scheduler job.

SQL Performance Analyzer: Overview

- Targeted users: DBAs, QAs, application developers
- Helps predict the impact of system changes on SQL workload response time
- Builds different versions of SQL workload performance (that is, SQL execution plans and execution statistics)
- Executes SQL serially (concurrency not honored)
- Analyzes performance differences
- Offers fine-grained performance analysis on individual SQL
- Is integrated with SQL Tuning Advisor to tune regressions

Oracle Database includes SQL Performance Analyzer, which gives you an exact and accurate assessment of the impact of change on the SQL statements that make up the workload. SQL Performance Analyzer helps you forecast the impact of a potential change on the performance of a SQL query workload. This capability provides you with detailed information about the performance of SQL statements, such as before-and-after execution statistics, and statements with performance improvement or degradation. This enables you (for example) to make changes in a test environment to determine whether the workload performance will be improved through a database upgrade.

SQL Performance Analyzer: Use Cases

SQL Performance Analyzer is beneficial in the following use cases:

- Database upgrades
- Implementation of tuning recommendations
- Schema changes
- Statistics gathering
- Database parameter changes
- OS and hardware changes

It is accessible through Enterprise Manager and the `DBMS_SQLPA` package.

SQL Performance Analyzer can be used to predict and prevent potential performance problems for any database environment change that affects the structure of the SQL execution plans. The changes can include (but are not limited to) any of the following:

- Database upgrades
- Implementation of tuning recommendations
- Schema changes
- Statistics gathering
- Database parameter changes
- OS and hardware changes

You can use SQL Performance Analyzer to predict SQL performance changes that result from changes for even the most complex environments. As applications evolve through the development life cycle, database application developers can test changes to schemas, database objects, and rewritten applications to mitigate any potential performance impact.

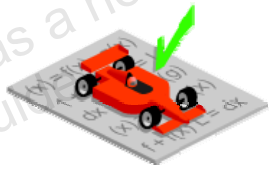
SQL Performance Analyzer also enables the comparison of SQL performance statistics.

You can access SQL Performance Analyzer through Enterprise Manager or by using the `DBMS_SQLPA` package.

For details about the `DBMS_SQLPA` package, see the *Oracle Database PL/SQL Packages and Types Reference Guide.*

Using SQL Performance Analyzer

1. Capture SQL workload on production.
2. Transport the SQL workload to a test system.
3. Build “before-change” performance data.
4. Make changes.
5. Build “after-change” performance data.
6. Compare results from steps 3 and 5.
7. Tune regressed SQL.



1. **Gather SQL:** In this phase, you collect the set of SQL statements that represent your SQL workload on the production system.
2. **Transport:** You must transport the resultant workload to the test system. The STS is exported from the production system and the STS is imported into the test system.
3. **Compute “before-version” performance:** Before any changes take place, you execute the SQL statements, collecting baseline information that is needed to assess the impact that a future change might have on the performance of the workload.
4. **Make a change:** After you have the before-version data, you can implement your planned change and start viewing the impact on performance.
5. **Compute “after-version” performance:** This step takes place after the change is made in the database environment. Each statement of the SQL workload runs under a mock execution (collecting statistics only), collecting the same information as captured in step 3.
6. **Compare and analyze SQL Performance:** After you have both versions of the SQL workload performance data, you can carry out the performance analysis by comparing the after-version data with the before-version data.
7. **Tune regressed SQL:** At this stage, you have identified exactly which SQL statements may cause performance problems when the database change is made. Here, you can use any of the database tools to tune the system. After implementing any tuning action, you should repeat the process to create a new after-version and analyze the performance differences to ensure that the new performance is acceptable.

Quiz

Even when you enable Automatic Maintenance tasks, the SQL Tuning Advisor always has to be started separately.

a. True
b. False

Answer: b

Quiz

You can receive performance recommendations for historical SQL statements that are collected by AWR snapshots.

a. True
b. False

Answer: a

Quiz

The SQL Access Advisor can recommend the proper set of materialized views, materialized view logs, partitioning, and indexes for a given workload.

a. True
b. False

Answer: a

Quiz

The SQL Performance Analyzer provides you with detailed information about the performance of SQL statements, such as before-and-after execution statistics, and statements with performance improvement or degradation.

a. True
b. False

Answer: a

Summary

In this lesson, you should have learned how to:

- Manage optimizer statistics
- Use the SQL Tuning Advisor to:
  - Identify SQL statements that are using the most resources
  - Tune SQL statements that are using the most resources
- Use the SQL Access Advisor to tune a workload

Practice 22

22-1: Using Automatic SQL Tuning