195
DBMS_SQLTUNE

The `DBMS_SQLTUNE` package is the interface for tuning SQL on demand. The related package `DBMS_AUTO_SQLTUNE` package provides the interface for SQL Tuning Advisor run as an automated task.

The chapter contains the following topics:

- DBMS_SQLTUNE Overview
- DBMS_SQLTUNE Security Model
- DBMS_SQLTUNE Data Structures
- DBMS_SQLTUNE Subprogram Groups
- Summary of DBMS_SQLTUNE Subprograms

> **See Also:**
>
> "DBMS_AUTO_SQLTUNE Overview"

DBMS_SQLTUNE Overview

The `DBMS_SQLTUNE` package provides a number of interrelated areas of functionality.

This section contains the following topics:

- DBMS_SQLTUNE SQL Tuning Advisor Subprograms
- DBMS_SQLTUNE SQL Profile Subprograms
- DBMS_SQLTUNE SQL Tuning Set Subprograms

SQL Tuning Advisor

SQL Tuning Advisor is one of a suite of advisors, a set of expert systems that identifies and helps resolve database performance problems. Specifically, SQL Tuning Advisor automates tuning of problematic SQL statements. It takes one or more SQL statements as input and gives precise advice on how to tune the statements. The advisor provides the advice in the form of SQL actions for tuning the SQL along with their expected performance benefit.

The group of DBMS_SQLTUNE SQL Tuning Advisor Subprograms provide a task-oriented interface that enables you to access the advisor. You can call the following subprograms in the order given to use some of SQL Tuning Advisor's features:

1. CREATE_TUNING_TASK Functions creates a tuning task for tuning one or more SQL statements.
2. The EXECUTE_TUNING_TASK Function and Procedure executes a previously created tuning task.
3. The REPORT_TUNING_TASK Function displays the results of a tuning task.

4. You use the SCRIPT_TUNING_TASK Function to create a SQL*Plus script which can then be executed to implement a set of Advisor recommendations

SQL Profile Subprograms

SQL Tuning Advisor may recommend the creation of a SQL profile to improve the performance of a statement. SQL profiles consist of auxiliary statistics specific to the statement. The query optimizer makes estimates about cardinality, selectivity, and cost that can sometimes be off by a significant amount, resulting in poor execution plans. The SQL profile addresses this problem by collecting additional information using sampling and partial execution techniques to adjust these estimates.

The group of DBMS_SQLTUNE SQL Profile Subprograms provides a mechanism for delivering statistics to the optimizer that targets one particular SQL statement, and helps the optimizer make good decisions for that statement by giving it the most accurate statistical information possible. For example:

- You can use the ACCEPT_SQL_PROFILE Procedure and Function to accept a SQL profile recommended by SQL Tuning Advisor.
- You can alter the `STATUS`, `NAME`, `DESCRIPTION`, and `CATEGORY` attributes of an existing SQL profile with the ALTER_SQL_PROFILE Procedure.
- You can drop a SQL profile with the DROP_SQL_PROFILE Procedure.

SQL Tuning Sets

SQL tuning sets store SQL statements along with the following information:

- The execution context, such as the parsing schema name and bind values
- Execution statistics such as average elapsed time and execution count
- Execution plans, which are the sequence of operations that the database performs to run SQL statements
- Row source statistics such as the number of rows processed for each operation executed within the plan

You can create SQL tuning sets by filtering or ranking SQL statements from several sources:

- The shared SQL area using the SELECT_CURSOR_CACHE Function
- Top SQL statements from the Automatic Workload Repository using the SELECT_WORKLOAD_REPOSITORY Function
- Other SQL tuning sets using the SELECT_SQLSET Function
- SQL Performance Analyzer task comparison results using the SELECT_SQLPA_TASK Function
- SQL Trace files using the SELECT_SQL_TRACE Function
- A user-defined workload

The complete group of DBMS_SQLTUNE SQL Tuning Set Subprograms facilitates this functionality. As examples:

- The CREATE_SQLSET Procedure and Function creates a SQL tuning set object in the database.
- The LOAD_SQLSET Procedure populates the SQL tuning set with a set of selected SQL.

- The CAPTURE_CURSOR_CACHE_SQLSET Procedure collects SQL statements from the shared SQL area over a specified time interval, attempting to build a realistic picture of database workload.

> **Note:**
>
> When manipulating SQL tuning sets, you can use DBMS_SQLSET as an alternative to `DBMS_SQLTUNE`.

Import and Export of SQL Tuning Sets and SQL Profiles

Use `DBMS_SQLTUNE` subprograms to move SQL profiles and SQL tuning sets from one system to another using a common programmatic model. In both cases, you create a staging table on the source database and populate this staging table with the relevant data. You then move that staging table to the destination system following the method of your choice (such as Oracle Data Pump, or a database link), where it is used to reconstitute the objects in their original form. The following steps are implemented by means of subprograms included in this package:

1. To create the staging table on the source system, call the CREATE_STGTAB_SQLPROF Procedure or the CREATE_STGTAB_SQLSET Procedure.
2. To populate the staging table with information from the source system, call the PACK_STGTAB_SQLPROF Procedure or PACK_STGTAB_SQLSET Procedure.
3. Move the staging table to the destination system.
4. To re-create the object on the new system, call the UNPACK_STGTAB_SQLPROF Procedure or the UNPACK_STGTAB_SQLSET Procedure.

> **See Also:**
>
> *Oracle Database SQL Tuning Guide* for more information about programmatic flow

Automatic Tuning Task Functions

The automated system task `SYS_AUTO_SQL_TUNING_TASK` is created by the database as part of the catalog scripts. This task automatically chooses a set of high-load SQL from AWR and runs SQL Tuning Advisor on this SQL. The automated task performs the same comprehensive analysis as any other SQL Tuning task.

You can obtain a report on the activity of the Automatic SQL Tuning task through the `DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK` API.

> **See Also:**
>
> DBMS_AUTO_SQLTUNE for the list of subprograms that you can use to manage the automated SQL tuning task.

Real-Time SQL Monitoring

Real-time SQL Monitoring enables DBAs or performance analysts to monitor the execution of long-running SQL statements while they are executing. Both cursor statistics (such as CPU times and IO times) and execution plan statistics (such as number of output rows, memory and temp space used) are updated in almost real time during statement execution. The `V$SQL_MONITOR` and `V$SQL_PLAN_MONITOR` views expose these statistics. In addition, `DBMS_SQLTUNE` provides the `REPORT_SQL_MONITOR` and `REPORT_SQL_MONITOR_LIST` functions to report monitoring information.

> **Note:**
>
> `DBMS_SQL_MONITOR` also contains the `REPORT_SQL_MONITOR` and `REPORT_SQL_MONITOR_LIST` functions.

Tuning a Standby Database Workload

In some cases, a standby database can assume a reporting role in addition to its data protection role. The standby database can have its own workload of queries, some of which may require tuning. You can issue SQL Tuning Advisor statements on a standby database, which is read-only. A standby-to-primary database link enables `DBMS_SQLTUNE` to write data to and read data from the primary database. The procedures that are eligible for tuning standby workloads include the `database_link_to` parameter.

DBMS_SQLTUNE Security Model

This package is available to `PUBLIC` and performs its own security checking.

Note the following:

- Because SQL Tuning Advisor relies on the advisor framework, all tuning task interfaces (`*_TUNING_TASK`) require the `ADVISOR` privilege.
- SQL tuning set subprograms (`*_SQLSET`) require either of the following privileges:
  - `ADMINISTER SQL TUNING SET`

    You can only create and modify a SQL tuning set that you own.
  - `ADMINISTER ANY SQL TUNING SET`

    You can operate on all SQL tuning sets, even those owned by other users.
- In earlier releases, three different privileges were needed to invoke subprograms involving SQL profiles:
  - `CREATE ANY SQL PROFILE`
  - `ALTER ANY SQL PROFILE`
  - `DROP ANY SQL PROFILE`

  The preceding privileges have been deprecated in favor of `ADMINISTER SQL MANAGEMENT OBJECT` .

DBMS_SQLTUNE Data Structures

The `SELECT_*` subprograms in the `DBMS_SQLTUNE` package return objects of the `SQLSET_ROW` type.

Object Types

- SQLSET_ROW Object Type

SQLSET_ROW Object Type

The `SQLSET_ROW` object models the content of a SQL tuning set for the user.

Logically, a SQL tuning set is a collection of `SQLSET_ROW` objects. Each `SQLSET_ROW` contains a single SQL statement along with its execution context, statistics, binds, and plan. The `SELECT_*` subprograms each model a data source as a collection of `SQLSET_ROW` objects, with each object uniquely identified by (`sql_id`, `plan_hash_value`). Similarly, the `LOAD_SQLSET` procedure takes as input a cursor whose row type is `SQLSET_ROW`, treating each `SQLSET_ROW` in isolation according to the policies requested by the user.

Several subprograms package accept basic filters on the content of a SQL tuning set or data source. These filters are expressed in terms of the attributes within the `SQLSET_ROW` as defined.

Syntax

```
CREATE TYPE sqlset_row AS object (
  sql_id                    VARCHAR(13),
  force_matching_signature  NUMBER,
  sql_text                  CLOB,
  object_list               sql_objects,
  bind_data                 RAW(2000),
  parsing_schema_name       VARCHAR2(30),
  module                    VARCHAR2(48),
  action                    VARCHAR2(32),
  elapsed_time              NUMBER,
  cpu_time                  NUMBER,
  buffer_gets               NUMBER,
  disk_reads                NUMBER,
  direct_writes             NUMBER,
  rows_processed            NUMBER,
  fetches                   NUMBER,
  executions                NUMBER,
  end_of_fetch_count        NUMBER,
  optimizer_cost            NUMBER,
  optimizer_env             RAW(2000),
  priority                  NUMBER,
  command_type              NUMBER,
  first_load_time           VARCHAR2(19),
  stat_period               NUMBER,
  active_stat_period        NUMBER,
  other                     CLOB,
  plan_hash_value           NUMBER,
  sql_plan                  sql_plan_table_type,
  bind_list                 sql_binds,
```

```
  con_dbid                 NUMBER,
  last_exec_start_time     VARCHAR2(19))
```

Attributes

Table 195-1 SQLSET_ROW Attributes

Attribute	Description
sql_id	Unique SQL ID.
forcing_matching_signature	Signature with literals, case, and whitespace removed.
sql_text	Full text for the SQL statement.
object_list	Currently not implemented.
bind_data	Bind data as captured for this SQL. Note that you cannot stipulate an argument for this parameter and also for bind_list - they are mutually exclusive.
parsing_schema_name	Schema where the SQL is parsed.
module	Last application module for the SQL.
action	Last application action for the SQL.
elapsed_time	Sum total elapsed time for this SQL statement.
cpu_time	Sum total CPU time for this SQL statement.
buffer_gets	Sum total number of buffer gets.
disk_reads	Sum total number of disk reads.
direct_writes	Sum total number of direct path writes.
rows_processed	Sum total number of rows processed by this SQL.
fetches	Sum total number of fetches.
executions	Total executions of this SQL statement.
end_of_fetch_count	Number of times the SQL statement was fully executed with all of its rows fetched.
optimizer_cost	Optimizer cost for this SQL.
optimizer_env	Optimizer environment for this SQL statement.
priority	User-defined priority (1,2,3).
command_type	Statement type, such as INSERT or SELECT.
first_load_time	Load time of the parent cursor.
stat_period	Period of time (seconds) when the statistics of this SQL statement were collected.
active_stat_period	Effective period of time (in seconds) during which the SQL statement was active.
other	Other column for user-defined attributes.
plan_hash_value	Plan hash value of the plan.
sql_plan	Execution plan for the SQL statement.
bind_list	List of user-specified binds for the SQL statement. This is used for user-specified workloads. Note that you cannot stipulate an argument for this parameter and also for bind_data: they are mutually exclusive.

Table 195-1 (Cont.) SQLSET_ROW Attributes

Attribute	Description
con_dbid	DBID of the PDB or CDB root.
last_exec_start_time	Most recent execution start time of this SQL statement.

DBMS_SQLTUNE Subprogram Groups

DBMS_SQLTUNE subprograms are grouped by function.

- DBMS_SQLTUNE SQL Tuning Advisor Subprograms
- DBMS_SQLTUNE SQL Profile Subprograms
- DBMS_SQLTUNE SQL Tuning Set Subprograms
- DBMS_SQLTUNE Real-Time SQL Monitoring Subprograms
- DBMS_SQLTUNE SQL Performance Reporting Subprograms

DBMS_SQLTUNE SQL Tuning Advisor Subprograms

This subprogram group provides an interface to manage SQL tuning tasks.

Table 195-2 SQL Tuning Task Subprograms

Subprogram	Description
"CANCEL_TUNING_TASK Procedure"	Cancels the currently executing tuning task
"CREATE_SQL_PLAN_BASELINE Procedure"	Creates a SQL plan baseline for an existing plan
"CREATE_TUNING_TASK Functions"	Creates a tuning of a single statement or SQL tuning set for either SQL Tuning Advisor
"DROP_TUNING_TASK Procedure"	Drops a SQL tuning task
"EXECUTE_TUNING_TASK Function and Procedure"	Executes a previously created tuning task
"IMPLEMENT_TUNING_TASK Procedure"	Implements a set of SQL profile recommendations made by SQL Tuning Advisor
"INTERRUPT_TUNING_TASK Procedure"	Interrupts the currently executing tuning task
"REPORT_AUTO_TUNING_TASK Function"	Displays a report from the automatic tuning task, reporting on a range of executions
"REPORT_TUNING_TASK Function"	Displays the results of a tuning task
"RESET_TUNING_TASK Procedure"	Resets the currently executing tuning task to its initial state
"RESUME_TUNING_TASK Procedure"	Resumes a previously interrupted task that was created to process a SQL tuning set
"SCHEDULE_TUNING_TASK Function"	Creates a tuning task and schedules its execution as a scheduler job
"SCRIPT_TUNING_TASK Function"	Creates a SQL*Plus script which can then be executed to implement a set of SQL Tuning Advisor recommendations

Table 195-2 (Cont.) SQL Tuning Task Subprograms

Subprogram	Description
"SET_TUNING_TASK_PARAMETER Procedures"	Updates the value of a SQL tuning parameter of type `VARCHAR2` or `NUMBER`

"Summary of DBMS_SQLTUNE Subprograms" contains a complete listing of all subprograms in the package.

DBMS_SQLTUNE SQL Profile Subprograms

This subprogram group provides an interface to manage SQL profiles.

Table 195-3 SQL Profile Subprograms

Subprogram	Description
ACCEPT_ALL_SQL_PROFILES Procedure	Accepts all SQL profiles recommended by a specific execution of a tuning task
ACCEPT_SQL_PROFILE Procedure and Function	Creates a SQL profile for the specified tuning task
ALTER_SQL_PROFILE Procedure	Alters specific attributes of an existing SQL profile object
CREATE_STGTAB_SQLPROF Procedure	Creates the staging table used for copying SQL profiles from one system to another
DROP_SQL_PROFILE Procedure	Drops the named SQL profile from the database
PACK_STGTAB_SQLPROF Procedure	Moves profile data out of the `SYS` schema into the staging table
REMAP_STGTAB_SQLPROF Procedure	Changes the profile data values kept in the staging table prior to performing an unpack operation
SQLTEXT_TO_SIGNATURE Function	Returns a SQL text's signature
UNPACK_STGTAB_SQLPROF Procedure	Uses the profile data stored in the staging table to create profiles on this system

"Summary of DBMS_SQLTUNE Subprograms" contains a complete listing of all subprograms in the package.

DBMS_SQLTUNE SQL Tuning Set Subprograms

This subprogram group provides an interface to manage SQL tuning sets.

Table 195-4 SQL Tuning Set Subprograms

Subprogram	Description
ADD_SQLSET_REFERENCE Function	Adds a new reference to an existing SQL tuning set to indicate its use by a client
CAPTURE_CURSOR_CACHE_SQLSET Procedure	Over a specified time interval incrementally captures a workload from the shared SQL area into a SQL tuning set
CREATE_SQLSET Procedure and Function	Creates a SQL tuning set object in the database

Table 195-4 (Cont.) SQL Tuning Set Subprograms

Subprogram	Description
CREATE_STGTAB_SQLSET Procedure	Creates a staging table through which SQL Tuning Sets are imported and exported
DELETE_SQLSET Procedure	Deletes a set of SQL statements from a SQL tuning set
DROP_SQLSET Procedure	Drops a SQL tuning set if it is not active
LOAD_SQLSET Procedure	Populates the SQL tuning set with a set of selected SQL
PACK_STGTAB_SQLSET Procedure	Copies tuning sets out of the `SYS` schema into the staging table
REMAP_STGTAB_SQLSET Procedure	Changes the tuning set names and owners in the staging table so that they can be unpacked with different values than they had on the host system
REMOVE_SQLSET_REFERENCE Procedure	Deactivates a SQL tuning set to indicate it is no longer used by the client
SELECT_CURSOR_CACHE Function	Collects SQL statements from the shared SQL area
SELECT_SQL_TRACE Function	Reads the content of one or more trace files and returns the SQL statements it finds in the format of `sqlset_row`
SELECT_SQLPA_TASK Function	Collects SQL statements from a SQL performance analyzer comparison task
SELECT_SQLSET Function	Collects SQL statements from an existing SQL tuning set
SELECT_WORKLOAD_REPOSITORY Function	Collects SQL statements from the workload repository
UNPACK_STGTAB_SQLSET Procedure	Copies one or more SQL tuning sets from the staging table
UPDATE_SQLSET Procedures	Updates whether selected string fields for a SQL statement in a SQL tuning set or the set numerical attributes of a SQL in a SQL tuning set

The Summary of DBMS_SQLTUNE Subprograms contains a complete listing of all subprograms in the package.

DBMS_SQLTUNE Real-Time SQL Monitoring Subprograms

This subprogram group provides function to report on monitoring data collected in `V$SQL_MONITOR` and `V$SQL_PLAN_MONITOR`.

Table 195-5 Real-Time SQL Monitoring Subprograms

Subprogram	Description
REPORT_SQL_MONITOR Function	Reports on Real-Time SQL Monitoring
REPORT_SQL_MONITOR_LIST Function	Builds a report for all or a subset of statements monitored by Oracle Database
REPORT_SQL_MONITOR_LIST_XML Function	Builds an XML report for all or a subset of statements monitored by Oracle Database

DBMS_SQLTUNE SQL Performance Reporting Subprograms

This subprogram group provides detailed reports on SQL performance using statistics from the shared SQL area and automatic workload repository (AWR).

Table 195-6 SQL Performance Reporting Subprograms

Subprogram	Description
REPORT_SQL_DETAIL Function	This function reports on a specific SQL ID.
REPORT_SQL_MONITOR Function	This function builds a report (text, simple HTML, active HTML, XML) for the monitoring information collected on behalf of the targeted statement execution.
REPORT_SQL_MONITOR_LIST Function	This function builds a report for all or a sub-set of statements monitored by Oracle. For each statement, the subprogram gives key information and associated global statistics.
REPORT_TUNING_TASK Function	This function displays the results of a tuning task.
REPORT_TUNING_TASK_XML Function	This function displays an XML report of a tuning task.

Summary of DBMS_SQLTUNE Subprograms

This table lists the `DBMS_SQLTUNE` subprograms and briefly describes them.

Table 195-7 DBMS_SQLTUNE Package Subprograms

Subprogram	Description	Group
ACCEPT_ALL_SQL_PROFILES Procedure	Accepts all SQL profiles recommended by a particular execution of a particular tuning task	DBMS_SQLTUNE SQL Profile Subprograms
ACCEPT_SQL_PROFILE Procedure and Function	Creates a SQL profile for the specified tuning task	DBMS_SQLTUNE SQL Profile Subprograms
ADD_SQLSET_REFERENCE Function	Adds a new reference to an existing SQL tuning set to indicate its use by a client	DBMS_SQLTUNE SQL Tuning Set Subprograms
ALTER_SQL_PROFILE Procedure	Alters specific attributes of an existing SQL profile object	DBMS_SQLTUNE SQL Profile Subprograms
CANCEL_TUNING_TASK Procedure	Cancels the currently executing tuning task	DBMS_SQLTUNE SQL Tuning Advisor Subprograms
CAPTURE_CURSOR_CACHE_SQLSET Procedure	Over a specified time interval incrementally captures a workload from the shared SQL area into a SQL tuning set	DBMS_SQLTUNE SQL Tuning Set Subprograms
CREATE_SQL_PLAN_BASELINE Procedure	Creates a SQL plan baseline for an existing plan	DBMS_SQLTUNE SQL Tuning Advisor Subprograms
CREATE_SQLSET Procedure and Function	Creates a SQL tuning set object in the database	DBMS_SQLTUNE SQL Tuning Set Subprograms

Table 195-7 (Cont.) DBMS_SQLTUNE Package Subprograms

Subprogram	Description	Group
CREATE_STGTAB_SQLPROF Procedure	Creates the staging table used for copying SQL profiles from one system to another	DBMS_SQLTUNE SQL Profile Subprograms
CREATE_STGTAB_SQLSET Procedure	Creates a staging table through which SQL tuning sets are imported and exported	DBMS_SQLTUNE SQL Tuning Set Subprograms
CREATE_TUNING_TASK Functions	Creates a tuning of a single statement or SQL tuning set for either SQL Tuning Advisor	DBMS_SQLTUNE SQL Tuning Advisor Subprograms
DELETE_SQLSET Procedure	Deletes a set of SQL statements from a SQL tuning set	DBMS_SQLTUNE SQL Tuning Set Subprograms
DROP_SQL_PROFILE Procedure	Drops the named SQL profile from the database	DBMS_SQLTUNE SQL Profile Subprograms
DROP_SQLSET Procedure	Drops a SQL tuning set if it is not active	DBMS_SQLTUNE SQL Tuning Set Subprograms
DROP_TUNING_TASK Procedure	Drops a SQL tuning task	DBMS_SQLTUNE SQL Tuning Advisor Subprograms
EXECUTE_TUNING_TASK Function and Procedure	Executes a previously created tuning task	DBMS_SQLTUNE SQL Tuning Advisor Subprograms
IMPLEMENT_TUNING_TASK Procedure	implements a set of SQL profile recommendations made by SQL Tuning Advisor	DBMS_SQLTUNE SQL Tuning Advisor Subprograms
INTERRUPT_TUNING_TASK Procedure	Interrupts the currently executing tuning task	DBMS_SQLTUNE SQL Tuning Advisor Subprograms
LOAD_SQLSET Procedure	Populates the SQL tuning set with a set of selected SQL	DBMS_SQLTUNE SQL Tuning Set Subprograms
PACK_STGTAB_SQLPROF Procedure	Moves profile data out of the `SYS` schema into the staging table	DBMS_SQLTUNE SQL Profile Subprograms
PACK_STGTAB_SQLSET Procedure	Moves tuning sets out of the `SYS` schema into the staging table	DBMS_SQLTUNE SQL Tuning Set Subprograms
REMAP_STGTAB_SQLPROF Procedure	Changes the profile data values kept in the staging table prior to performing an unpack operation	DBMS_SQLTUNE SQL Profile Subprograms
REMAP_STGTAB_SQLSET Procedure	Changes the tuning set names and owners in the staging table so that they can be unpacked with different values than they had on the host system	DBMS_SQLTUNE SQL Tuning Set Subprograms
REMOVE_SQLSET_REFERENCE Procedure	Deactivates a SQL tuning set to indicate it is no longer used by the client	DBMS_SQLTUNE SQL Tuning Set Subprograms
REPORT_AUTO_TUNING_TASK Function	Displays a report from the automatic tuning task, reporting on a range of subtasks	DBMS_SQLTUNE SQL Tuning Set Subprograms
REPORT_SQL_DETAIL Function	Reports on a specific SQL ID	DBMS_SQLTUNE SQL Performance Reporting Subprograms

Table 195-7 (Cont.) DBMS_SQLTUNE Package Subprograms

Subprogram	Description	Group
REPORT_SQL_MONITOR Function	Builds a report (text, simple HTML, active HTML, XML) for the monitoring information collected on behalf of the targeted statement execution	DBMS_SQLTUNE Real-Time SQL Monitoring Subprograms
REPORT_SQL_MONITOR_LIST Function	Builds a report for all or a subset of statements monitored by Oracle Database. For each statement, the subprogram gives key information and associated global statistics	DBMS_SQLTUNE Real-Time SQL Monitoring Subprograms
REPORT_SQL_MONITOR_LIST_XML Function	Equivalent to the `REPORT_SQL_MONITOR_LIST` function, except that it returns `XMLType`	DBMS_SQLTUNE Real-Time SQL Monitoring Subprograms
REPORT_TUNING_TASK Function	Displays the results of a tuning task	DBMS_SQLTUNE SQL Performance Reporting Subprograms
REPORT_TUNING_TASK_XML Function	Displays an XML report of a tuning task	DBMS_SQLTUNE SQL Performance Reporting Subprograms
RESET_TUNING_TASK Procedure	Resets the currently executing tuning task to its initial state	DBMS_SQLTUNE SQL Tuning Advisor Subprograms
RESUME_TUNING_TASK Procedure	Resumes a previously interrupted task that was created to process a SQL tuning set	DBMS_SQLTUNE SQL Tuning Advisor Subprograms
SCHEDULE_TUNING_TASK Function	Creates a SQL tuning task and schedule its execution as a scheduler job	DBMS_SQLTUNE SQL Tuning Advisor Subprograms
SCRIPT_TUNING_TASK Function	Creates a SQL*Plus script which can then be executed to implement a set of SQL Tuning Advisor recommendations	DBMS_SQLTUNE SQL Tuning Advisor Subprograms
SELECT_CURSOR_CACHE Function	Collects SQL statements from the shared SQL area	DBMS_SQLTUNE SQL Tuning Set Subprograms
SELECT_SQL_TRACE Function	Reads the content of one or more trace files and returns the SQL statements it finds in the format of `sqlset_row`	DBMS_SQLTUNE SQL Tuning Set Subprograms
SELECT_SQLPA_TASK Function	Collects SQL statements from a SQL Performance Analyzer comparison task	DBMS_SQLTUNE SQL Tuning Set Subprograms
SELECT_SQLSET Function	Collects SQL statements from an existing SQL tuning set	DBMS_SQLTUNE SQL Tuning Set Subprograms
SELECT_WORKLOAD_REPOSITORY Function	Collects SQL statements from the workload repository	DBMS_SQLTUNE SQL Tuning Set Subprograms
SET_TUNING_TASK_PARAMETER Procedures	Updates the value of a SQL tuning parameter of type `VARCHAR2` or `NUMBER`	DBMS_SQLTUNE SQL Tuning Advisor Subprograms

Table 195-7 (Cont.) DBMS_SQLTUNE Package Subprograms

Subprogram	Description	Group
SQLTEXT_TO_SIGNATURE Function	Returns a SQL text's signature	DBMS_SQLTUNE SQL Profile Subprograms
UNPACK_STGTAB_SQLPROF Procedure	Uses the profile data stored in the staging table to create profiles on this system	DBMS_SQLTUNE SQL Profile Subprograms
UNPACK_STGTAB_SQLSET Procedure	Moves one or more SQL tuning sets from the staging table	DBMS_SQLTUNE SQL Tuning Set Subprograms
UPDATE_SQLSET Procedures	Updates selected fields for a SQL statement in a SQL tuning set	DBMS_SQLTUNE SQL Tuning Set Subprograms

ACCEPT_ALL_SQL_PROFILES Procedure

This procedure accepts all SQL profiles recommended by a specific execution of a tuning task, and sets the attributes of the SQL profiles according to the parameter values passed by the user.

> **See Also:**
>
> DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.ACCEPT_ALL_SQL_PROFILES (
   task_name         IN VARCHAR2,
   category          IN VARCHAR2 := NULL,
   replace           IN BOOLEAN  := FALSE,
   force_match       IN BOOLEAN  := FALSE,
   profile_type      IN VARCHAR2 := REGULAR_PROFILE,
   autotune_period   IN NUMBER   := NULL,
   execution_name    IN VARCHAR2 := NULL,
   task_owner        IN VARCHAR2 := NULL,
   description       IN VARCHAR2 := NULL,
   database_link_to  IN VARCHAR2 := NULL);
```

Parameters

Table 195-8 ACCEPT_ALL_SQL_PROFILES Procedure Parameters

Parameter	Description
task_name	The (mandatory) name of the SQL tuning task

Table 195-8 (Cont.) ACCEPT_ALL_SQL_PROFILES Procedure Parameters

Parameter	Description
category	This is the category name which must match the value of the SQLTUNE_CATEGORY parameter in a session for the session to use this SQL profile. It defaults to the value "DEFAULT". This is also the default of the SQLTUNE_CATEGORY parameter. The category must be a valid Oracle identifier. The category name specified is always converted to upper case. The combination of the normalized SQL text and category name creates a unique key for a SQL profile. An ACCEPT_SQL_PROFILE fails if this combination is duplicated.
replace	If the profile already exists, it is replaced if this argument is TRUE. It is an error to pass a name that is already being used for another signature/ category pair, even with replace set to TRUE.
force_match	If TRUE this causes SQL profiles to target all SQL statements which have the same text after normalizing all literal values into bind variables. (Note that if a combination of literal values and bind values is used in a SQL statement, no bind transformation occurs.) This is analogous to the matching algorithm used by the FORCE option of the cursor_sharing parameter. If FALSE, literals are not transformed. This is analogous to the matching algorithm used by the EXACT option of the cursor_sharing parameter.
profile_type	Options: • REGULAR_PROFILE - profile without a change to parallel execution (Default, equivalent to NULL). Note that if the SQL statement currently has a parallel execution plan, the regular profile will cause the optimizer to choose a different, but still parallel, execution plan. • PX_PROFILE - regular profile with a change to parallel execution
autotune_period	The time period for the automatic SQL tuning. This setting applies only to the automatic SQL Tuning Advisor task. Possible values are as follows: • null or negative value (default) - all or full. The result includes all task executions. • 0 - result of the current or most recent task execution. • 1 - result for the most recent 24-hour period. • 7 - result for the most recent 7-day period. The procedure interprets any other value as the time of the most recent task execution minus the value of this argument.
execution_name	Name of the task execution to use. If null, then the procedure generates the report for the most recent task execution.
task_owner	Owner of the tuning task. This is an optional parameter that must be specified to accept a SQL profile associated to a tuning task owned by another user. The current user is the default value.
description	A user specified string describing the purpose of the SQL profile. The description is truncated if longer than 256 characters. The maximum size is 500 characters.

Table 195-8 (Cont.) ACCEPT_ALL_SQL_PROFILES Procedure Parameters

Parameter	Description
database_link_to	Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri: CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY *password* USING 'inst1';

Security Model

The ADMINISTER SQL MANAGEMENT OBJECT privilege is required. The CREATE ANY SQL PROFILE privilege is deprecated.

ACCEPT_SQL_PROFILE Procedure and Function

This subprogram creates a SQL profile recommended by SQL Tuning Advisor.

The SQL text is normalized for matching purposes although it is stored in the data dictionary in denormalized form for readability. SQL text is provided through a reference to the SQL Tuning task. If the referenced SQL statement does not exist, then the database reports an error.

> **See Also:**
>
> DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
   task_name     IN  VARCHAR2,
   object_id     IN  NUMBER   := NULL,
   name          IN  VARCHAR2 := NULL,
   description   IN  VARCHAR2 := NULL,
   category      IN  VARCHAR2 := NULL);
   task_owner    IN VARCHAR2  := NULL,
   replace       IN BOOLEAN   := FALSE,
```

```
   force_match  IN BOOLEAN   := FALSE,
   profile_type IN VARCHAR2  := REGULAR_PROFILE);


DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
   task_name         IN  VARCHAR2,
   object_id         IN  NUMBER    := NULL,
   name              IN  VARCHAR2  := NULL,
   description       IN  VARCHAR2  := NULL,
   category          IN  VARCHAR2  := NULL;
   task_owner        IN  VARCHAR2  := NULL,
   replace           IN  BOOLEAN   := FALSE,
   force_match       IN  BOOLEAN   := FALSE,
   profile_type      IN  VARCHAR2  := REGULAR_PROFILE,
   database_link_to  IN  VARCHAR2  := NULL)
 RETURN VARCHAR2;
```

Parameters

Table 195-9 ACCEPT_SQL_PROFILE Procedure and Function Parameters

Parameter	Description
task_name	The (mandatory) name of the SQL tuning task
object_id	The identifier of the advisor framework object representing the SQL statement associated with the tuning task
name	The name of the SQL profile. It cannot contain double quotation marks. The name is case sensitive. If not specified, the system generates a unique name for the SQL profile.
description	A user specified string describing the purpose of the SQL profile. The description is truncated if longer than 256 characters. The maximum size is 500 characters.
category	The category name. This name must match the value of the SQLTUNE_CATEGORY parameter in a session for the session to use this SQL profile. It defaults to the value "DEFAULT". This is also the default of the SQLTUNE_CATEGORY parameter. The category must be a valid Oracle identifier. The category name specified is always converted to upper case. The combination of the normalized SQL text and category name creates a unique key for a SQL profile. An ACCEPT_SQL_PROFILE fails if this combination is duplicated.
task_owner	Owner of the tuning task. This is an optional parameter that has to be specified to accept a SQL profile associated to a tuning task owned by another user. The current user is the default value.
replace	If the profile already exists, it is replaced if this argument is TRUE. It is an error to pass a name that is already being used for another signature/ category pair, even with replace set to TRUE.
force_match	If TRUE this causes SQL profiles to target all SQL statements which have the same text after normalizing all literal values into bind variables. (Note that if a combination of literal values and bind values is used in a SQL statement, no bind transformation occurs.) This is analogous to the matching algorithm used by the FORCE option of the cursor_sharing parameter. If FALSE, literals are not transformed. This is analogous to the matching algorithm used by the EXACT option of the cursor_sharing parameter.

Table 195-9 (Cont.) ACCEPT_SQL_PROFILE Procedure and Function Parameters

Parameter	Description
profile_type	Options: • REGULAR_PROFILE - profile without a change to parallel execution (Default, equivalent to NULL). Note that if the SQL statement currently has a parallel execution plan, the regular profile will cause the optimizer to choose a different, but still parallel, execution plan. • PX_PROFILE - regular profile with a change to parallel execution
database_link_to	Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri: CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY *password* USING 'inst1';

Return Values

The name of the SQL profile.

Usage Notes

The ADMINISTER SQL MANAGEMENT OBJECT privilege is required. The CREATE ANY SQL PROFILE privilege is deprecated.

Examples

You use both the procedure and the function versions of the subprogram in the same way except you must specify a return value to invoke the function. Here we give examples of the procedure only.

In this example, you tune a single SQL statement form the workload repository and you create the SQL profile recommended by SQL Tuning Advisor.

```
VARIABLE stmt_task VARCHAR2(64);
VARIABLE sts_task VARCHAR2(64);

-- create a tuning task tune the statement
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(
   begin_snap  => 1, -
   end_snap    => 2, -
   sql_id      => 'ay1m3ssvtrh24');

-- execute the resulting task
```

```
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:stmt_task);

EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE(:stmt_task);
```

Note that you do not have to specify the ID (that is, `object_id`) for the advisor framework object created by SQL Tuning Advisor to represent the tuned SQL statement.

You might also want to accept the recommended SQL profile in a different category, (for example, `TEST`), so that it is not used by default.

```
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
   task_name  =>  :stmt_task, -
   category   =>  'TEST');
```

You can use command `ALTER SESSION SET SQLTUNE_CATEGORY = 'TEST'` to see how this profile behaves.

The following call creates a SQL profile that targets any SQL statement with the same `force_matching_signature` as the tuned statement.

```
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (task_name   =>  :stmt_task, -
                                      force_match =>  TRUE);
```

In the following example, you tune a SQL tuning set, and you create a SQL profile for only one of the SQL statements in the SQL tuning set. The SQL statement is represented by an advisor framework object with ID equal to `5`. You must pass an object ID to the `ACCEPT_SQL_PROFILE` procedure because there are potentially many SQL profiles for the tuning task. This object ID is given along with the report.

```
EXEC :sts_task := DBMS_SQLTUNE.CREATE_TUNING_TASK ( -
  sqlset_name   => 'my_workload',  -
  rank1         => 'ELAPSED_TIME', -
  time_limit    => 3600,           -
  description   => 'my workload ordered by elapsed time');

-- execute the resulting task
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:sts_task);

 -- create the profile for the sql statement corresponding to object_id = 5.
EXEC DBMS_SQLTUNE.ACCEPT_SQL_PROFILE (
   task_name  =>  :sts_task, -
   object_id  =>  5);
```

ADD_SQLSET_REFERENCE Function

This procedure adds a new reference to an existing SQL tuning set to indicate its use by a client.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.ADD_SQLSET_REFERENCE (
   sqlset_name  IN  VARCHAR2,
   description  IN  VARCHAR2 := NULL)
 RETURN NUMBER;
```

Parameters

The parameters are identical for `DBMS_SQLTUNE.ADD_SQLSET_REFERENCE` and `DBMS_SQLSET.ADD_REFERENCE`.

Table 195-10 ADD_SQLSET_REFERENCE and ADD_REFERENCE Function Parameters

Parameter	Description
`sqlset_name`	Specifies the name of the SQL tuning set.
`description`	Provides an optional description of the usage of SQL tuning set. The description is truncated if longer than 256 characters.
`sqlset_owner`	Specifies the owner of the SQL tuning set, or `NULL` for the current schema owner.

Return Values

The identifier of the added reference.

Examples

You can add reference to a SQL tuning set. This prevents the tuning set from being modified while it is being used. References are automatically added when you invoke SQL Tuning Advisor on the SQL tuning set, so you should use this function for custom purposes only. The function returns a reference ID that is used to remove it later. You use the `REMOVE_SQLSET_REFERENCE` procedure to delete references to a SQL tuning set.

```
VARIABLE rid NUMBER;
EXEC :rid := DBMS_SQLTUNE.ADD_SQLSET_REFERENCE( -
                                sqlset_name => 'my_workload', -
                                description => 'my sts reference');
```

You can use the `DBA_SQLSET_REFERENCES` view to find all references on a given SQL tuning set.

ALTER_SQL_PROFILE Procedure

This procedure alters specific attributes of an existing SQL profile object.

The following attributes can be altered (using these attribute names):

- `STATUS` can be set to `ENABLED` or `DISABLED`.
- `NAME` can be reset to a valid name which must be a valid Oracle identifier and must be unique.
- `DESCRIPTION` can be set to any string of size no more than 500 characters.
- `CATEGORY` can be reset to a valid category name which must be a valid Oracle identifier and must be unique when combined with normalized SQL text).

> **See Also:**
>
> DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.ALTER_SQL_PROFILE (
   name                 IN  VARCHAR2,
   attribute_name       IN  VARCHAR2,
   value                IN  VARCHAR2);
```

Parameters

Table 195-11 ALTER_SQL_PROFILE Procedure Parameters

Parameter	Description
`name`	The (mandatory) name of the existing SQL profile to alter
`attribute_name`	The (mandatory) attribute name to alter (case insensitive) using valid attribute names
`value`	The (mandatory) new value of the attribute using valid attribute values

Usage Notes

Requires the `ALTER ANY SQL PROFILE` privilege.

Examples

```
-- Disable a profile, so it is not be used by any sessions.
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name            =>  :pname,   -
                                      attribute_name  =>  'STATUS', -
                                      value           =>  'DISABLED');

-- Enable it back:
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name            =>   :pname,   -
                                      attribute_name  =>   'STATUS', -
```

```
                                        value           =>   'ENABLED');

-- Change the category of the profile so it is used only by sessions
-- with category set to TEST.
-- Use ALTER SESSION SET SQLTUNE_CATEGORY = 'TEST' to see how this profile
-- behaves.
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name            =>  :pname,   -
                                      attribute_name  =>  'CATEGORY', -
                                      value           =>  'TEST');

-- Change it back:
EXEC DBMS_SQLTUNE.ALTER_SQL_PROFILE ( name            =>  :pname,   -
                                      attribute_name  =>  'CATEGORY', -
                                      value           =>  'DEFAULT');
```

CANCEL_TUNING_TASK Procedure

This procedure cancels the currently executing tuning task. All intermediate result data is deleted.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.CANCEL_TUNING_TASK (
 task_name          IN VARCHAR2);
```

Parameters

Table 195-12 CANCEL_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	Specifies the name of the task to cancel

Examples

You cancel a task when you need to stop it executing and do not require to view any already-completed results.

```
EXEC DBMS_SQLTUNE.CANCEL_TUNING_TASK(:my_task);
```

CAPTURE_CURSOR_CACHE_SQLSET Procedure

This procedure captures a workload from the shared SQL area into a SQL tuning set.

The procedure polls the cache multiple times over a time period, and updates the workload data stored there. It can execute over as long a period as required to capture an entire system workload.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET (
    sqlset_name         IN VARCHAR2,
    time_limit          IN POSITIVE := 1800,
    repeat_interval     IN POSITIVE := 300,
    capture_option      IN VARCHAR2 := 'MERGE',
    capture_mode        IN NUMBER   := MODE_REPLACE_OLD_STATS,
    basic_filter        IN VARCHAR2 := NULL,
    sqlset_owner        IN VARCHAR2 := NULL,
    recursive_sql       IN VARCHAR2 := HAS_RECURSIVE_SQL);
```

Parameters

The parameters are the same for both `DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET` and `DBMS_SQLSET.CAPTURE_CURSOR_CACHE`.

Table 195-13 CAPTURE_CURSOR_CACHE_SQLSET and CAPTURE_CURSOR_CACHE Procedure Parameters

Parameter	Description
`sqlset_name`	Specifies the SQL tuning set name
`time_limit`	Defines the total amount of time, in seconds, to execute.
`repeat_interval`	Defines the amount of time, in seconds, to pause between sampling.
`capture_option`	Specifies whether to insert new statements, update existing statements, or both. Values are `INSERT`, `UPDATE`, or `MERGE`. The values are the same as for `load_option` in `load_sqlset`.
`capture_mode`	Specifies the capture mode (`UPDATE` and `MERGE` capture options). Possible values: • `MODE_REPLACE_OLD_STATS` — Replaces statistics when the number of executions is greater than the number stored in the SQL tuning set • `MODE_ACCUMULATE_STATS` — Adds new values to current values for SQL that is already stored. Note that this mode detects if a statement has been aged out, so the final value for a statistics is the sum of the statistics of all cursors that statement existed under.

Table 195-13 (Cont.) CAPTURE_CURSOR_CACHE_SQLSET and CAPTURE_CURSOR_CACHE Procedure Parameters

Parameter	Description
basic_filter	Defines a filter to apply to the shared SQL area for each sample. If basic_filter is not set by the caller, then the subprogram captures only statements of type CREATE TABLE, INSERT, SELECT, UPDATE, DELETE, and MERGE.
sqlset_owner	Specifies the owner of the SQL tuning set or NULL for current schema owner
recursive_sql	Defines a filter that includes recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL) or excludes it (NO_RECURSIVE_SQL).

Examples

In this example capture takes place over a 30-second period, polling the cache once every five seconds. This captures all statements run during that period but not before or after. If the same statement appears a second time, the process replaces the stored statement with the new occurrence.

Note that in production systems the time limit and repeat interval would be set much higher. You should tune the time_limit and repeat_interval parameters based on the workload time and shared SQL area turnover properties of your system.

```
EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
                                        sqlset_name     => 'my_workload', -
                                        time_limit      =>  30, -
                                        repeat_interval =>  5);
```

In the following call you accumulate execution statistics as you go. This option produces an accurate picture of the cumulative activity of each cursor, even across age-outs, but it is more expensive than the previous example.

```
EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
                         sqlset_name     => 'my_workload', -
                         time_limit      => 30, -
                         repeat_interval => 5, -
                         capture_mode    =>
dbms_sqltune.MODE_ACCUMULATE_STATS);
```

This call performs a very inexpensive capture where you only insert new statements and do not update their statistics once they have been inserted into the SQL tuning set

```
EXEC DBMS_SQLTUNE.CAPTURE_CURSOR_CACHE_SQLSET( -
                                    sqlset_name     => 'my_workload', -
                                    time_limit      => 30, -
                                    repeat_interval => 5, -
                                    capture_option  => 'INSERT');
```

CREATE_SQL_PLAN_BASELINE Procedure

This procedure creates a SQL plan baseline for an execution plan. It can be used in the context of an Alternative Plan Finding made by SQL Tuning Advisor.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.CREATE_SQL_PLAN_BASELINE (
   task_name            IN VARCHAR2,
   object_id            IN NUMBER   := NULL,
   plan_hash_value      IN NUMBER,
   owner_name           IN VARCHAR2 := NULL,
   database_link_to     IN VARCHAR2 := NULL);
```

Parameters

Table 195-14 CREATE_SQL_PLAN_BASELINE Procedure Parameters

Parameter	Description
task_name	Name of the task for which to get a script
object_id	Object ID to which the SQL corresponds
plan_hash_value	Plan to create plan baseline
owner_name	Owner of the relevant tuning task. Defaults to the current schema owner.
database_link_to	Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri: `CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY password USING 'inst1';`

CREATE_SQLSET Procedure and Function

This procedure or function creates a SQL tuning set object in the database.

Syntax

```
DBMS_SQLTUNE.CREATE_SQLSET (
   sqlset_name  IN  VARCHAR2,
   description  IN  VARCHAR2 := NULL
   sqlset_owner IN  VARCHAR2 := NULL);


DBMS_SQLTUNE.CREATE_SQLSET (
   sqlset_name  IN  VARCHAR2 := NULL,
   description  IN  VARCHAR2 := NULL,
   sqlset_owner IN  VARCHAR2 := NULL)
 RETURN VARCHAR2;
```

Parameters

Table 195-15 CREATE_SQLSET Procedure Parameters

Parameter	Description
`sqlset_name`	Specifies the name of the created SQL tuning set. The name is the name passed to the function. If no name is passed to the function, then the function generates an automatic name.
`description`	Provides an optional description of the SQL tuning set.
`sqlset_owner`	Specifies the owner of the SQL tuning set, or `NULL` for the current schema owner.

Examples

```
EXEC DBMS_SQLTUNE.CREATE_SQLSET(-
  sqlset_name => 'my_workload', -
  description => 'complete application workload');
```

CREATE_STGTAB_SQLPROF Procedure

This procedure creates the staging table used for copying SQL profiles from one system to another.

> **See Also:**
>
> DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.CREATE_STGTAB_SQLPROF (
   table_name            IN VARCHAR2,
   schema_name           IN VARCHAR2 := NULL,
   tablespace_name       IN VARCHAR2 := NULL);
```

Parameters

Table 195-16 CREATE_STGTAB_SQLPROF Procedure Parameters

Parameter	Description
table_name	The name of the table to create (case-insensitive unless double quoted).
schema_name	The schema to create the table in, or NULL for the current schema (case-insensitive unless double quoted).
tablespace_name	The tablespace to store the staging table within, or NULL for the default tablespace of the current user (case-insensitive unless double quoted).

Usage Notes

- Call this procedure once before issuing a call to the PACK_STGTAB_SQLPROF Procedure.
- To put different SQL profiles in different staging tables, you can call this procedure multiple times.
- This is a DDL operation, so it does not occur within a transaction.

Examples

Create a staging table to store profile data that can be moved to another system.

```
EXEC DBMS_SQLTUNE.CREATE_STGTAB_SQLPROF (table_name  => 'PROFILE_STGTAB');
```

CREATE_STGTAB_SQLSET Procedure

This procedure creates a staging table through which SQL tuning sets are imported and exported.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.CREATE_STGTAB_SQLSET (
   table_name            IN VARCHAR2,
   schema_name           IN VARCHAR2 := NULL,
   tablespace_name       IN VARCHAR2 := NULL,
   db_version            IN NUMBER   := NULL);
```

Parameters

Table 195-17 CREATE_STGTAB_SQLSET and CREATE_STGTAB Procedure Parameters

Parameter	Description
table_name	Specifies the of the table to create. The name is case sensitive.
schema_name	Defines the schema in which to create the table, or NULL for the current schema. The name is case sensitive.
tablespace_name	Specifies the tablespace in which to store the staging table, or NULL for the default tablespace of the current user. The name is case sensitive.
db_version	Specifies the database version that determines the format of the staging table. You can also create an older database version staging table to export an STS to an older database version. Use one of the following values: • NULL (default) — Specifies the current database version. • STS_STGTAB_10_2_VERSION — Specifies the 10.2 database version. • STS_STGTAB_11_1_VERSION — Specifies the 11.1 database version. • STS_STGTAB_11_2_VERSION — Specifies the 11.2 database version. • STS_STGTAB_12_1_VERSION — Specifies the 12.1 database version. • STS_STGTAB_12_2_VERSION — Specifies the 12.2 database version.

Security Model

You must have CREATE TABLE permissions in the specified schema and tablespace.

Usage Notes

- Call this procedure once before packing the SQL set.
- To have different tuning sets in different staging tables, you can call this procedure multiple times.
- This is a DDL operation, so it does not occur within a transaction.
- The staging table contains nested table columns and indexes, so it should not be renamed.

Examples

Create a staging table for packing and eventually exporting a SQL tuning sets

```
EXEC DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(table_name => 'STGTAB_SQLSET');
```

Create a staging table to pack a SQL tuning set in Oracle Database 11*g* Release 2 (11.2) format

```
BEGIN
  DBMS_SQLTUNE.CREATE_STGTAB_SQLSET(
      table_name => 'STGTAB_SQLSET'
  ,   db_version => DBMS_SQLTUNE.STS_STGTAB_11_2_VERSION );
END;
```

CREATE_TUNING_TASK Functions

This function creates a SQL Tuning Advisor task.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

You can use different forms of this function to:

- Create a tuning task for a single statement given its text.
- Create a tuning task for a single statement from the shared SQL area given its identifier.
- Create a tuning task for a single statement from the workload repository given a range of snapshot identifiers.
- Create a tuning task for a SQL tuning set.
- Create a tuning task for SQL Performance Analyzer.

In all cases, the function mainly creates a SQL Tuning Advisor task and sets its parameters.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

SQL text format:

```
DBMS_SQLTUNE.CREATE_TUNING_TASK (
  sql_text         IN CLOB,
  bind_list        IN sql_binds := NULL,
  user_name        IN VARCHAR2  := NULL,
  scope            IN VARCHAR2  := SCOPE_COMPREHENSIVE,
  time_limit       IN NUMBER    := TIME_LIMIT_DEFAULT,
  task_name        IN VARCHAR2  := NULL,
  description      IN VARCHAR2  := NULL,
  con_name         IN VARCHAR2  := NULL,
  database_link_to IN VARCHAR2  := NULL)
RETURN VARCHAR2;
```

SQL ID format:

```
DBMS_SQLTUNE.CREATE_TUNING_TASK (
  sql_id           IN VARCHAR2,
```

```
  plan_hash_value  IN NUMBER    := NULL,
  scope            IN VARCHAR2  := SCOPE_COMPREHENSIVE,
  time_limit       IN NUMBER    := TIME_LIMIT_DEFAULT,
  task_name        IN VARCHAR2  := NULL,
  description      IN VARCHAR2  := NULL,
  con_name         IN VARCHAR2  := NULL,
  database_link_to IN VARCHAR2  := NULL)
RETURN VARCHAR2;
```

AWR format:

```
DBMS_SQLTUNE.CREATE_TUNING_TASK (
  begin_snap       IN NUMBER,
  end_snap         IN NUMBER,
  sql_id           IN VARCHAR2,
  plan_hash_value  IN NUMBER    := NULL,
  scope            IN VARCHAR2  := SCOPE_COMPREHENSIVE,
  time_limit       IN NUMBER    := TIME_LIMIT_DEFAULT,
  task_name        IN VARCHAR2  := NULL,
  description      IN VARCHAR2  := NULL,
  con_name         IN VARCHAR2  := NULL,
  dbid             IN NUMBER    := NULL,
  database_link_to IN VARCHAR2  := NULL)
RETURN VARCHAR2;
```

SQL tuning set format:

```
DBMS_SQLTUNE.CREATE_TUNING_TASK (
  sqlset_name       IN VARCHAR2,
  basic_filter      IN VARCHAR2 :=  NULL,
  object_filter     IN VARCHAR2 :=  NULL,
  rank1             IN VARCHAR2 :=  NULL,
  rank2             IN VARCHAR2 :=  NULL,
  rank3             IN VARCHAR2 :=  NULL,
  result_percentage IN NUMBER   :=  NULL,
  result_limit      IN NUMBER   :=  NULL,
  scope             IN VARCHAR2 :=  SCOPE_COMPREHENSIVE,
  time_limit        IN NUMBER   :=  TIME_LIMIT_DEFAULT,
  task_name         IN VARCHAR2 :=  NULL,
  description       IN VARCHAR2 :=  NULL
  plan_filter       IN VARCHAR2 :=  'MAX_ELAPSED_TIME',
  sqlset_owner      IN VARCHAR2 :=  NULL,
  database_link_to IN VARCHAR2  :=  NULL)
RETURN VARCHAR2;
```

SQL Performance Analyzer format:

```
DBMS_SQLTUNE.CREATE_TUNING_TASK (
   spa_task_name     IN VARCHAR2,
   spa_task_owner    IN VARCHAR2 :=  NULL,
   spa_compare_exec  IN VARCHAR2 :=  NULL,
   basic_filter      IN VARCHAR2 :=  NULL,
   time_limit        IN NUMBER   :=  TIME_LIMIT_DEFAULT,
   task_name         IN VARCHAR2 :=  NULL,
```

```
   description       IN VARCHAR2 :=  NULL)
RETURN VARCHAR2;
```

Parameters

Table 195-18 *CREATE_TUNING_TASK Function Parameters*

Parameter	Description
sql_text	Specifies the text of a SQL statement.
begin_snap	Specifies the begin snapshot identifier.
end_snap	Specifies the end snapshot identifier.
sql_id	Specifies the identifier of a SQL statement.
bind_list	Defines an ordered list of bind values in ANYDATA type. **NOTE:** This parameter is not supported on a standby database.
plan_hash_value	Specifies the hash value of the SQL execution plan.
sqlset_name	Specifies the SQL tuning set name.
basic_filter	Specifies the predicate used to filter the SQL from the SQL tuning set.
object_filter	Specifies the object filter.
rank(i)	Specifies an ORDER BY clause on the selected SQL statement.
result_percentage	Specifies the percentage on the sum of a ranking measure.
result_limit	Specifies the top L(imit) SQL from the filtered or ranked SQL.
user_name	Specifies the user name for whom the statement is to be tuned.
scope	Specifies the tuning scope: • LIMITED: SQL Tuning Advisor produces recommendations based on statistical checks, access path analysis, and SQL structure analysis. SQL profile recommendations are not generated. • COMPREHENSIVE: SQL Tuning Advisor carries out all the analysis it performs under limited scope plus SQL profiling.
time_limit	Specifies the maximum duration in seconds for the tuning session.
task_name	Specifies an optional tuning task name.
description	Provides a description of the SQL tuning session, up to a maximum of 256 characters.

Table 195-18 (Cont.) *CREATE_TUNING_TASK Function Parameters*

Parameter	Description
plan_filter	Specifies the plan filter. It is applicable when multiple plans (plan_hash_value) are associated with the same statement. This filter allows for selecting one plan (plan_hash_value) only. Possible values are: • LAST_GENERATED: most recent timestamp • FIRST_GENERATED: earliest timestamp, the opposite to LAST_GENERATED • LAST_LOADED: most recent first_load_time statistics information • FIRST_LOADED: earliest first_load_time statistics information, the opposite to LAST_LOADED • MAX_ELAPSED_TIME: maximum elapsed time • MAX_BUFFER_GETS: maximum buffer gets • MAX_DISK_READS: maximum disk reads • MAX_DIRECT_WRITES: maximum direct writes • MAX_OPTIMIZER_COST: maximum optimizer cost
sqlset_owner	Specifies the owner of the SQL tuning set, or NULL for the current schema owner.
spa_task_name	Specifies the name of the SQL Performance Analyzer task whose regressions are to be tuned.
spa_task_owner	Specifies the owner of specified SQL Performance Analyzer task or NULL for current user.
spa_compare_exec	Specifies the execution name of the Compare Performance trial of SQL Performance Analyzer task. If NULL, then the advisor uses the most recent execution of the given SQL Performance Analyzer task, of type COMPARE PERFORMANCE.
dbid	Specifies the DBID for imported or PDB-level AWR data. If NULL, then the current database DBID is used.

Table 195-18 (Cont.) *CREATE_TUNING_TASK Function Parameters*

Parameter	Description
con_name	Specifies the container for the tuning task. The semantics depend on the function format: • For the SQL text format, this parameter specifies the container in which SQL Tuning Advisor tunes the SQL statement. If null (default), then SQL Tuning Advisor uses the current container. • For the SQL ID format, this parameter specifies the container from which the database fetches the SQL statement for tuning. SQL Tuning Advisor tunes the statement in this container. If null, then the database uses the current PDB for tuning, fetches the statement from the cursor cache of all valid containers executing the SQL statement, and tunes the most expensive statement in its container. • For the AWR format, this parameter specifies the container from whose AWR data the database fetches the SQL statement for tuning. SQL Tuning Advisor tunes the statement in this container. If null, then the database uses the current PDB for tuning, fetches the statement from the AWR of all valid containers that have this SQL statement, and tunes the most expensive statement in its container. The following statements are true of all function formats: • In a non-CDB, this parameter is ignored. • In a PDB, this parameter must be null or match the container name of the PDB. Otherwise, an error occurs. • In a CDB root, this parameter must be null or match the container name of a container in this CDB. Otherwise, an error occurs.
database_link_to	The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri: `CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY password USING 'inst1';`

Return Values

A SQL tuning task name that is unique by user (two different users can give the same name to their advisor tasks).

Usage Notes

With regard to the form of this subprogram that takes a SQL tuning set, filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.

Example 195-1 Examples

The following examples assume the following variable definitions:

```
VARIABLE stmt_task     VARCHAR2(64);
VARIABLE sts_task      VARCHAR2(64);
VARIABLE spa_tune_task VARCHAR2(64);
```

Example 195-2 Create Tuning Task with SQL Text Format

```
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK( -
  sql_text => 'SELECT quantity_sold FROM sales s, times t WHERE s.time_id =
t.time_id AND s.time_id = TO_DATE(''24-NOV-00'')');
```

Example 195-3 Create Tuning Task with SQL ID Format

```
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'ay1m3ssvtrh24');

EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'ay1m3ssvtrh24',
-
   scope => 'LIMITED');

EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_id => 'ay1m3ssvtrh24',
-
   time_limit => 600);
```

Example 195-4 Create Tuning Task with AWR Snapshot Format

```
EXEC :stmt_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(begin_snap => 1, -
   end_snap => 2, sql_id => 'ay1m3ssvtrh24');
```

Example 195-5 Create Tuning Task with SQL Tuning Set Format

This example creates a task that tunes SQL statements in order by buffer gets, and also sets a time limit of one hour. The default ranking measure is elapsed time.

```
EXEC :sts_task := DBMS_SQLTUNE.CREATE_TUNING_TASK( -
  sqlset_name  => 'my_workload', -
  rank1        => 'BUFFER_GETS', -
  time_limit   => 3600, -
  description  => 'tune my workload ordered by buffer gets');
```

Example 195-6 Create Tuning Task with SPA Task Format

This example tunes the SQL statement that were reported as having regressed from the compare performance execution of the SQL Performance Analyzer task named `task_123`.

```
EXEC :spa_tune_task := DBMS_SQLTUNE.CREATE_TUNING_TASK(
   spa_task_name     => 'task_123',
   spa_task_owner    => 'SCOTT',
   spa_compare_exec  => 'exec1');
```

Example 195-7 Creating SQL Tuning Task on Standby Database

This example creates a tuning task on the standby database. The `tune_stby_wkld` task uses the `lnk_to_primary` database link to write data to the primary database, which is open read/write.

```
VAR tname VARCHAR2(30);
VAR query VARCHAR2(500);
EXEC :tname := 'tune_stby_wkld';
EXEC :query := 'SELECT /*+ FULL(t)*/ col1 FROM table1 t WHERE col1=9000';
EXEC :tname := DBMS_SQLTUNE.CREATE_TUNING_TASK(sql_text => :query,-
                 task_name => :tname, database_link_to => 'lnk_to_primary');
```

DELETE_SQLSET Procedure

This procedure deletes a set of SQL statements from a SQL tuning set.

Syntax

```
DBMS_SQLTUNE.DELETE_SQLSET (
   sqlset_name   IN  VARCHAR2,
   basic_filter  IN  VARCHAR2 := NULL,
   sqlset_owner  IN  VARCHAR2 := NULL);
```

Parameters

Table 195-19 DELETE_SQLSET Procedure Parameters

Parameter	Description
`sqlset_name`	Specifies the name of the SQL tuning set.
`basic_filter`	Specifies the SQL predicate to filter the SQL from the SQL tuning set. This basic filter is used as a where clause on the SQL tuning set content to select a desired subset of SQL from the SQL tuning set.
`sqlset_owner`	Specifies the owner of the SQL tuning set, or `NULL` for current schema owner.

Examples

```
-- Delete all statements in a sql tuning set.
EXEC DBMS_SQLTUNE.DELETE_SQLSET(sqlset_name   => 'my_workload');

-- Delete all statements in a sql tuning set which ran for less than a second
EXEC DBMS_SQLTUNE.DELETE_SQLSET(sqlset_name   => 'my_workload', -
                                basic_filter  => 'elapsed_time < 1000000');
```

DROP_SQL_PROFILE Procedure

This procedure drops the named SQL profile from the database.

> **See Also:**
>
> DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.DROP_SQL_PROFILE (
   name          IN  VARCHAR2,
   ignore        IN  BOOLEAN  := FALSE);
```

Parameters

Table 195-20 DROP_SQL_PROFILE Procedure Parameters

Parameter	Description
`name`	The (mandatory) name of SQL profile to be dropped. The name is case sensitive.
`ignore`	Ignores errors due to object not existing

Usage Notes

Requires the `DROP ANY SQL PROFILE` privilege.

Examples

```
-- Drop the profile:
EXEC DBMS_SQLTUNE.DROP_SQL_PROFILE(:pname);
```

DROP_SQLSET Procedure

This procedure drops a SQL tuning set if it is not active.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.DROP_SQLSET (
   sqlset_name   IN  VARCHAR2,
   sqlset_owner  IN  VARCHAR2 := NULL);
```

Parameters

Table 195-21 DROP_SQLSET Procedure Parameters

Parameter	Description
sqlset_name	Specifies the name of the SQL tuning set.
sqlset_owner	Specifies the owner of the SQL tuning set, or NULL for current schema owner.

Usage Notes

You cannot drop a SQL tuning set when it is referenced by one or more clients.

Examples

```
-- Drop the sqlset.
EXEC DBMS_SQLTUNE.DROP_SQLSET ('my_workload');
```

DROP_TUNING_TASK Procedure

This procedure drops a SQL tuning task. The task and all its result data are deleted.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.DROP_TUNING_TASK (
 task_name          IN VARCHAR2);
```

Parameters

Table 195-22 DROP_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	Specifies name of the tuning task to drop.

EXECUTE_TUNING_TASK Function and Procedure

This function and procedure executes a previously created tuning task. Both the function and the procedure run in the context of a new task execution. The difference is that the function version returns that new execution name.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.EXECUTE_TUNING_TASK (
   task_name         IN VARCHAR2,
   execution_name    IN VARCHAR2               := NULL,
   execution_params  IN dbms_advisor.argList   := NULL,
   execution_desc    IN VARCHAR2               := NULL,
   database_link_to  IN VARCHAR2               := NULL)
 RETURN VARCHAR2;


DBMS_SQLTUNE.EXECUTE_TUNING_TASK (
   task_name         IN VARCHAR2,
   execution_name    IN VARCHAR2               := NULL,
   execution_params  IN dbms_advisor.argList   := NULL,
   execution_desc    IN VARCHAR2               := NULL,
   database_link_to  IN VARCHAR2               := NULL);
```

Parameters

Table 195-23 EXECUTE_TUNING_TASK Function & Procedure Parameters

Parameter	Description
task_name	Name of the tuning task to execute.
execution_name	A name to qualify and identify an execution. If not specified, it is generated by the advisor and returned by function.
execution_params	List of parameters (name, value) for the specified execution. The execution parameters have effect only on the execution for which they are specified. They override the values for the parameters stored in the task (set through the SET_TUNING_TASK_PARAMETER Procedures).
execution_desc	A 256-length string describing the execution.

Table 195-23 (Cont.) EXECUTE_TUNING_TASK Function & Procedure Parameters

Parameter	Description
database_link_to	Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri: CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY *password* USING 'inst1';

Usage Notes

A tuning task can be executed multiples times without having to reset it.

Examples

```
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:stmt_task);
```

IMPLEMENT_TUNING_TASK Procedure

This procedure implements a set of SQL profile recommendations made by SQL Tuning Advisor.

Executing IMPLEMENT_TUNING_TASK is equivalent to executing the SCRIPT_TUNING_TASK Function and then running the script.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.IMPLEMENT_TUNING_TASK(
    task_name          IN  VARCHAR2,
    rec_type           IN  VARCHAR2 := REC_TYPE_SQL_PROFILES,
    owner_name         IN  VARCHAR2 := NULL,
    execution_name     IN  VARCHAR2 := NULL,
    database_link_to   IN  VARCHAR2 := NULL);
```

Parameters

Table 195-24 IMPLEMENT_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	Name of the tuning task for which to implement recommendations.
rec_type	Filter the types of recommendations to implement. Only 'PROFILES' is supported.
owner_name	Owner of the relevant tuning task or NULL for the current user.
execution_name	Name of the task execution to use. If NULL, then the procedure implements recommendations from the last task execution.
database_link_to	Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri: CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY *password* USING 'inst1';

INTERRUPT_TUNING_TASK Procedure

This procedure interrupts the currently executing tuning task. The task ends its operations as it would at normal exit so that the user can access the intermediate results.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.INTERRUPT_TUNING_TASK (
 task_name          IN VARCHAR2);
```

Parameters

Table 195-25 INTERRUPT_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	Name of the tuning task to interrupt

Examples

```
EXEC DBMS_SQLTUNE.INTERRUPT_TUNING_TASK(:my_task);
```

LOAD_SQLSET Procedure

This procedure populates the SQL tuning set with a set of selected SQL statements. You can call the procedure multiple times to add new SQL statements or replace attributes of existing statements.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.LOAD_SQLSET (
   sqlset_name       IN  VARCHAR2,
   populate_cursor   IN  sqlset_cursor,
   load_option       IN VARCHAR2 := 'INSERT',
   update_option     IN VARCHAR2 := 'REPLACE',
   update_condition  IN VARCHAR2 :=  NULL,
   update_attributes IN VARCHAR2 :=  NULL,
   ignore_null       IN BOOLEAN  :=  TRUE,
   commit_rows       IN POSITIVE :=  NULL,
   sqlset_owner      IN VARCHAR2 := NULL);
```

Parameters

Table 195-26 LOAD_SQLSET Procedure Parameters

Parameter	Description
sqlset_name	Specifies the name of SQL tuning set to be loaded.
populate_cursor	Specifies the cursor reference to the SQL tuning set to be loaded.

Table 195-26 (Cont.) LOAD_SQLSET Procedure Parameters

Parameter	Description
load_option	Specifies which statements are loaded into the SQL tuning set. The possible values are: • INSERT (default) — Adds only new statements. • UPDATE — Updates existing the SQL statements and ignores any new statements. • MERGE — Inserts new statements and updates the information of the existing ones.
update_option	Specifies how existing SQL statements are updated. This parameter is considered only if load_option is specified with UPDATE or MERGE as an option. The possible values are: • REPLACE (default) — Updates the statement using the new statistics, bind list, object list, and so on. • ACCUMULATE — Combines attributes when possible (for example, statistics such as elapsed_time), and otherwise replaces the existing values (for example, module and action) with the provided values. The SQL statement attributes that can be accumulated are: elapsed_time, buffer_gets, direct_writes, disk_reads, row_processed, fetches, executions, end_of_fetch_count, stat_period and active_stat_period.
update_condition	Specifies when to perform the update. The procedure only performs the update when the specified condition is satisfied. The condition can refer to either the data source or destination. The condition must use the following prefixes to refer to attributes from the source or the destination: • OLD — Refers to statement attributes from the SQL tuning set (destination). • NEW — Refers to statement attributes from the input statements (source).
update_attributes	Specifies the list of SQL statement attributes to update during a merge or update. The possible values are: • NULL (default) — Specifies the content of the input cursor except the execution context. On other terms, it is equivalent to ALL without execution contexts such as module and action. • BASIC — Specifies statistics and binds only. • TYPICAL — Specifies BASIC with SQL plans (without row source statistics) and without an object reference list. • ALL — Specifies all attributes, including the execution context attributes such as module and action. • List of comma separated attribute names to update: – EXECUTION_CONTEXT – EXECUTION_STATISTICS – SQL_BINDS – SQL_PLAN – SQL_PLAN_STATISTICS (similar to SQL_PLAN with added row source statistics)
ignore_null	Specifies whether to update attributes when the new value is NULL. If TRUE, then the procedure does not update an attribute when the new value is NULL. That is, do not override with NULL values unless intentional.

Table 195-26 (Cont.) LOAD_SQLSET Procedure Parameters

Parameter	Description
commit_rows	Specifies whether to commit statements after DML. If a value is provided, then the load commits after each specified number of statements is inserted. If NULL is provided, then the load commits only once, at the end of the operation. Providing a value for this argument enables you to monitor the progress of a SQL tuning set load operation in the DBA_SQLSET views. The STATEMENT_COUNT value increases as new SQL statements are loaded.
sqlset_owner	Defines the owner of the SQL tuning set, or the current schema owner (or NULL for the current owner).

Exceptions

- This procedure returns an error when sqlset_name is invalid, or a corresponding SQL tuning set does not exist, or the populate_cursor is incorrect and cannot be executed.
- Exceptions are also raised when invalid filters are provided. Filters can be invalid either because they don't parse (for example, they refer to attributes not in sqlset_row), or because they violate the user's privileges.

Usage Notes

Rows in the input populate_cursor must be of type SQLSET_ROW.

Examples

In this example, you create and populate a SQL tuning set with all shared SQL area statements with an elapsed time of 5 seconds or more excluding statements that belong to SYS schema (to simulate an application user workload). You select all attributes of the SQL statements and load them in the tuning set using the default mode, which loads only new statements, since the SQL tuning set is empty.

```
-- create the tuning set
EXEC DBMS_SQLTUNE.CREATE_SQLSET('my_workload');
-- populate the tuning set from the shared SQL area
DECLARE
  cur DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
      FROM table(
        DBMS_SQLTUNE.SELECT_CURSOR_CACHE(
          'parsing_schema_name <> ''SYS'' AND elapsed_time > 5000000',
           NULL, NULL, NULL, NULL, 1, NULL,
          'ALL')) P;

  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload', populate_cursor =>
cur);
END;
/
```

Suppose now you wish to augment this information with what is stored in the workload repository (AWR). You populate the tuning set with `'ACCUMULATE'` as your `update_option` because it is assumed the cursors currently in the cache had aged out since the snapshot was taken.

You omit the `elapsed_time` filter because it is assumed that any statement captured in AWR is important, but still you throw away the `SYS`-parsed cursors to avoid recursive SQL.

```
DECLARE
  cur DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
      FROM table(
        DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(1,2,
                                                'parsing_schema_name <>
''SYS''',
                                                NULL, NULL,NULL,NULL,
                                                1,
                                                NULL,
                                                'ALL')) P;

  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name     => 'my_workload',
                           populate_cursor => cur,
                           Using DBMS_SQLTUNE
                           load_option => 'MERGE',
                           update_option => 'ACCUMULATE');
END;
```

The following example is a simple load that only inserts new statements from the workload repository, skipping existing ones (in the SQL tuning set). Note that `'INSERT'` is the default value for the `load_option` argument of the `LOAD_SQLSET` procedure.

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
  SELECT VALUE(P)
  FROM table(DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(1,2)) P;
  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name => 'my_workload', populate_cursor =>
cur);
END;
/
```

The next example demonstrates a load with `UPDATE` option. This updates statements that already exist in the SQL tuning set but does not add new ones. By default, old statistics are replaced by their new values.

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE) P;
```

```
  DBMS_SQLTUNE.LOAD_SQLSET(sqlset_name     => 'my_workload',
                           populate_cursor => cur,
                           load_option     => 'UPDATE');
END;
/
```

PACK_STGTAB_SQLPROF Procedure

This procedure copies profile data from the `SYS.` schema into the staging table.

> **See Also:**
>
> DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (
   profile_name          IN VARCHAR2 := '%',
   profile_category      IN VARCHAR2 := 'DEFAULT',
   staging_table_name    IN VARCHAR2,
   staging_schema_owner  IN VARCHAR2 := NULL);
```

Parameters

Table 195-27 PACK_STGTAB_SQLPROF Procedure Parameters

Parameter	Description
`profile_name`	The name of the profile to pack (% wildcards acceptable, case-sensitive)
`profile_category`	The category to pack profiles from (% wildcards acceptable, case-sensitive)
`staging_table_name`	The name of the table to use (case-insensitive unless double quoted). Required.
`staging_schema_owner`	The schema where the table resides, or `NULL` for current schema (case-insensitive unless double quoted)

Security Model

This procedures requires `ADMINISTER SQL MANAGEMENT OBJECT` privilege and `INSERT` privilege on the staging table.

Usage Notes

This function issues a `COMMIT` after packing each SQL profile. If an error is raised mid-execution, then clear the staging table by deleting its rows.

Examples

Put only those profiles in the `DEFAULT` category into the staging table. This corresponds to all profiles used by default on this system.

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (staging_table_name =>
'PROFILE_STGTAB');
```

This is another example where you put all profiles into the staging table. Note this moves profiles that are not currently being used by default but are in other categories, such as for testing purposes.

```
EXEC DBMS_SQLTUNE.PACK_STGTAB_SQLPROF (profile_category => '%', -
                                       staging_table_name => 'PROFILE_STGTAB');
```

PACK_STGTAB_SQLSET Procedure

This procedure copies one or more SQL tuning sets from their location in the `SYS` schema to a staging table created by the `CREATE_STGTAB_SQLSET` procedure.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.PACK_STGTAB_SQLSET (
   sqlset_name          IN VARCHAR2,
   sqlset_owner         IN VARCHAR2 := NULL,
   staging_table_name   IN VARCHAR2,
   staging_schema_owner IN VARCHAR2 := NULL,
   db_version           IN NUMBER   := NULL);
```

Examples

Put all SQL tuning sets on the database in the staging table:

```
BEGIN
  DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
      sqlset_name        => '%'
  ,   sqlset_owner       => '%'
  ,   staging_table_name => 'STGTAB_SQLSET');
END;
```

Put only those SQL tuning sets owned by the current user in the staging table:

```
BEGIN
  DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
      sqlset_name        => '%'
```

```
  ,   staging_table_name => 'STGTAB_SQLSET');
END;
```

Pack a specific SQL tuning set:

```
BEGIN
  DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
      sqlset_name         => 'my_workload'
  ,   staging_table_name  => 'STGTAB_SQLSET');
END;
```

Pack a second SQL tuning set:

```
BEGIN
  DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
      sqlset_name         => 'workload_subset'
  ,   staging_table_name  => 'STGTAB_SQLSET');
END;
```

Pack the STS `my_workload_subset` into a staging table `stgtab_sqlset` created for Oracle Database 11*g* Release 1 (11.2):

```
BEGIN
  DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
      sqlset_name          => 'workload_subset'
  ,   staging_table_name   => 'STGTAB_SQLSET'
  ,   db_version           => DBMS_SQLTUNE.STS_STGTAB_11_2_VERSION);
END;
```

REMAP_STGTAB_SQLPROF Procedure

This procedure changes the profile data values kept in the staging table prior to performing an unpack operation.

You can use this procedure to change the category of a profile. You can also use it to change the name of a profile if one already exists on the system with the same name.

> **See Also:**
>
> DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF (
  old_profile_name      IN VARCHAR2,
  new_profile_name      IN VARCHAR2 := NULL,
  new_profile_category  IN VARCHAR2 := NULL,
  staging_table_name    IN VARCHAR2,
  staging_schema_owner  IN VARCHAR2 := NULL);
```

Parameters

Table 195-28 REMAP_STGTAB_SQLPROF Procedure Parameters

Parameter	Description
old_profile_name	The name of the profile to target for a remap operation (case-sensitive)
new_profile_name	The new name of the profile, or NULL to remain the same (case-sensitive)
new_profile_category	The new category for the profile, or NULL to remain the same (case-sensitive)
staging_table_name	The name of the table on which to perform the remap operation (case-sensitive). Required.
staging_schema_owner	The schema where the table resides, or NULL for current schema (case-sensitive)

Security Model

This procedure requires the UPDATE privilege on the staging table.

Examples

Change the name of a profile before we unpack, to avoid conflicts

```
BEGIN
  DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF(
      old_profile_name    => :pname
  ,   new_profile_name    => 'IMP' || :pname
  ,   staging_table_name  => 'PROFILE_STGTAB');
END;
```

Change the SQL profile in the staging table to be 'TEST' category before we import it. This way users can test the profile on the new system before it is active.

```
BEGIN
  DBMS_SQLTUNE.REMAP_STGTAB_SQLPROF(
      old_profile_name     => :pname
  ,   new_profile_category => 'TEST'
  ,   staging_table_name   => 'PROFILE_STGTAB');
END;
```

REMAP_STGTAB_SQLSET Procedure

This procedure changes the tuning set names and owners in the staging table so that they can be unpacked with different values.

> **See Also:**
>
> DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REMAP_STGTAB_SQLSET (
   old_sqlset_name        IN VARCHAR2,
   old_sqlset_owner       IN VARCHAR2 := NULL,
   new_sqlset_name        IN VARCHAR2 := NULL,
   new_sqlset_owner       IN VARCHAR2 := NULL,
   staging_table_name     IN VARCHAR2,
   staging_schema_owner   IN VARCHAR2 := NULL
   old_con_dbid           IN NUMBER   := NULL,
   new_con_dbid           IN NUMBER   := NULL);
);
```

Parameters

The parameters are identical for the DBMS_SQLTUNE.REMAP_STGTAB_SQLSET and DBMS_SQLSET.REMAP_SQLSET procedures.

Table 195-29 REMAP_STGTAB_SQLSET and REMAP_SQLSET Procedure Parameters

Parameter	Description
old_sqlset_name	Specifies the name of the tuning set to target for a remap operation. Wildcard characters (%) are not supported.
old_sqlset_owner	Specifies the new name of the tuning set owner to target for a remap operation. NULL for current schema owner
new_sqlset_name	Specifies the new name for the tuning set, or NULL to keep the same tuning set name.
new_sqlset_owner	Specifies the new owner for the tuning set, or NULL to keep the same owner name.
staging_table_name	Specifies the name of the table on which to perform the remap operation. The value is case sensitive.
staging_schema_owner	Specifies the name of staging table owner, or NULL for the current schema owner. The value is case sensitive.
old_con_dbid	Specifies the old container DBID to be remapped to a new container DBID. Specify NULL to use the same container DBID. You must provide both old_con_dbid and new_con_dbid for the remap to succeed.
new_con_dbid	Specifies the new container DBID to replace with the old container DBID. Specify NULL to use the same container DBID. You must provide both old_con_dbid and new_con_dbid for the remap to succeed.

Usage Notes

Call this procedure multiple times to remap more than one tuning set name or owner. This procedure only handles one tuning set per call.

Examples

```
-- Change the name of an STS in the staging table before unpacking it.
BEGIN
  DBMS_SQLTUNE.REMAP_STGTAB_SQLSET(
      old_sqlset_name    =>
```

```
'my_workload'
  ,   old_sqlset_owner   => 'SH'
  ,   new_sqlset_name    =>
'imp_workload'
  ,   staging_table_name => 'STGTAB_SQLSET');

-- Change the owner of an STS in the staging table before unpacking it.
  DBMS_SQLTUNE.REMAP_STGTAB_SQLSET(
      old_sqlset_name    => 'imp_workload'
  ,   old_sqlset_owner   => 'SH'
  ,   new_sqlset_owner   => 'SYS'
  ,   staging_table_name => 'STGTAB_SQLSET');
END;
```

REMOVE_SQLSET_REFERENCE Procedure

This procedure deactivates a SQL tuning set to indicate that it is no longer used by the client.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REMOVE_SQLSET_REFERENCE (
   sqlset_name    IN  VARCHAR2,
   reference_id   IN  NUMBER,
   sqlset_owner   IN  VARCHAR2 := NULL,
   force_remove   IN  NUMBER   := 0);
```

Parameters

The parameters are identical for the `DBMS_SQLTUNE.REMOVE_SQLSET_REFERENCE` and `DBMS_SQLSET.REMOVE_REFERENCE` procedures.

Table 195-30 REMOVE_SQLSET_REFERENCE and REMOVE_REFERENCE Procedure Parameters

Parameter	Description
`sqlset_name`	Specifies the name of the SQL tuning set.
`reference_id`	Specifies the identifier of the reference to remove.
`sqlset_owner`	Specifies the owner of the SQL tuning set (or `NULL` for the current schema owner).
`force_remove`	Specifies whether references can be removed for other users (`1`) or whether they cannot be removed (`0`). Setting this parameter to `1` only takes effect when the user has the `ADMINISTER ANY SQL TUNING SET` privilege. Otherwise, the database only removes references owned by the user.

Examples

You can remove references on a given SQL tuning set when you finish using it and want to make it writable again. The following example removes the reference to `my_workload`:

```
EXEC DBMS_SQLTUNE.REMOVE_SQLSET_REFERENCE( -
                                sqlset_name   => 'my_workload', -
                                reference_id  => :rid,
                                sqlset_owner  => NULL,
                                force_remove  => 0);
```

To find all references to a given SQL tuning set, query the `DBA_SQLSET_REFERENCES` view.

REPORT_AUTO_TUNING_TASK Function

This function displays a report from the automatic tuning task.

This function reports on a range of task executions, whereas the REPORT_TUNING_TASK Function reports on a single execution. Note that this function is deprecated with Oracle Database 11*g* Release 2 (11.2) in favor of `DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK`.

> **See Also:**
>
> - DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group
> - REPORT_AUTO_TUNING_TASK Function

Syntax

```
DBMS_SQLTUNE.REPORT_AUTO_TUNING_TASK(
    begin_exec      IN VARCHAR2  := NULL,
    end_exec        IN VARCHAR2  := NULL,
    type            IN VARCHAR2  := TYPE_TEXT,
    level           IN VARCHAR2  := LEVEL_TYPICAL,
    section         IN VARCHAR2  := SECTION_ALL,
    object_id       IN NUMBER    := NULL,
    result_limit    IN NUMBER    := NULL)
  RETURN CLOB;
```

Parameters

Table 195-31 REPORT_AUTO_TUNING_TASK Function Parameters

Parameter	Description
`begin_exec`	Specifies the name of the execution from which to begin the report. `NULL` retrieves a report on the most recent run.
`end_exec`	Specifies the name of the execution at which to end the report. `NULL` retrieves a report on the most recent run.

Table 195-31 (Cont.) REPORT_AUTO_TUNING_TASK Function Parameters

Parameter	Description
type	Specifies the type of the report to produce. Possible values are TYPE_TEXT which produces a text report
level	Specifies the level of detail in the report: • LEVEL_BASIC: simple version of the report. Just show info about the actions taken by the advisor. • LEVEL_TYPICAL: show information about every statement analyzed, including requests not implemented. • LEVEL_ALL: highly detailed report level, also provides annotations about statements skipped over.
section	Limits the report to a single section (ALL for all sections): • SECTION_SUMMARY - summary information • SECTION_FINDINGS - tuning findings • SECTION_PLAN - explain plans • SECTION_INFORMATION - general information • SECTION_ERROR - statements with errors • SECTION_ALL - all statements
object_id	Specifies the advisor framework object ID that represents a single statement to restrict reporting to. Specify NULL for all statements. Only valid for reports that target a single execution.
result_limit	Specifies the maximum number of SQL statements to show in the report.

Return Values

A CLOB containing the desired report.

REPORT_SQL_DETAIL Function

This function builds a report for a specific SQLID. For each SQLID it gives various statistics and details as obtained from the V$ views and AWR.

> **See Also:**
>
> DBMS_SQLTUNE SQL Performance Reporting Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REPORT_SQL_DETAIL (
   sql_id                  IN  VARCHAR2   DEFAULT NULL,
   sql_plan_hash_value     IN  NUMBER     DEFAULT NULL,
   start_time              IN  DATE       DEFAULT NULL,
   duration                IN  NUMBER     DEFAULT NULL,
   inst_id                 IN  NUMBER     DEFAULT NULL,
   dbid                    IN  NUMBER     DEFAULT NULL,
   event_detail            IN  VARCHAR2   DEFAULT 'YES',
   bucket_max_count        IN  NUMBER     DEFAULT 128,
```

```
   bucket_interval          IN  NUMBER     DEFAULT NULL,
   top_n                    IN  NUMBER     DEFAULT 10,
   report_level             IN  VARCHAR2   DEFAULT 'TYPICAL',
   type                     IN  VARCHAR2   DEFAULT 'ACTIVE',
   data_source              IN  VARCHAR2   DEFAULT 'AUTO',
   end_time                 IN  DATE       DEFAULT NULL,
   duration_stats           IN  NUMBER     DEFAULT NULL,
   con_name                 IN  VARCHAR2   DEFAULT NULL)
RETURN CLOB;
```

Parameters

Table 195-32 REPORT_SQL_DETAIL Function Parameters

Parameter	Description
sql_id	SQLID for which monitoring information should be displayed. If NULL (the default), display statistics for the SQLID of the last SQL statement executed in the current session.
sql_plan_hash_value	Displays SQL statistics and details for a specific plan_hash_value. If NULL (default), displays statistics and details for all plans of the SQL_ID.
start_time	If specified, shows SQL activity (from GV$ACTIVE_SESSION_HISTORY) starting at this time. On Oracle RAC, the minimum start_time is the earliest sample_time of the in-memory ASH buffers across all instances. If NULL (default), one hour before the current time.
duration	Duration of activity in seconds for the report. If NULL (default) uses a value of 1 hour.
inst_id	Target instance to get SQL details from. If NULL, uses data from all instances. If 0 or -1, uses current instance.
dbid	DBID from which to get SQL details. If NULL, uses current DBID.
event_detail	When set to 'NO', the activity is aggregated by wait_class only. Use 'YES' (the default) to aggregate by (wait_class, event_name).
bucket_max_count	If specified, this should be the maximum number of histogram buckets created in the report. If not specified, a value of 128 is used.
bucket_interval	If specified, this represents the exact time interval in seconds, of all histogram buckets. If specified, bucket_max_count is ignored.
top_n	Controls the number of entries to display per dimension in the top dimensions section. If not specified, a default value of 10 is used.

Table 195-32 (Cont.) REPORT_SQL_DETAIL Function Parameters

Parameter	Description
report_level	Level of detail for the report, either 'BASIC', 'TYPICAL' or 'ALL'. Default assumes 'TYPICAL'. Their meanings are explained below. In addition, individual report sections can also be enabled or disabled by using a +/- *section_name*. Several sections are defined: • 'TOP'- Show top values for the ASH dimensions for a SQL statement; ON by default • 'SPM'- Show existing plan baselines for a SQL statement; OFF by default • 'MISMATCH'- Show reasons for creating new child cursors (sharing criteria violations); OFF by default. • 'STATS'- Show SQL execution statistics per plan from GV$SQLAREA_PLAN_HASH; ON by default • 'ACTIVITY' - Show top activity from ASH for each plan of a SQL statement; ON by default • 'ACTIVITY_ALL' - Show top activity from ASH for each line of the plan for a SQL statement; OFF by default • 'HISTOGRAM' - Show activity histogram for each plan of a SQL statement (plan time line histogram); ON by default • 'SESSIONS' - Show activity for top sessions for each plan of a SQL statement; OFF by default • 'MONITOR' - Show show one monitored SQL execution per execution plan; ON by default • 'XPLAN' - Show execution plans; ON by default • 'BINDS' - show captured bind data; ON by default In addition, SQL text can be specified at different levels: • -SQL_TEXT - No SQL text in report • +SQL_TEXT - OK with partial SQL text up to the first 2000 chars as stored in GV$SQL_MONITOR • -SQL_FULLTEXT - No full SQL text (+SQL_TEXT) • +SQL_FULLTEXT - Show full SQL text (default value) The meanings of the three top-level report levels are: • NONE - minimum possible • BASIC - SQL_TEXT+STATS+ACTIVITY+HISTOGRAM • TYPICAL - SQL_FULLTEXT+TOP+STATS+ACTIVITY+HISTOGRAM+XPLAN+MONITOR • ALL - everything Only one of these 4 levels can be specified and, if it is, it has to be at the start of the REPORT_LEVEL string
type	Report format: 'ACTIVE' by default. Can also be 'XML' (see Usage Notes).
data_source	Determines the data source of SQL data based on one of the following values: • MEMORY: The data source is GV$ view • DISK: The data source is DBA_HIST_* view • AUTO: Automatically determines the data source based on the time frame (default)
end_time	If specified, shows SQL activity from start_time to end_time. If NULL (default), shows SQL activity for systimestamp.

Table 195-32 (Cont.) REPORT_SQL_DETAIL Function Parameters

Parameter	Description
duration_stats	Duration of additional SQL execution statistics from AWR (in hours), for the report. If NULL (default), then the duration of 24 hours is considered.
con_name	Name of the multitenant container database (CDB).

Security Model

The invoker needs the EXECUTE privilege on the DBMS_XPLAN package.

Return Values

A CLOB containing the desired report.

Usage Notes

- ACTIVE reports have a rich, interactive user interface similar to Enterprise Manager while not requiring any EM installation. The report file built is in HTML format, so it can be interpreted by most modern browsers. The code powering the active report is downloaded transparently by the web browser when the report is first viewed, hence viewing it requires outside connectivity.
- The invoker needs the SELECT or READ privilege on the following views:
  - V$SESSION
  - DBA_ADVISOR_FINDINGS
  - V$DATABASE
  - GV$ASH_INFO
  - GV$ACTIVE_SESSION_HISTORY
  - GV$SQLAREA_PLAN_HASH
  - GV$SQL
  - DBA_HIST_SNAPSHOT
  - DBA_HIST_WR_CONTROL
  - DBA_HIST_ACTIVE_SESS_HISTORY
  - DBA_HIST_SQLSTAT
  - DBA_HIST_SQL_BIND_METADATA
  - DBA_HIST_SQLTEXT
  - DBA_SQL_PLAN_BASELINES
  - DBA_SQL_PROFILES
  - DBA_ADVISOR_TASKS
  - DBA_SERVICES
  - DBA_USERS
  - DBA_OBJECTS

- DBA_PROCEDURES

REPORT_SQL_MONITOR Function

This function builds a report (text, simple HTML, active HTML, XML) for the monitoring information collected on behalf of the targeted statement execution.

> **See Also:**
>
> Real-Time SQL Monitoring for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REPORT_SQL_MONITOR(
   sql_id                    IN VARCHAR2  DEFAULT  NULL,
   dbop_name                 IN VARCHAR2  DEFAULT  NULL,
   dbop_exec_id              IN NUMBER    DEFAULT  NULL,
   session_id                IN NUMBER    DEFAULT  NULL,
   session_serial            IN NUMBER    DEFAULT  NULL,
   sql_exec_start            IN DATE      DEFAULT  NULL,
   sql_exec_id               IN NUMBER    DEFAULT  NULL,
   inst_id                   IN NUMBER    DEFAULT  NULL,
   start_time_filter         IN DATE      DEFAULT  NULL,
   end_time_filter           IN DATE      DEFAULT  NULL,
   instance_id_filter        IN NUMBER    DEFAULT  NULL,
   parallel_filter           IN VARCHAR2  DEFAULT  NULL,
   plan_line_filter          IN NUMBER    DEFAULT  NULL,
   event_detail              IN VARCHAR2  DEFAULT  'YES',
   bucket_max_count          IN NUMBER    DEFAULT  128,
   bucket_interval           IN NUMBER    DEFAULT  NULL,
   base_path                 IN VARCHAR2  DEFAULT  NULL,
   last_refresh_time         IN DATE      DEFAULT  NULL,
   report_level              IN VARCHAR2  DEFAULT 'TYPICAL',
   type                      IN VARCHAR2  DEFAULT 'TEXT',
   sql_plan_hash_value       IN NUMBER    DEFAULT  NULL,
   con_name                  IN VARCHAR2  DEFAULT  NULL,
   report_id                 IN NUMBER    DEFAULT  NULL)
 RETURN CLOB;
```

Parameters

Table 195-33 REPORT_SQL_MONITOR Function Parameters

Parameter	Description
sql_id	SQL_ID for which monitoring information should be displayed. Use NULL (the default) to report on the last statement monitored by Oracle.
dbop_name	DBOP_NAME for which monitoring information of the composite database operation is displayed.
dbop_exec_id	Execution ID for the composite database operation for which monitoring information is displayed.

Table 195-33 (Cont.) REPORT_SQL_MONITOR Function Parameters

Parameter	Description
session_id	If not NULL, this parameters targets only the sub-set of statements executed by the specified session. Default is NULL. Use SYS_CONTEXT('SID') for current session.
session_serial	In addition to the session_id parameter, one can also specify its session serial to ensure that the desired session incarnation is targeted. This parameter is ignored when session_id is NULL.
sql_exec_start	This parameter, along with sql_exec_id, is only applicable when sql_id is also specified. Jointly, they can be used to display monitoring information associated to any execution of the statement identified by sql_id, assuming that this statement was monitored. When NULL (the default), the last monitored execution of SQL sql_id is shown.
sql_exec_id	This parameter, along with sql_exec_start, is only applicable when sql_id is also specified. Jointly, they can be used to display monitoring information associated to any execution of the statement identified by sql_id, assuming that this statement was monitored. When NULL (the default), the last monitored execution of SQL sql_id is shown.
inst_id	Only considers statements started on the specified instance. Use -1 to target the login instance. NULL (default) targets all instances.
start_time_filter	If not NULL, the report considers only the activity (from GV$ACTIVE_SESSION_HISTORY) recorded after the specified date. If NULL, the reported activity starts when the execution of the targeted SQL statement has started.
end_time_filter	If not NULL, the report shows only the activity (from GV$ACTIVE_SESSION_HISTORY) collected before the date end_time_filter. If NULL, the reported activity ends when the targeted SQL statement execution has ended or is the current time if the statement is still executing.
instance_id_filter	Only applies when the execution runs parallel across multiple Oracle Real Application Cluster (Oracle RAC) instances. This parameter allows to only report the activity of the specified instance. Use a NULL value (the default) to include the activity on all instances where the parallel query was executed.

Table 195-33 (Cont.) REPORT_SQL_MONITOR Function Parameters

Parameter	Description
parallel_filter	Applies only to parallel execution and allows reporting the activity of only a subset of the processes involved in the parallel execution (Query Coordinator and/or Parallel eXecution servers). The value of this parameter can be: • NULL to target all processes • [qc][servers(<svr_grp>[,] <svr_set>[,] <srv_num>)]: 'qc' stands for query coordinator and servers() stipulate which PX servers to consider. The following examples show how to target a subset of the parallel processes: • qc: targets only the query coordinator • servers(1): targets all parallel execution servers in group number 1. Note that statement running parallel have one main server group (group number 1) plus one additional group for each nested sub-query running parallel. • servers(,2): targets all parallel execution servers from any group but only running in set 1 of each group (each group has at most two set of parallel execution servers) • servers(1,1): consider only group 1, set 1 • servers(1,2,4): consider only group 1, set 2, server number 4. This reports for a single parallel server process • qc servers(1,2,4): same as above by also including the query coordinator
event_detail	When value is 'YES' (the default), reported activity from GV$ACTIVE_SESSION_HISTORY is aggregated by (wait_class, event_name). Use 'NO' to only aggregate by wait_class.
bucket_max_count	If specified, this should be the maximum number of histogram buckets created in the report
bucket_interval	If specified, this represents the exact time interval in seconds, of all histogram buckets. If specified, bucket_max_count is ignored.
base_path	URL path for flex HTML resources since flex HTML format is required to access external files (java scripts and the flash SWF file itself)
last_refresh_time	If not NULL (default is NULL), the time when the report was last retrieved (see SYSDATE attribute of the report tag). Use this option to display the report of a running query, and when the report is refreshed on a regular basis. This optimizes the size of the report since only the new or changed information is returned. In particular, the following are optimized: • SQL text is not returned when this option is specified • activity histogram starts at the bucket that intersect at that time. The entire content of the bucket is returned, even if last_refresh_time is after the start of that bucket

Table 195-33 (Cont.) REPORT_SQL_MONITOR Function Parameters

Parameter	Description
report_level	Level of detail for the report: 'NONE', 'BASIC', 'TYPICAL' or 'ALL'. Default assumes 'TYPICAL'. In addition, individual report sections can also be enabled or disabled by using a +/- *section_name*. Several sections are defined: • 'XPLAN'- Show explain plan; ON by default • 'PLAN'- Show plan monitoring statistics; ON by default • 'SESSIONS'- Show session details. Applies only to parallel queries; ON by default • 'INSTANCE'- Show instance details. Applies only to parallel and cross instance; ON by default • 'PARALLEL'- An umbrella parameter for specifying sessions+instance details • 'ACTIVITY' - Show activity summary at global level, plan line level and session or instance level (if applicable); ON by default • 'BINDS' - Show bind information when available; ON by default • 'METRICS' - Show metric data (CPU, I/Os, ...) over time; ON by default • 'ACTIVITY_HISTOGRAM' - Show an histogram of the overall query activity; ON by default • 'PLAN_HISTOGRAM' - Show activity histogram at plan line level; OFF by default • 'OTHER' - Other info; ON by default In addition, SQL text can be specified at different levels: • SQL_TEXT - No SQL text in report • +SQL_TEXT - OK with partial SQL text up to the first 2000 chars as stored in GV$SQL_MONITOR • -SQL_FULLTEXT - No full SQL text (+SQL_TEXT) • +SQL_FULLTEXT - Show full SQL text (default value)
report_level (contd.)	The meanings of the three top-level report levels are: • NONE - minimum possible • +BASIC - SQL_TEXT-PLAN-XPLAN-SESSIONS-INSTANCE-ACTIVITY_HISTOGRAM-PLAN_HISTOGRAM-METRICS • TYPICAL - everything but PLAN_HISTOGRAM • ALL - everything Only one of these 4 levels can be specified and, if it is, it has to be at the start of the REPORT_LEVEL string
type	Report format, 'TEXT' by default. Can be 'TEXT', 'HTML', 'XML' or 'ACTIVE' (see Usage Notes).
sql_plan_hash_value	Target only those SQL executions with the specified plan_hash_value. Default is NULL.
con_name	Name of the multitenant container database (CDB).
report_id	ID of the report in auto-report repository. Report IDs can be found in DBA_HIST_REPORTS.

Return Values

A CLOB containing the desired report.

Usage Notes

- The target SQL statement for this report can be:
  - The most recent SQL statement monitored by Oracle Database. This is the default behavior, so there is no need to specify any parameter.
  - The most recent SQL statement executed by a specific session and monitored by Oracle. The session is identified by its session id and optionally it serial number. For example, use `session_id =>` for the current session or `session_id => 20, session_serial => 103` for session ID 20, serial number 103.
  - The most recent execution of a specific statement identified by its `sql_id`.
  - A specific execution of a SQL statement identified by its execution key (`sql_id`, `sql_exec_start` and `sql_exec_id`).
- This report produces performance data exposed by several fixed views, listed below. For this reason, the invoker of the report function must have privilege to select data from these fixed views (such as the `SELECT_CATALOG` role).
  - `GV$SQL_MONITOR`
  - `GV$SQL_PLAN_MONITOR`
  - `GV$SQL_PLAN`
  - `GV$ACTIVE_SESSION_HISTORY`
  - `GV$SESSION_LONGOPS`
  - `GV$SQL`
- The `bucket_max_count` and `bucket_interval` parameters control the activity histogram.

  By default, the maximum number of buckets is set to 128. The database derives the `bucket_interval` value based on this count. The `bucket_interval` (value is in seconds) is computed such that it is the smallest possible power of 2 value (starting at 1 second) without exceeding the maximum number of buckets. For example, if the query has executed for 600 seconds, then the database selects a `bucket_interval` of 8 seconds (a power of two). The database chooses the value of 8 because 600/8 = 74, which is less than 128 buckets maximum. Smaller than 8 seconds would be 4 seconds, which would lead to more buckets than the 128 maximum. If `bucket_interval` is specified, then the database uses the specified value instead of deriving it from `bucket_max_count`.
- `ACTIVE` reports have a rich, interactive user interface similar to Enterprise Manager, while not requiring any EM installation.

  The report file is in HTML format. The code powering the active report is downloaded transparently by the web browser when the report is first viewed. Therefore, viewing the report requires outside connectivity.

> **See Also:**
>
> *Oracle Database SQL Tuning Guide* for more information about SQL real-time monitoring.

REPORT_SQL_MONITOR_LIST Function

This function builds a report for all or a subset of statements monitored by Oracle Database. For each statement, the subprogram gives key information and associated global statistics.

Use the REPORT_SQL_MONITOR Function to get detailed monitoring information for a single SQL statement.

> **See Also:**
>
> Real-Time SQL Monitoring for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REPORT_SQL_MONITOR_LIST(
   sql_id                    IN VARCHAR2  DEFAULT  NULL,
   session_id                IN NUMBER    DEFAULT  NULL,
   session_serial            IN NUMBER    DEFAULT  NULL,
   inst_id                   IN NUMBER    DEFAULT  NULL,
   active_since_date         IN DATE      DEFAULT  NULL,
   active_since_sec          IN NUMBER    DEFAULT  NULL,
   active_before_date        IN DATE      DEFAULT  NULL,
   last_refresh_time         IN DATE      DEFAULT  NULL,
   dbop_name                 IN VARCHAR2  DEFAULT  NULL,
   monitor_type              IN NUMBER    DEFAULT  MONITOR_TYPE_ALL,
   max_sqltext_length        IN NUMBER    DEFAULT  NULL,
   top_n_count               IN NUMBER    DEFAULT  NULL,
   top_n_rankby              IN VARCHAR2  DEFAULT  'LAST_ACTIVE_TIME',
   report_level              IN VARCHAR2  DEFAULT  'TYPICAL',
   auto_refresh              IN NUMBER    DEFAULT  NULL,
   base_path                 IN VARCHAR2  DEFAULT  NULL,
   type                      IN VARCHAR2  DEFAULT 'TEXT',
   con_name                  IN VARCHAR2  DEFAULT  NULL,
   top_n_detail_count        IN NUMBER    DEFAULT  NULL)
 RETURN CLOB;
```

Parameters

Table 195-34 REPORT_SQL_MONITOR_LIST Function Parameters

Parameter	Description
sql_id	SQL_ID for which monitoring information should be displayed. Use NULL (the default) to report on the last statement monitored by Oracle.
session_id	If not NULL, then this parameter targets only the subset of statements executed by the specified session. Default is NULL. Use -1 or SYS_CONTEXT('SID') for current session.
session_serial	In addition to the session_id parameter, you can also specify its session serial to ensure that the desired session incarnation is targeted. This parameter is ignored when session_id is NULL.

Table 195-34 (Cont.) REPORT_SQL_MONITOR_LIST Function Parameters

Parameter	Description
inst_id	Only considers statements started on the specified instance. Use -1 to target the login instance. NULL (default) targets all instances.
active_since_date	If not NULL (default), returns only monitored statements active since the specified time. This includes all statements that are still executing along with all statements that have completed their execution after the specified date and time.
active_since_sec	Same as active_since_date but with the date specified relative to the current SYSDATE minus a specified number of seconds. For example, use 3600 to apply a limit of 1 hour.
active_before_date	If not NULL (default), returns only monitored statements that have been active before the specified date and time.
last_refresh_time	If not NULL (default), the date and time when the list report was last retrieved. This optimizes the case where an application shows the list and refreshes the report on a regular basis (such as once every 5 seconds). In this case, the report shows detail about the execution of monitored queries that active since the specified last_refresh_time. For other queries, the report returns the execution key (sql_id, sql_exec_start, sql_exec_id). For queries with a first refresh time after the specified date, the function returns only the SQL execution key and statistics.
dbop_name	DB operation name. Specify NULL to display all the monitored DB operations.
monitor_type	Type of the SQL Monitor operation. Specify one of the following values: • MONITOR_TYPE_SQL - Returns only SQL statements • MONITOR_TYPE_DBOP - Returns only database operations • MONITOR_TYPE_ALL - Returns SQL statements as well as database operations
max_sqltext_length	Maximum length of the SQL text. Default is NULL (no limit).
top_n_count	Limits the number of top-N SQL statements that need to be included in the report.
top_n_rankby	Specifies the attribute to rank the SQL statements. Specify this value when top_n_count value is not NULL. The ranking of an SQL statement is done based on one of the following values: • LAST_ACTIVE_TIME - Last active date and time (top N most recent) • DURATION - Total duration of execution • DB_TIME - DB time used • CPU_TIME - CPU time used • IO_REQUESTS - Number of I/O requests • IO_BYTES - Number of I/O bytes
report_level	Level of detail for the report. The level is one of the following: • BASIC - SQL text up to 200 characters • TYPICAL - include full SQL text assuming that cursor has not aged out, in which case the SQL text is included up to 2000 characters • ALL - currently the same as TYPICAL
auto_refresh	Currently non-operational, reserved for future use.
base_path	URL path for flex HTML resources because flex HTML format is required to access external files (java scripts and the flash SWF file itself).

Table 195-34 (Cont.) REPORT_SQL_MONITOR_LIST Function Parameters

Parameter	Description
type	Report format: TEXT (default), HTML, or XML.
con_name	Name of the multitenant container database (CDB)
top_n_detail_count	Limits the number of top-N SQL statements for which the SQL monitor details need to be included in the report.

Return Values

A report for the list of SQL statements that have been monitored. The report type is text, XML, or HTML.

Usage Notes

You must have the privilege to access the following fixed views: GV$SQL_MONITOR and GV$SQL.

> **See Also:**
>
> *Oracle Database SQL Tuning Guide* for more information about SQL real-time monitoring.

REPORT_TUNING_TASK Function

This function displays the results of a tuning task. By default the report is in text format.

> **See Also:**
>
> DBMS_SQLTUNE SQL Performance Reporting Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REPORT_TUNING_TASK(
   task_name        IN   VARCHAR2,
   type             IN   VARCHAR2   := 'TEXT',
   level            IN   VARCHAR2   := 'TYPICAL',
   section          IN   VARCHAR2   := ALL,
   object_id        IN   NUMBER     := NULL,
   result_limit     IN   NUMBER     := NULL,
   owner_name       IN   VARCHAR2   := NULL,
   execution_name   IN   VARCHAR2   := NULL,
   database_link_to IN   VARCHAR2   := NULL)
RETURN CLOB;
```

Parameters

Table 195-35 REPORT_TUNING_TASK Function Parameters

Parameter	Description
task_name	Name of the tuning task.
type	Type of the report to produce. Possible values are TEXT which produces a text report.
level	Level of detail in the report: • BASIC: simple version of the report. Just show info about the actions taken by the advisor. • TYPICAL: show information about every statement analyzed, including requests not implemented. • ALL: highly detailed report level, also provides annotations about statements skipped over.
section	Section of the report to include. You can limit the report to any of the following single sections (ALL for all sections): • SUMMARY - Summary information • FINDINGS - Tuning findings • PLAN - Explain plans • INFORMATION - General information • ERROR - Statements with errors • ALL - All statements
object_id	Advisor framework object ID that represents a single statement to restrict reporting to. NULL for all statements. Only valid for reports that target a single execution.
result_limit	Maximum number of SQL statements to show in the report.
owner_name	Owner of the relevant tuning task. The default is the current schema owner.
execution_name	Name of the task execution to use. If NULL, then the function generates the report for the last task execution.
database_link_to	Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri: `CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY password USING 'inst1';`

Return Values

A `CLOB` containing the desired report.

Examples

```
-- Display the report for a single statement.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:stmt_task)
FROM   DUAL;

-- Display the summary for a SQL tuning set.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:sts_task, 'TEXT', 'TYPICAL',
'SUMMARY')
FROM   DUAL;

-- Display the findings for a specific statement.
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK(:sts_task, 'TEXT',
'TYPICAL','FINDINGS', 5)
FROM   DUAL;
```

REPORT_TUNING_TASK_XML Function

This function displays an XML report of a tuning task.

> **See Also:**
>
> DBMS_SQLTUNE SQL Performance Reporting Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.REPORT_TUNING_TASK_LIST_XML(
   task_name        IN   VARCHAR2   := NULL,
   level            IN   VARCHAR2   := LEVEL_TYPICAL,
   section          IN   VARCHAR2   := SECTION_ALL,
   object_id        IN   NUMBER     := NULL,
   result_limit     IN   NUMBER     := 160,
   owner_name       IN   VARCHAR2   := NULL,
   execution_name   IN   VARCHAR2   := NULL,
   autotune_period  IN   NUMBER     := NULL,
   report_tag       IN   VARCHAR2   := NULL)
RETURN XMLTYPE;
```

Parameters

Table 195-36 REPORT_TUNING_TASK_XML Function Parameters

Parameter	Description
`task_name`	Name of the tuning task.

Table 195-36 (Cont.) REPORT_TUNING_TASK_XML Function Parameters

Parameter	Description
level	Level of detail in the report: • BASIC: simple version of the report. Just show info about the actions taken by the advisor. • TYPICAL: show information about every statement analyzed, including requests not implemented. • ALL: highly detailed report level, also provides annotations about statements skipped over.
section	Section of the report to include. You can limit the report to any of the following single sections (ALL for all sections): • SUMMARY - Summary information • ALL - All statements
object_id	Advisor framework object ID that represents a single statement to restrict reporting to. NULL for all statements. Only valid for reports that target a single execution.
result_limit	The number of statements in a SQL tuning set or snapshot range for which the report is generated. The default is 160 (20 statements * 8 categories). The categories are as follows: • Profile • Index • Restructure SQL • Alternate plan • Statistics • Errors • Information • No findings
owner_name	Owner of the relevant tuning task. The default is the current schema owner.
execution_name	Name of the task execution to use. If NULL, then the function generates the report for the most recent task execution.
autotune_period	The time period for the automatic SQL tuning. This setting applies only to the automatic SQL Tuning Advisor task. Possible values are as follows: • Null or negative value (default) — All or full. The result includes all task executions. • 0 — Result of the current or most recent task execution. • 1 — Result for the most recent 24-hour period. • 7 — Result for the most recent 7-day period. The procedure interprets any other value as the time of the most recent task execution minus the value of this argument.
report_tag	The name of the root XML tag. By default, the tag is the report reference generated by the reporting framework.

Return Values

A CLOB containing the desired report.

RESET_TUNING_TASK Procedure

This procedure is called on a tuning task that is not currently executing to prepare it for re-execution.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.RESET_TUNING_TASK(
 task_name          IN VARCHAR2);
```

Parameters

Table 195-37 RESET_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	The name of the tuning task to reset

Examples

```
-- reset and re-execute a task
EXEC DBMS_SQLTUNE.RESET_TUNING_TASK(:sts_task);

-- re-execute the task
EXEC DBMS_SQLTUNE.EXECUTE_TUNING_TASK(:sts_task);
```

RESUME_TUNING_TASK Procedure

This procedure resumes a previously interrupted task that was created to process a SQL tuning set.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.RESUME_TUNING_TASK(
 task_name          IN VARCHAR2,
 basic_filter       IN VARCHAR2 := NULL);
```

Parameters

Table 195-38 RESUME_TUNING_TASK Procedure Parameters

Parameter	Description
task_name	The name of the tuning task to resume.
basic_filter	A SQL predicate to filter the SQL from the SQL tuning set. Note that this filter is applied in conjunction with the parameter basic_filteri when calling CREATE_TUNING_TASK Functions.

Usage Notes

Resuming a single SQL tuning task (a task that was created to tune a single SQL statement as compared to a SQL tuning set) is not supported.

Examples

```
-- Interrupt the task
EXEC DBMS_SQLTUNE.INTERRUPT_TUNING_TASK(:conc_task);

-- Once a task is interrupted, we can elect to reset it, resume it, or check
-- out its results and then decide.  For this example we will just resume.

EXEC DBMS_SQLTUNE.RESUME_TUNING_TASK(:conc_task);
```

SCHEDULE_TUNING_TASK Function

This function creates a tuning task for a single SQL statement and schedules a DBMS_SCHEDULER job to execute the tuning task. One form of the function finds the information about the statement to be tuned in the shared SQL area, whereas the other finds the information in AWR.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

Shared SQL Area Format:

```
DBMS_SQLTUNE.SCHEDULE_TUNING_TASK(
  sql_id          IN VARCHAR2,
  plan_hash_value IN NUMBER                    := NULL,
  start_date      IN TIMESTAMP WITH TIME ZONE  := NULL,
  scope           IN VARCHAR2                  := SCOPE_COMPREHENSIVE,
  time_limit      IN NUMBER                    := TIME_LIMIT_DEFAULT,
  task_name       IN VARCHAR2                  := NULL,
  description     IN VARCHAR2                  := NULL,
```

```
  con_name        IN VARCHAR2                 := NULL)
RETURN VARCHAR2;
```

AWR Format:

```
DBMS_SQLTUNE.SCHEDULE_TUNING_TASK(
  begin_snap      IN NUMBER,
  end_snap        IN NUMBER,
  sql_id          IN VARCHAR2,
  plan_hash_value IN NUMBER                   := NULL,
  start_date      IN TIMESTAMP WITH TIME ZONE := NULL,
  scope           IN VARCHAR2                 := SCOPE_COMPREHENSIVE,
  time_limit      IN NUMBER                   := TIME_LIMIT_DEFAULT,
  task_name       IN VARCHAR2                 := NULL,
  description     IN VARCHAR2                 := NULL,
  con_name        IN VARCHAR2                 := NULL,
  dbid            IN NUMBER                   := NULL)
RETURN VARCHAR2;
```

Parameters

Table 195-39 SCHEDULE_TUNING_TASK Function Parameters

Parameter	Description
begin_snap	The beginning snapshot identifier. The range is exclusive, which means that SQL statements in this snapshot ID are not included.
end_snap	The end snapshot identifier. The range is inclusive, which means that SQL statements in this snapshot ID are included.
sql_id	The SQL ID of the statement to be tuned.
plan_hash_value	The plan hash value of the statement to be tuned. For example, the tuning job fetches captured binds for this SQL plan.
start_date	The date on which the schedule becomes valid. If null, then SQL Tuning Advisor immediately executes the task.
scope	The scope of the tuning job: limited, or comprehensive.
time_limit	The maximum duration in seconds for the SQL tuning session.
task_name	Optional SQL tuning task name.
description	Description of the SQL tuning session. The description can contain a maximum of 256 characters.
con_name	The container from which SQL Tuning Advisor accesses the SQL statement information.
dbid	DBID for imported or PDB-level AWR data. If NULL, then the current database DBID is used.

Security Model

The caller must possess the CREATE JOB privilege for the job.

Return Values

A SQL tuning task name that is unique for each user. Multiple users can assign the same name to their advisor tasks.

Usage Notes

- The task is scheduled only once.
- The name of the scheduler job is created as follows: `sqltune_job_taskid_orahash(systimestamp)`.

SCRIPT_TUNING_TASK Function

This function creates a SQL*Plus script which can then be executed to implement a set of SQL Tuning Advisor recommendations.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Advisor Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SCRIPT_TUNING_TASK(
  task_name         IN VARCHAR2,
  rec_type          IN VARCHAR2  := REC_TYPE_ALL,
  object_id         IN NUMBER    := NULL,
  result_limit      IN NUMBER    := NULL,
  owner_name        IN VARCHAR2  := NULL,
  execution_name    IN VARCHAR2  := NULL,
  database_link_to  IN VARCHAR2  := NULL)
 RETURN CLOB;
```

Parameters

Table 195-40 SCRIPT_TUNING_TASK Function Parameters

Parameter	Description
`task_name`	Name of the tuning task for which to apply a script.
`rec_type`	Filter the script by types of recommendations to include. You can use any subset of the following values, separated by commas: 'ALL: ''PROFILES' ''STATISTICS' ''INDEXES'. For example, a script with profiles and statistics would use the filter 'PROFILES,STATISTICS'.
`object_id`	Optionally filters by a single object ID.
`result_limit`	Optionally shows commands for only top *n* SQL (ordered by `object_id` and ignored if an `object_id` is also specified).
`owner_name`	Owner of the relevant tuning task. Defaults to the current schema owner.
`excution_name`	Name of the task execution to use. If `NULL`, the script is generated for the last task execution.

Table 195-40 (Cont.) SCRIPT_TUNING_TASK Function Parameters

Parameter	Description
database_link_to	Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri: CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY *password* USING 'inst1';

Return Values

Returns a script in the form of a CLOB.

Usage Notes

- After the script is returned, check it before executing it.
- Wrap with a call to DBMS_ADVISOR.CREATE_FILE to put it into a file.

Examples

```
SET LINESIZE 140

-- Get a script for all actions recommended by the task.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task) FROM DUAL;

-- Get a script of only the sql profiles we should create.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task, 'PROFILES') FROM DUAL;

-- Get a script of only stale / missing stats
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:stmt_task, 'STATISTICS') FROM DUAL;

-- Get a script with recommendations about only one SQL statement when we have
-- tuned an entire STS.
SELECT DBMS_SQLTUNE.SCRIPT_TUNING_TASK(:sts_task, 'ALL', 5) FROM DUAL;
```

SELECT_CURSOR_CACHE Function

This function collects SQL statements from the shared SQL area.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SELECT_CURSOR_CACHE (
  basic_filter         IN   VARCHAR2 := NULL,
  object_filter        IN   VARCHAR2 := NULL,
  ranking_measure1     IN   VARCHAR2 := NULL,
  ranking_measure2     IN   VARCHAR2 := NULL,
  ranking_measure3     IN   VARCHAR2 := NULL,
  result_percentage    IN   NUMBER   := 1,
  result_limit         IN   NUMBER   := NULL,
  attribute_list       IN   VARCHAR2 := NULL,
  recursive_sql        IN   VARCHAR2 := HAS_RECURSIVE_SQL)
 RETURN sys.sqlset PIPELINED;
```

Parameters

Table 195-41 SELECT_CURSOR_CACHE Function Parameters

Parameter	Description
`basic_filter`	Specifies the SQL predicate that filters the SQL from the shared SQL area defined on attributes of the `SQLSET_ROW`. If `basic_filter` is not set by the caller, then the subprogram captures only statements of the type `CREATE TABLE`, `INSERT`, `SELECT`, `UPDATE`, `DELETE`, and `MERGE`.
`object_filter`	Currently not supported.
`ranking_measure(n)`	Defines an `ORDER BY` clause on the selected SQL.
`result_percentage`	Specifies a filter that picks the top *n*% according to the supplied ranking measure. The value applies only if one ranking measure is supplied.
`result_limit`	Defines the top limit SQL from the filtered source ranked by the ranking measure.

Table 195-41 (Cont.) SELECT_CURSOR_CACHE Function Parameters

Parameter	Description
attribute_list	Specifies the list of SQL statement attributes to return in the result. Possible values are: • TYPICAL — Specifies BASIC plus SQL plan (without row source statistics) and without object reference list (default). • BASIC — Specifies all attributes (such as execution statistics and binds) except the plans. The execution context is always part of the result. • ALL — Specifies all attributes. • Comma-separated list of attribute names. This values returns only a subset of SQL attributes: – EXECUTION_STATISTICS – BIND_LIST – OBJECT_LIST – SQL_PLAN – SQL_PLAN_STATISTICS — Similar to SQL_PLAN plus row source statistics
recursive_sql	Specifies that the filter must include recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL, which is the default) or exclude it (NO_RECURSIVE_SQL).

Return Values

This function returns a one SQLSET_ROW per SQL_ID or PLAN_HASH_VALUE pair found in each data source.

Usage Notes

- Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.
- Users need privileges on the shared SQL area views.

Examples

```
-- Get sql ids and sql text for statements with 500 buffer gets.
SELECT sql_id, sql_text
FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('buffer_gets > 500'))
ORDER BY sql_id;

-- Get all the information we have about a particular statement.
SELECT *
FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('sql_id = ''4rm4183czbs7j'''));

-- Notice that some statements can have multiple plans.  The output of the
-- SELECT_XXX table functions is unique by (sql_id, plan_hash_value).  This is
-- because a data source can store multiple plans per sql statement.
SELECT sql_id, plan_hash_value
FROM table(dbms_sqltune.select_cursor_cache('sql_id = ''ay1m3ssvtrh24'''))
ORDER BY sql_id, plan_hash_value;

-- PL/SQL examples: load_sqlset is called after opening a cursor, along the
```

```
-- lines given below

-- Select all statements in the shared SQL area.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT value(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE) P;

  -- Process each statement (or pass cursor to load_sqlset).

  CLOSE cur;
END;/


-- Look for statements not parsed by SYS.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur for
    SELECT VALUE(P)
    FROM table(
     DBMS_SQLTUNE.SELECT_CURSOR_CACHE('parsing_schema_name <> ''SYS''')) P;

  -- Process each statement (or pass cursor to load_sqlset).

  CLOSE cur;
end;/


-- All statements from a particular module/action.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(
      DBMS_SQLTUNE.SELECT_CURSOR_CACHE(
         'module = ''MY_APPLICATION'' and action = ''MY_ACTION''')) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;/


-- all statements that ran for at least five seconds
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE('elapsed_time > 5000000')) P;

  -- Process each statement (or pass cursor to load_sqlset)
```

```
  CLOSE cur;
end;/


-- select all statements that pass a simple buffer_gets threshold and
-- are coming from an APPS user
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(
      DBMS_SQLTUNE.SELECT_CURSOR_CACHE(
        'buffer_gets > 100 and parsing_schema_name = ''APPS'''))P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
end;/


-- select all statements exceeding 5 seconds in elapsed time, but also
-- select the plans (by default we only select execution stats and binds
-- for performance reasons - in this case the SQL_PLAN attribute of sqlset_row
-- is NULL)
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(dbms_sqltune.select_cursor_cache(
      'elapsed_time > 5000000', NULL, NULL, NULL, NULL, 1, NULL,
      'EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN')) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;/


-- Select the top 100 statements in the shared SQL area ordering by
elapsed_time.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE(NULL,
                                                NULL,
                                                'ELAPSED_TIME', NULL, NULL,
                                                1,
                                                100)) P;

  -- Process each statement (or pass cursor to load_sqlset)
```

```
  CLOSE cur;
end;/


-- Select the set of statements which cumulatively account for 90% of the
-- buffer gets in the shared SQL area.  This means that the buffer gets of all
-- of these statements added up is approximately 90% of the sum of all
-- statements currently in the cache.
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLTUNE.SELECT_CURSOR_CACHE(NULL,
                                                NULL,
                                                'BUFFER_GETS', NULL, NULL,
                                                .9)) P;

  -- Process each statement (or pass cursor to load_sqlset).

  CLOSE cur;
END;
/
```

SELECT_SQL_TRACE Function

This table function reads the content of one or more trace files and returns the SQL statements it finds in the format of `sqlset_row`.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SELECT_SQL_TRACE (
  directory              IN VARCHAR2,
  file_name              IN VARCHAR2 := NULL,
  mapping_table_name     IN VARCHAR2 := NULL,
  mapping_table_owner    IN VARCHAR2 := NULL,,
  select_mode            IN POSITIVE := SINGLE_EXECUTION,
  options                IN BINARY_INTEGER := LIMITED_COMMAND_TYPE,
  pattern_start          IN VARCHAR2 := NULL,
  pattern_end            IN VARCHAR2 := NULL,
  result_limit           IN POSITIVE := NULL)
 RETURN sys.sqlset PIPELINED;
```

Parameters

Table 195-42 SELECT_SQL_TRACE Function Parameters

Parameter	Description
directory	Defines the directory object containing the trace files. This field is mandatory.
file_name	Specifies all or part of the name of the trace files. If NULL, then the function uses the current or most recent file in the specified location or path. '%' wildcards are supported for matching trace file names.
mapping_table_name	Specifies the mapping table name. Note that the mapping table name is case insensitive. If the mapping table name is NULL, then the function uses the mappings in the current database.
mapping_table_owner	Specifies the mapping table owner. If it is NULL, then the function uses the current user.
select_mode	Specifies the mode for selecting SQL from the trace. Possible values are: • SINGLE_EXECUTION — Returns one execution of a SQL. This is the default. • ALL_EXECUTIONS — Returns all executions.
options	Specifies which types of SQL statements are returned. • LIMITED_COMMAND_TYPE — Returns the SQL statements with the command types CREATE, INSERT, SELECT, UPDATE, DELETE, and MERGE. This value is the default. • ALL_COMMAND_TYPE — Returns the SQL statements with all command types.
pattern_start	Specifies the delimiting pattern of the trace file sections to consider. CURRENTLY INOPERABLE.
pattern_end	Specifies the closing delimiting pattern of the trace file sections to process. CURRENTLY INOPERABLE.
result_limit	Specifies the top SQL from the filtered source. Default to MAXSB4 if NULL.

Return Values

This function returns a SQLSET_ROW object.

Usage Notes

The ability to create a directory object for the system directory creates a potential security issue. For example, in a CDB, all containers write trace files to the same directory. A local user with SELECT privileges on this directory can read the contents of trace files belonging to any container.

To prevent this type of unauthorized access, copy the files from the default SQL trace directory into a different directory, and then create a directory object. Use the PATH_PREFIX clause of the CREATE PLUGGABLE DATABASE statement to ensure that all directory object paths associated with the PDB are restricted to the specified directory or its subdirectories.

Examples

The following code shows how to enable SQL trace for a few SQL statements and load the results into a SQL tuning set:

```
-- turn on the SQL trace in the capture database
ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT FOREVER, LEVEL 4'

-- run sql statements
SELECT 1 FROM DUAL;
SELECT COUNT(*) FROM dba_tables WHERE table_name = :mytab;

ALTER SESSION SET EVENTS '10046 TRACE NAME CONTEXT OFF';

-- create mapping table from the capture database
CREATE TABLE mapping AS
SELECT object_id id, owner, substr(object_name, 1, 30) name
   FROM dba_objects
   WHERE object_type NOT IN ('CONSUMER GROUP', 'EVALUATION CONTEXT',
                             'FUNCTION', 'INDEXTYPE', 'JAVA CLASS',
                             'JAVA DATA', 'JAVA RESOURCE', 'LIBRARY',
                             'LOB', 'OPERATOR', 'PACKAGE',
                             'PACKAGE BODY', 'PROCEDURE', 'QUEUE',
                             'RESOURCE PLAN', 'TRIGGER', 'TYPE',
                             'TYPE BODY')
UNION ALL
SELECT user_id id, username owner, NULL name
   FROM dba_users;

-- create the directory object where the SQL traces are stored
CREATE DIRECTORY SQL_TRACE_DIR as '/home/foo/trace';

-- create the STS
EXEC DBMS_SQLTUNE.CREATE_SQLSET('my_sts', 'test purpose');

-- load the SQL statements into STS from SQL TRACE
DECLARE
   cur sys_refcursor;
BEGIN
   OPEN cur FOR
   SELECT value(p)
     FROM TABLE(
        DBMS_SQLTUNE.SELECT_SQL_TRACE(
           directory=>'SQL_TRACE_DIR',
           file_name=>'%trc',
           mapping_table_name=>'mapping')) p;
   DBMS_SQLTUNE.LOAD_SQLSET('my_sts', cur);
   CLOSE cur;
END;
/
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* to learn more about the `PATH_PREFIX` clause

SELECT_SQLPA_TASK Function

This function collects SQL statements from a SQL Performance Analyzer comparison task.

> **See Also:**
>
> - DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group
> - *Oracle Database Testing Guide* for a `SELECT_SQLPA_TASK` example

Syntax

```
DBMS_SQLTUNE.SELECT_SQLPA_TASK(
    task_name         IN VARCHAR2,
    task_owner        IN VARCHAR2 := NULL,
    execution_name    IN VARCHAR2 := NULL,
    level_filter      IN VARCHAR2 := 'REGRESSED',
    basic_filter      IN VARCHAR2 := NULL,
    object_filter     IN VARCHAR2 := NULL,
    attribute_list    IN VARCHAR2 := 'TYPICAL')
RETURN sys.sqlset PIPELINED;
```

Parameters

Table 195-43 SELECT_SQLPA_TASK Function Parameters

Parameter	Description
`task_name`	Specifies the name of the SQL Performance Analyzer task.
`task_owner`	Specifies the owner of the SQL Performance Analyzer task. If `NULL`, then assume the current user.
`execution_name`	Specifies the name of the SQL Performance Analyzer task execution (type `COMPARE PERFORMANCE`) from which the provided filters will be applied. If `NULL`, then assume the most recent `COMPARE PERFORMANCE` execution.

Table 195-43 (Cont.) SELECT_SQLPA_TASK Function Parameters

Parameter	Description
level_filter	Specifies which subset of SQL statements to include. Same format as DBMS_SQLPA.REPORT_ANALYSIS_TASK.LEVEL, with some possible strings removed. • IMPROVED includes only improved SQL. • REGRESSED includes only regressed SQL (default). • CHANGED includes only SQL with changed performance. • UNCHANGED includes only SQL with unchanged performance. • CHANGED_PLANS includes only SQL with plan changes. • UNCHANGED_PLANS includes only SQL with unchanged plans. • ERRORS includes only SQL with errors only. • MISSING_SQL includes only missing SQL statements (across STS). • NEW_SQL includes only new SQL statements (across STS).
basic filter	Specifies the SQL predicate to filter the SQL in addition to the level filters.
object_filter	Currently not supported.
attribute_list	Defines the SQL statement attributes to return in the result. Possible values are: • TYPICAL — Returns BASIC plus the SQL plan (without row source statistics) and without an object reference list. This is the default. • BASIC — Returns all attributes (such as execution statistics and binds) except the plans. The execution context is always part of the result. • ALL — Returns all attributes. • Comma-separated list of attribute names this allows to return only a subset of SQL attributes: EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN_STATISTICS (similar to SQL_PLAN + row source statistics).

Return Values

This function returns a SQL tuning set object.

Usage Notes

For example, you can use this function to create a SQL tuning set containing the subset of SQL statements that regressed during a SQL Performance Analyzer (SPA) experiment. You can also specify other arbitrary filters.

SELECT_SQLSET Function

This is a table function that reads the contents of a SQL tuning set.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SELECT_SQLSET (
  sqlset_name          IN   VARCHAR2,
  basic_filter         IN   VARCHAR2 := NULL,
  object_filter        IN   VARCHAR2 := NULL,
  ranking_measure1     IN   VARCHAR2 := NULL,
  ranking_measure2     IN   VARCHAR2 := NULL,
  ranking_measure3     IN   VARCHAR2 := NULL,
  result_percentage    IN   NUMBER   := 1,
  result_limit         IN   NUMBER   := NULL)
  attribute_list       IN   VARCHAR2 := NULL,
  plan_filter          IN   VARCHAR2 := NULL,
  sqlset_owner         IN   VARCHAR2 := NULL,
  recursive_sql        IN   VARCHAR2 := HAS_RECURSIVE_SQL)
 RETURN sys.sqlset PIPELINED;
```

Parameters

Table 195-44 SELECT_SQLSET Function Parameters

Parameter	Description
`sqlset_name`	Specifies the name of the SQL tuning set to query.
`basic_filter`	Specifies the SQL predicate to filter the SQL from the SQL tuning set defined on attributes of the `SQLSET_ROW`.
`object_filter`	Currently not supported.
`ranking_measure(n)`	Specifies an `ORDER BY` clause on the selected SQL.
`result_percentage`	Specifies a filter that picks the top *n*% according to the supplied ranking measure. Note that this parameter applies only if one ranking measure is supplied.
`result_limit`	The top limit SQL from the filtered source, ranked by the ranking measure.
`attribute_list`	Defines the SQL statement attributes to return in the result. The possible values are: • `BASIC` — Returns all attributes (such as execution statistics and binds) except the plans. The execution context is included in the result. • `TYPICAL` — Returns `BASIC` plus the SQL plan, but without row source statistics and without the object reference list. This is the default. • `ALL` — Returns all attributes. • Comma-separated list of attribute names. This value enables the function to return only a subset of SQL attributes: – `EXECUTION_STATISTICS` – `SQL_BINDS` – `SQL_PLAN_STATISTICS` (similar to `SQL_PLAN` plus row source statistics)

Table 195-44 (Cont.) SELECT_SQLSET Function Parameters

Parameter	Description
plan_filter	Specifies the plan filter. This parameter enables you to select a single plan when a statement has multiple plans. Possible values are: • LAST_GENERATED — Returns the plan with the most recent timestamp. • FIRST_GENERATED — Returns the plan with the least recent timestamp. • LAST_LOADED — Returns the plan with the most recent FIRST_LOAD_TIME statistical information. • FIRST_LOADED — Returns the plan with the least recent FIRST_LOAD_TIME statistical information. • MAX_ELAPSED TIME — Returns the plan with the maximum elapsed time. • MAX_BUFFER_GETS — Returns the plan with the maximum buffer gets. • MAX_DISK_READS — Returns the plan with the maximum disk reads. • MAX_DIRECT_WRITES — Returns the plan with the maximum direct writes. • MAX_OPTIMIZER_COST — Returns the plan with the maximum optimizer cost value.
sqlset_owner	Specifies the owner of the SQL tuning set, or NULL for the current schema owner.
recursive_sql	Specifies that the filter must include recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL, which is the default) or exclude it (NO_RECURSIVE_SQL).

Return Values

This function returns one SQLSET_ROW per SQL_ID or PLAN_HASH_VALUE pair found in each data source.

Usage Notes

Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.

Examples

```
-- select from a sql tuning set
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE (P)
    FROM table(dbms_sqltune.select_sqlset('my_workload')) P;

  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;
/
```

SELECT_WORKLOAD_REPOSITORY Function

This function collects SQL statements from the workload repository.

The overloaded forms enable you to collect SQL statements from the following sources:

- Snapshots between `begin_snap` and `end_snap`
- A workload repository baseline

Syntax

```
DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY (
  begin_snap         IN NUMBER,
  end_snap           IN NUMBER,
  basic_filter       IN VARCHAR2 := NULL,
  object_filter      IN VARCHAR2 := NULL,
  ranking_measure1   IN VARCHAR2 := NULL,
  ranking_measure2   IN VARCHAR2 := NULL,
  ranking_measure3   IN VARCHAR2 := NULL,
  result_percentage  IN NUMBER   := 1,
  result_limit       IN NUMBER   := NULL,
  attribute_list     IN VARCHAR2 := NULL,
  recursive_sql      IN VARCHAR2 := HAS_RECURSIVE_SQL,
  dbid               IN NUMBER   := NULL)
 RETURN sys.sqlset PIPELINED;

DBMS_SQLTUNE.SELECT_WORKLOAD REPOSITORY (
  baseline_name      IN VARCHAR2,
  basic_filter       IN VARCHAR2 := NULL,
  object_filter      IN VARCHAR2 := NULL,
  ranking_measure1   IN VARCHAR2 := NULL,
  ranking_measure2   IN VARCHAR2 := NULL,
  ranking_measure3   IN VARCHAR2 := NULL,
  result_percentage  IN NUMBER   := 1,
  result_limit       IN NUMBER   := NULL,
  attribute_list     IN VARCHAR2 := NULL,
  recursive_sql      IN VARCHAR2 := HAS_RECURSIVE_SQL,
  dbid               IN NUMBER   := NULL)
 RETURN sys.sqlset PIPELINED;
```

Parameters

Table 195-45 SELECT_WORKLOAD_REPOSITORY Function Parameters

Parameter	Description
`begin_snap`	Defines the beginning AWR snapshot (non-inclusive).
`end_snap`	Defines the ending AWR snapshot (inclusive).
`baseline_name`	Specifies the name of the AWR baseline period.

Table 195-45 (Cont.) SELECT_WORKLOAD_REPOSITORY Function Parameters

Parameter	Description
basic_filter	Specifies the SQL predicate to filter the SQL from the workload repository. The filter is defined on attributes of the SQLSET_ROW. If basic_filter is not set by the caller, then the subprogram captures only statements of type CREATE TABLE, INSERT, SELECT, UPDATE, DELETE, and MERGE.
object_filter	Currently not supported.
ranking_measure(n)	Defines an ORDER BY clause on the selected SQL.
result_percentage	Specifies a filter that picks the top *n*% according to the supplied ranking measure. Note that this percentage applies only if one ranking measure is given.
result_limit	Specifies the top limit SQL from the source according to the supplied ranking measure.
attribute_list	Specifies the SQL statement attributes to return in the result. The possible values are: • TYPICAL — Returns BASIC plus SQL plan (without row source statistics) and without object reference list. This is the default. • BASIC — Returns all attributes (such as execution statistics and binds) are returned except the plans. The execution context is always part of the result. • ALL — Returns all attributes • Comma-separated list of attribute names this allows to return only a subset of SQL attributes: EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN_STATISTICS (similar to SQL_PLAN plus row source statistics).
recursive_sql	Specifies the filter that includes recursive SQL in the SQL tuning set (HAS_RECURSIVE_SQL) or excludes it (NO_RECURSIVE_SQL).
dbid	Specifies the DBID for imported or PDB-level AWR data. If NULL, then the function uses the current database DBID.

Return Values

This function returns one SQLSET_ROW per SQL_ID or PLAN_HASH_VALUE pair found in each data source.

Usage Notes

Filters provided to this function are evaluated as part of a SQL run by the current user. As such, they are executed with that user's security privileges and can contain any constructs and subqueries that user can access, but no more.

Examples

```
-- select statements from snapshots 1-2
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE (P)
    FROM table(dbms_sqltune.select_workload_repository(1,2)) P;
```

```
  -- Process each statement (or pass cursor to load_sqlset)

  CLOSE cur;
END;
/
```

SET_TUNING_TASK_PARAMETER Procedures

This procedure updates the value of a SQL tuning parameter of type `VARCHAR2` or `NUMBER`.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER (
   task_name        IN  VARCHAR2,
   parameter        IN  VARCHAR2,
   value            IN  VARCHAR2,
   database_link_to IN  VARCHAR2);

DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER (
   task_name        IN  VARCHAR2,
   parameter        IN  VARCHAR2,
   value            IN  NUMBER,
   database_link_to IN  VARCHAR2);
);
```

Parameters

Table 195-46 SET_TUNING_TASK_PARAMETER Procedure Parameters

Parameter	Description
`task_name`	Identifier of the task to execute

Table 195-46 (Cont.) SET_TUNING_TASK_PARAMETER Procedure Parameters

Parameter	Description
parameter	Name of the parameter to set. The possible tuning parameters that can be set by this procedure using the parameter in the form VARCHAR2: • APPLY_CAPTURED_COMPILENV: indicates whether the advisor could use the compilation environment captured with the SQL statements. The default is 0 (that is, NO). • BASIC_FILTER: basic filter for SQL tuning set • DAYS_TO_EXPIRE: number of days until the task is deleted • DEFAULT_EXECUTION_TYPE: the task defaults to this type of execution when none is specified by the EXECUTE_TUNING_TASK Function and Procedure • EXECUTION_DAYS_TO_EXPIRE: number of days until the tasks's executions is deleted (without deleting the task) • LOCAL_TIME_LIMIT: per-statement time out (seconds) • MODE: tuning scope (comprehensive, limited) • OBJECT_FILTER: object filter for SQL tuning set • PLAN_FILTER: plan filter for SQL tuning set (see SELECT_SQLSET for possible values) • RANK_MEASURE1: first ranking measure for SQL tuning set • RANK_MEASURE2: second possible ranking measure for SQL tuning set • RANK_MEASURE3: third possible ranking measure for SQL tuning set • RESUME_FILTER: a extra filter for SQL tuning sets besides BASIC_FILTER • SQL_LIMIT: maximum number of SQL statements to tune • SQL_PERCENTAGE: percentage filter of SQL tuning set statements • TEST_EXECUTE: FULL/AUTO/OFF. * FULL - test-execute for as much time as necessary, up to the local time limit for the SQL (or the global task time limit if no SQL time limit is set) * AUTO - test-execute for an automatically-chosen time proportional to the tuning time * OFF - do not test-execute • TIME_LIMIT: global time out (seconds) • USERNAME: username under which the statement is parsed
value	New value of the specified parameter

Table 195-46 (Cont.) SET_TUNING_TASK_PARAMETER Procedure Parameters

Parameter	Description
database_link_to	Name of a database link that exists on a standby database. The link specifies the connection to a primary database. By default, the value is null, which means that the SQL Tuning Advisor session is local. Use DBMS_SQLTUNE to tune high-load SQL statements running on a standby database in an Active Data Guard scenario. When you execute REPORT_TUNING_TASK locally on the standby database, the function uses the database link to obtain the data from the primary database, and then constructs it locally on the standby database. The database_link_to parameter must specify a private database link. This link must be owned by SYS and accessed by the default privileged user SYS$UMF. The following sample statement creates a link named lnk_to_pri: `CREATE DATABASE LINK lnk_to_pri CONNECT TO SYS$UMF IDENTIFIED BY password USING 'inst1';`

Usage Notes

When setting automatic tuning task parameters, use the SET_AUTO_TUNING_TASK_PARAMETER Procedures in the DBMS_AUTO_SQLTUNE package.

SQLTEXT_TO_SIGNATURE Function

This function returns a SQL text's signature. The signature can be used to identify SQL text in dba_sql_profiles.

> **See Also:**
>
> DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE (
  sql_text     IN CLOB,
  force_match IN BOOLEAN  := FALSE)
RETURN NUMBER;
```

Parameters

Table 195-47 SQLTEXT_TO_SIGNATURE Function Parameters

Parameter	Description
sql_text	SQL text whose signature is required. Required.

Table 195-47 (Cont.) SQLTEXT_TO_SIGNATURE Function Parameters

Parameter	Description
force_match	If TRUE, this returns a signature that supports SQL matching with literal values transformed into bind variables. If FALSE, returns the signature based on the text with literals not transformed

Return Values

This function returns the signature of the specified SQL text.

UNPACK_STGTAB_SQLPROF Procedure

This procedure copies profile data stored in the staging table to create profiles on the system.

> **See Also:**
>
> DBMS_SQLTUNE SQL Profile Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF (
   profile_name          IN VARCHAR2 := '%',
   profile_category      IN VARCHAR2 := 'DEFAULT',
   replace               IN BOOLEAN,
   staging_table_name    IN VARCHAR2,
   staging_schema_owner  IN VARCHAR2 := NULL);
```

Parameters

Table 195-48 UNPACK_STGTAB_SQLPROF Procedure Parameters

Parameter	Description
profile_name	The name of the profile to unpack (% wildcards acceptable, case-sensitive)
profile_category	The category from which to unpack profiles (% wildcards acceptable, case-sensitive)
replace	The option to replace profiles if they already exist. Note that profiles cannot be replaced if one in the staging table has the same name as an active profile in a different SQL statement. If FALSE, this function raises errors if you try to create a profile that already exists
staging_table_name	The name of the table on which to perform the remap operation (case-insensitive unless double quoted). Required.
staging_schema_owner	The schema where the table resides, or NULL for current schema (case-insensitive unless double quoted)

Usage Notes

Using this procedure requires the `CREATE ANY SQL PROFILE` privilege and the `SELECT` privilege on staging table.

Examples

```
-- Unpack all profiles stored in a staging table.
BEGIN
  DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF(
      replace            => FALSE
  ,   staging_table_name => 'PROFILE_STGTAB');
END;

-- If there is a failure during the unpack operation, you can find the profile
-- that caused the error and perform a remap_stgtab_sqlprof operation
targeting it.
-- You can resume the unpack operation by setting replace to TRUE so that
-- the profiles that were already created are replaced.
BEGIN
  DBMS_SQLTUNE.UNPACK_STGTAB_SQLPROF(
      replace            => TRUE
  ,   staging_table_name => 'PROFILE_STGTAB');
END;
```

UNPACK_STGTAB_SQLSET Procedure

This procedure copies one or more SQL tuning sets from their location in the staging table into the SQL tuning sets schema, making them proper SQL tuning sets.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET (
   sqlset_name          IN VARCHAR2 := '%',
   sqlset_owner         IN VARCHAR2 := NULL,
   replace              IN BOOLEAN,
   staging_table_name   IN VARCHAR2,
   staging_schema_owner IN VARCHAR2 := NULL);
```

Parameters

The parameters are identical for `DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET` and `DBMS_SQLSET.UNPACK_STGTAB`.

Table 195-49 UNPACK_STGTAB_SQLSET and UNPACK_STGTAB Procedure Parameters

Parameter	Description
sqlset_name	Specifies the name of the tuning set to unpack (not null). Wildcard characters (%) are supported to unpack multiple tuning sets in a single call. For example, specify % to unpack all tuning sets from the staging table.
sqlset_owner	Specifies the name of tuning set owner, or NULL for the current schema owner. Wildcard characters (%) are supported.
replace	Specifies whether to replace an existing SQL tuning set. If FALSE, then this procedure raises errors when you try to create a tuning set that already exists.
staging_table_name	Specifies the name of the staging table, moved after a call to the DBMS_SQLTUNE.PACK_STGTAB_SQLSET or DBMS_SQLSET.PACK_STGTAB procedure (case-sensitive).
staging_schema_owner	Specifies the name of staging table owner, or NULL for the current schema owner (case-sensitive).

Examples

```
 -- unpack all STS in the staging table
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(sqlset_name         => '%', -
                                       sqlset_owner        => '%', -
                                       replace             => FALSE, -
                                       staging_table_name  =>
'STGTAB_SQLSET');

-- errors can arise during STS unpack when a STS in the staging table has the
-- same name/owner as STS on the system.  In this case, users should call
-- remap_stgtab_sqlset to patch the staging table and with which to call
unpack
-- Replace set to TRUE.
EXEC DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(sqlset_name         => '%', -
                                       sqlset_owner        => '%', -
                                       replace             => TRUE, -
                                       staging_table_name  =>
'STGTAB_SQLSET');
```

UPDATE_SQLSET Procedures

This overloaded procedure updates selected fields for SQL statements in a SQL tuning set.

> **See Also:**
>
> DBMS_SQLTUNE SQL Tuning Set Subprograms for other subprograms in this group

Syntax

```
DBMS_SQLTUNE.UPDATE_SQLSET (
   sqlset_name      IN  VARCHAR2,
   sql_id           IN  VARCHAR2,
   attribute_name   IN  VARCHAR2,
   attribute_value  IN  VARCHAR2 := NULL);

DBMS_SQLTUNE.UPDATE_SQLSET (
   sqlset_name      IN  VARCHAR2,
   sql_id           IN  VARCHAR2,
   attribute_name   IN  VARCHAR2,
   attribute_value IN NUMBER := NULL);
```

Parameters

Table 195-50 UPDATE_SQLSET Procedure Parameters

Parameter	Description
sqlset_name	Specifies the name of the SQL tuning set.
sql_id	Specifies the identifier of the SQL statement to be updated.
plan_hash value	Specifies the hash value of the execution plan for a SQL statement. Use this parameter when you want to update the attribute for a specific plan for a statement, but not all plans for the statement.
attribute_name	Specifies the name of the attribute to be modified. You can update the text field for MODULE, ACTION, PARSING_SCHEMA_NAME, and OTHER. The only numerical field that you can update is PRIORITY. If a statement has multiple plans, then the procedure changes the attribute value for all plans.
attribute_value	Specifies the new value of the attribute.