18
DBMS_ADVISOR

DBMS_ADVISOR is part of the server manageability suite of advisors, a set of expert systems that identifies and helps resolve performance problems relating to database server components.

Some advisors have their own packages. For these advisors, Oracle recommends that you use the advisor-specific package rather than DBMS_ADVISOR. Each of the following advisors has its own package, tailored to its specific functionality:

- Automatic Database Diagnostic Monitor (DBMS_ADDM)
- SQL Performance Analyzer (DBMS_SQLPA)
- SQL Repair Advisor (DBMS_SQLDIAG)
- SQL Tuning Advisor (DBMS_SQLTUNE)
- Compression Advisor (DBMS_COMPRESSION.GET_COMPRESSION_RATIO)

SQL Access Advisor and Segment Advisor are the only advisors with common use cases for DBMS_ADVISOR. Undo Advisor and Compression Advisor do not support DBMS_ADVISOR subprograms.

This chapter contains the following topics:

- DBMS_ADVISOR Deprecated Subprograms
- DBMS_ADVISOR Security Model
- Summary of DBMS_ADVISOR Subprograms

See Also:

- *Oracle Database Administrator's Guide* to learn about Segment Advisor
- *Oracle Database Get Started with Performance Tuning* to learn how to use SQL Access Advisor in Enterprise Manager
- *Oracle Database SQL Tuning Guide* to learn more about SQL Access Advisor

DBMS_ADVISOR Deprecated Subprograms

The section lists programs that are deprecated with Oracle Database 11*g*.

Note:

Oracle recommends that you do not use deprecated procedures in new applications. Support for deprecated features is for backward compatibility only.

The following subprograms are deprecated:

- ADD_SQLWKLD_REF Procedure
- CREATE_SQLWKLD Procedure
- DELETE_SQLWKLD Procedure
- DELETE_SQLWKLD_REF Procedure
- DELETE_SQLWKLD_STATEMENT Procedure
- IMPORT_SQLWKLD_SCHEMA Procedure
- IMPORT_SQLWKLD_SQLCACHE Procedure
- IMPORT_SQLWKLD_STS Procedure
- IMPORT_SQLWKLD_SUMADV Procedure
- IMPORT_SQLWKLD_USER Procedure
- RESET_SQLWKLD Procedure
- SET_SQLWKLD_PARAMETER Procedure
- UPDATE_SQLWKLD_ATTRIBUTES Procedure
- UPDATE_SQLWKLD_STATEMENT Procedure

DBMS_ADVISOR Security Model

The `ADVISOR` privilege is required to use the DBMS_ADVISOR package.

Summary of DBMS_ADVISOR Subprograms

This topic lists and describes the subprograms in the DBMS_ADVISOR package.

In the following table, the `Used in` column lists advisors relevant for each subprogram, but excludes ADDM, SQL Performance Analyzer, SQL Repair Advisor, and SQL Tuning Advisor because these advisors have their own packages.

Table 18-1 DBMS_ADVISOR Package Subprograms

Subprogram	Description	Used in
ADD_SQLWKLD_REF Procedure	Adds a workload reference to an Advisor task (Caution: Deprecated Subprogram)	SQL Access Advisor
ADD_SQLWKLD_STATEMENT Procedure	Adds a single statement to a workload	SQL Access Advisor
ADD_STS_REF Procedure	Establishes a link between the current SQL Access Advisor task and a SQL tuning set	SQL Access Advisor
CANCEL_TASK Procedure	Cancels a currently executing task operation	Segment Advisor, SQL Access Advisor
COPY_SQLWKLD_TO_STS Procedure	Copies the contents of a SQL workload object to a SQL tuning set	SQL Access Advisor
CREATE_FILE Procedure	Creates an external file from a PL/SQL CLOB variable, which is useful for creating scripts and reports	SQL Access Advisor

Table 18-1 (Cont.) DBMS_ADVISOR Package Subprograms

Subprogram	Description	Used in
CREATE_OBJECT Procedure	Creates a new task object	Segment Advisor
CREATE_SQLWKLD Procedure	Creates a new workload object (Caution: Deprecated Subprogram)	SQL Access Advisor
CREATE_TASK Procedures	Creates a new Advisor task in the repository	Segment Advisor, SQL Access Advisor
DELETE_SQLWKLD Procedure	Deletes an entire workload object (Caution: Deprecated Subprogram)	SQL Access Advisor
DELETE_SQLWKLD_REF Procedure	Deletes an entire workload object (Caution: Deprecated Subprogram)	SQL Access Advisor
DELETE_SQLWKLD_STATEMENT Procedure	Deletes one or more statements from a workload (Caution: Deprecated Subprogram)	SQL Access Advisor
DELETE_STS_REF Procedure	Removes a link between the current SQL Access Advisor task and a SQL tuning set object	SQL Access Advisor
DELETE_TASK Procedure	Deletes the specified task from the repository	SQL Access Advisor
EXECUTE_TASK Procedure	Executes the specified task	Segment Advisor, SQL Access Advisor
GET_REC_ATTRIBUTES Procedure	Retrieves specific recommendation attributes from a task	SQL Access Advisor
GET_TASK_REPORT Function	Creates and returns a report for the specified task	
GET_TASK_SCRIPT Function	Creates and returns an executable SQL script of the Advisor task's recommendations in a buffer	SQL Access Advisor
IMPLEMENT_TASK Procedure	Implements the recommendations for a task	SQL Access Advisor
IMPORT_SQLWKLD_SCHEMA Procedure	Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)	SQL Access Advisor
IMPORT_SQLWKLD_SQLCACHE Procedure	Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)	SQL Access Advisor
IMPORT_SQLWKLD_STS Procedure	Imports data from a SQL tuning set into a SQL workload data object (Caution: Deprecated Subprogram)	SQL Access Advisor
IMPORT_SQLWKLD_SUMADV Procedure	Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)	SQL Access Advisor
IMPORT_SQLWKLD_USER Procedure	Imports data into a workload from the current SQL cache (Caution: Deprecated Subprogram)	SQL Access Advisor
INTERRUPT_TASK Procedure	Stops a currently executing task, ending its operations as it would at a normal exit, so that the recommendations are visible	Segment Advisor, SQL Access Advisor

Table 18-1 (Cont.) DBMS_ADVISOR Package Subprograms

Subprogram	Description	Used in
MARK_RECOMMENDATION Procedure	Sets the `annotation_status` for a particular recommendation	Segment Advisor, SQL Access Advisor
QUICK_TUNE Procedure	Performs an analysis on a single SQL statement	SQL Access Advisor
RESET_SQLWKLD Procedure	Resets a workload to its initial starting point (Caution: Deprecated Subprogram)	SQL Access Advisor
RESET_TASK Procedure	Resets a task to its initial state	Segment Advisor, SQL Access Advisor
SET_DEFAULT_SQLWKLD_PARAMETER Procedure	Imports data into a workload from schema evidence	SQL Access Advisor
SET_DEFAULT_TASK_PARAMETER Procedure	Modifies a default task parameter	Segment Advisor, SQL Access Advisor
SET_SQLWKLD_PARAMETER Procedure	Sets the value of a workload parameter	SQL Access Advisor
SET_TASK_PARAMETER Procedure	Sets the specified task parameter value	Segment Advisor, SQL Access Advisor
TUNE_MVIEW Procedure	Shows how to decompose a materialized view into two or more materialized views or to restate the materialized view in a way that is more advantageous for fast refresh and query rewrite	SQL Access Advisor
UPDATE_OBJECT Procedure	Updates a task object	Segment Advisor
UPDATE_REC_ATTRIBUTES Procedure	Updates an existing recommendation for the specified task	SQL Access Advisor
UPDATE_SQLWKLD_ATTRIBUTES Procedure	Updates a workload object	SQL Access Advisor
UPDATE_SQLWKLD_STATEMENT Procedure	Updates one or more SQL statements in a workload	SQL Access Advisor
UPDATE_TASK_ATTRIBUTES Procedure	Updates a task's attributes	Segment Advisor, SQL Access Advisor

ADD_SQLWKLD_REF Procedure

This procedure establishes a link between the current SQL Access Advisor task and a SQL Workload object.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

The link allows an advisor task to access interesting data for doing an analysis. The link also provides a stable view of the data. Once a connection between a SQL Access Advisor task and a SQL Workload object is made, the workload is protected from removal or modification.

Users should use `ADD_STS_REF` instead of `ADD_SQLWKLD_REF` for all SQL tuning set-based advisor runs. This function is only provided for backward compatibility.

Syntax

```
DBMS_ADVISOR.ADD_SQLWKLD_REF (
   task_name               IN VARCHAR2,
   workload_name           IN VARCHAR2,
   is_sts                  IN NUMBER :=0);
```

Parameters

Table 18-2 ADD_SQLWKLD_REF Procedure Parameters

Parameter	Description
`task_name`	The SQL Access Advisor task name that uniquely identifies an existing task.
`workload_name`	The name of the workload object to be linked. Once a object has been linked to a task, it becomes read-only and cannot be deleted. There is no limit to the number of links to workload objects. To remove the link to the workload object, use the procedure `DELETE_REFERENCE`.
`is_sts`	Indicates the type of workload source. Possible values are: • 0 - SQL workload object • 1 - SQL tuning set

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name, 1);
```

```
END;
/
```

ADD_SQLWKLD_STATEMENT Procedure

This procedure adds a single statement to the specified workload.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT (
   workload_name           IN VARCHAR2,
   module                  IN VARCHAR2,
   action                  IN VARCHAR2,
   cpu_time                IN NUMBER := 0,
   elapsed_time            IN NUMBER := 0,
   disk_reads              IN NUMBER := 0,
   buffer_gets             IN NUMBER := 0,
   rows_processed          IN NUMBER := 0,
   optimizer_cost          IN NUMBER := 0,
   executions              IN NUMBER := 1,
   priority                IN NUMBER := 2,
   last_execution_date     IN DATE := 'SYSDATE',
   stat_period             IN NUMBER := 0,
   username                IN VARCHAR2,
   sql_text                IN CLOB);
```

Parameters

Table 18-3 ADD_SQLWKLD_STATEMENT Procedure Parameters

Parameter	Description
`workload_name`	The workload name that uniquely identifies an existing workload.
`module`	An optional business application module that will be associated with the SQL statement.
`action`	An optional application action that will be associated with the SQL statement.
`cpu_time`	The total CPU time in seconds that is consumed by the SQL statement.
`elapsed_time`	The total elapsed time in seconds that is consumed by the SQL statement.
`disk_reads`	The total disk-read operations that are consumed by the SQL statement.
`buffer_gets`	The total buffer-get operations that are consumed by the SQL statement.
`rows_processed`	The average number of rows processed by the SQL statement.
`optimizer_cost`	The cost value calculated by the optimizer.
`executions`	The total execution count of the SQL statement. This value should be greater than zero.

Table 18-3 (Cont.) ADD_SQLWKLD_STATEMENT Procedure Parameters

Parameter	Description
`priority`	The relative priority of the SQL statement. The value must be one of the following: 1-`HIGH`, 2-`MEDIUM`, or 3-`LOW`.
`last_execution_date`	The date and time at which the SQL statement last executed. If the value is `NULL`, then the database uses the current date and time.
`stat_period`	Time interval in seconds from which statement statistics were calculated.
`username`	The database user that executed the SQL statement. Because a user name is an Oracle identifier, the `username` value must be entered exactly as it is stored in the server. For example, if the user `SCOTT` is the executing user, then you must provide the user identifier `SCOTT` in all uppercase letters. It will not recognize the user `scott` or `Scott` as a match for `SCOTT`.
`sql_text`	The complete SQL statement. To increase the quality of a recommendation, the SQL statement should not contain bind variables.

Usage Notes

You cannot modify or delete a workload when it is currently referenced by an active task. A task is considered active if it is not in its initial state. See RESET_TASK Procedure for directions on setting a task to its initial state.

The `ADD_SQLWKLD_STATEMENT` procedure accepts several parameters that may be ignored by the caller. The database only uses the `disk_reads`, `buffer_gets`, and `optimizer_cost` parameters to sort workload data when actual analysis occurs. Therefore, actual values are only necessary when the `order_list` task parameter references a particular statistic.

To determine what statistics to provide when adding a new SQL statement to a workload, examine or set the task parameter `order_list`. The `order_list` parameter accepts any combination of the keys:

- `cpu_time`
- `elapsed_time`
- `buffer_gets`
- `optimizer_cost`
- `disk_reads`
- `executions`
- `priority`

The `optimizer_cost` key, which is a typical setting of `priority`, indicates that SQL Access Advisor sorts the workload data by `priority` and `optimizer_cost`, and processes the highest cost statements first. Any statements that you add to the workload must include appropriate `priority` and `optimizer_cost` values. All other statistics can be defaulted or set to zero.

For the statistical keys referenced by the `order_list` task parameter, the actual parameter values should be reasonably accurate since they will be compared to other statements in the workload. If the caller is unable to estimate values, then choose values that would determine its importance relative to other statements in the workload. For example, if the current statement is considered the most critical query in your business, then an appropriate value would be anything greater than all other values for the same statistic found in the workload.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT
AVG(amount_sold) FROM sh.sales');
END;
/
```

ADD_STS_REF Procedure

This procedure establishes a link between the current SQL Access Advisor task and a SQL tuning set.

The link enables an advisor task to access data for the purpose of doing an analysis. The link also provides a stable view of the data. Once a connection between a SQL Access Advisor task and a SQL tuning set is made, the STS is protected from removal or modification.

Use `ADD_STS_REF` for any STS-based advisor runs. The older method of using `ADD_SQLWKLD_REF` with parameter `IS_STS=1` is only supported for backward compatibility. Furthermore, the `ADD_STS_REF` function accepts a SQL tuning set owner name, whereas `ADD_SQLWKLD_REF` does not.

Syntax

```
DBMS_ADVISOR.ADD_STS_REF(
  task_name       IN VARCHAR2 NOT NULL,
  sts_owner       IN VARCHAR2,
  workload_name   IN VARCHAR2 NOT NULL);
```

Parameters

Table 18-4 ADD_STS_REF Procedure Parameters

Parameter	Description
`task_name`	The SQL Access Advisor task name that uniquely identifies an existing task.
`sts_owner`	The owner of the SQL tuning set. The value of this parameter may be `NULL`, in which case the advisor assumes the SQL tuning set to be owned by the currently logged-in user.

Table 18-4 (Cont.) ADD_STS_REF Procedure Parameters

Parameter	Description
`workload_name`	The name of the workload to be linked. A workload consists of one or more SQL statements, plus statistics and attributes that fully describe each statement. The database stores a workload as a SQL tuning set. After a workload has been linked to a task, it becomes read-only and cannot be deleted. There is no limit to the number of links to workloads. To remove the link to the workload, use the procedure `DBMS_ADVISOR.DELETE_STS_REF`.

Examples

```
DBMS_ADVISOR.ADD_STS_REF ('My Task', 'SCOTT', 'My Workload');
```

CANCEL_TASK Procedure

This procedure causes a currently executing operation to terminate.

This call performs a soft interrupt. It will not break into a low-level database access call like a hard interrupt such as `Ctrl-C`. The SQL Access Advisor periodically checks for soft interrupts and acts appropriately. As a result, this operation may take a few seconds to respond to a call.

Syntax

```
DBMS_ADVISOR.CANCEL_TASK (
   task_name       IN   VARCHAR2);
```

Parameters

Table 18-5 CANCEL_TASK Procedure Parameter

Parameter	Description
`task_name`	A valid Advisor task name that uniquely identifies an existing task.

Usage Notes

A cancel command restores the task to its condition prior to the start of the canceled operation. Therefore, a canceled task or data object cannot be resumed.

Because all Advisor task procedures are synchronous, to cancel an operation, you must use a separate database session.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
```

```
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.CANCEL_TASK('My Task');
END;
/
```

COPY_SQLWKLD_TO_STS Procedure

This procedure copies the contents of a SQL workload object to a SQL tuning set.

Syntax

To use this procedure, the caller must have privileges to create and modify a SQL tuning set.

```
DBMS_ADVISOR.COPY_SQLWKLD_TO_STS (
   workload_name          IN VARCHAR2,
   sts_name               IN VARCHAR2,
   import_mode            IN VARCHAR2 := 'NEW');
```

Parameters

Table 18-6   COPY_SQLWKLD_TO_STS Procedure Parameter

Parameter	Description
`workload_name`	The SQL Workload object name to copy.
`sts_name`	The SQL tuning set name into which the SQL Workload object will be copied.
`import_mode`	Specifies the handling of the target SQL tuning set. Possible values are: • `APPEND` Causes SQL Workload data to be appended to the target SQL tuning set. • `NEW` Indicates the SQL tuning set can only contain the copied contents. If the SQL tuning set exists and has data, an error will be reported. • `REPLACE` Causes any existing data in the target SQL tuning set to be purged prior to the workload copy. In all cases, if the specified SQL tuning set does not exist, it will be created.

Usage Notes

To use this procedure, the caller must have privileges to create and modify a SQL tuning set.

Examples

```
BEGIN
   DBMS_ADVISOR.COPY_SQLWKLD_TO_STS('MY_OLD_WORKLOAD', 'MY_NEW_STS', 'NEW');
END;
/
```

CREATE_FILE Procedure

This procedure creates an external file from a PL/SQL `CLOB` variable, which is used for creating scripts and reports.

Syntax

```
DBMS_ADVISOR.CREATE_FILE (
   buffer        IN  CLOB,
   location      IN  VARCHAR2,
   filename      IN  VARCHAR2);
```

Parameters

Table 18-7 CREATE_FILE Procedure Parameters

Parameter	Description
`buffer`	A `CLOB` buffer containing report or script information.
`location`	The name of the directory that will contain the output file. You must use the alias as defined by the `CREATE DIRECTORY` statement. The Advisor translates the alias into the actual directory location.
`filename`	The name of the output file. The file name can only contain the name and an optional file type of the form `filename.filetype`.

Usage Notes

You must embed all formatting within the `CLOB`.

The database restricts file access within stored procedures. This means that file locations and names must adhere to the known file permissions in the server.

Examples

```
CREATE DIRECTORY MY_DIR as '/homedir/user4/gssmith';
GRANT READ,WRITE ON DIRECTORY MY_DIR TO PUBLIC;

DECLARE
  v_task_id NUMBER;
  v_task_name VARCHAR2(30);
  v_workload_name VARCHAR2(30);
BEGIN
  v_task_name := 'My Task';
  v_workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(
      advisor_name => DBMS_ADVISOR.SQLACCESS_ADVISOR
  ,   task_id      => v_task_id
  ,   task_name    => v_task_name );
  DBMS_ADVISOR.CREATE_SQLWKLD(
      workload_name => v_workload_name
  ,   description   => 'My Workload' );
  DBMS_ADVISOR.ADD_SQLWKLD_REF(
      task_name     => v_task_name
  ,   workload_name => v_workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(
      workload_name       => v_workload_name
  ,   module              => 'MONTHLY'
  ,   action              => 'ROLLUP'
  ,   cpu_time            => 100
  ,   elapsed_time        => 400
  ,   disk_reads          => 5041
  ,   buffer_gets         => 103
  ,   rows_processed      => 640445
  ,   optimizer_cost      => 680000
  ,   executions          => 2
  ,   priority            => 1
  ,   last_execution_date => SYSDATE
  ,   stat_period         => 1
  ,   username            => 'SH'
  ,   sql_text            => 'SELECT AVG(amount_sold) FROM sh.sales' );
  DBMS_ADVISOR.EXECUTE_TASK(v_task_name);
  DBMS_ADVISOR.CREATE_FILE(
      buffer   => DBMS_ADVISOR.GET_TASK_SCRIPT(v_task_name)
  ,   location => 'MY_DIR'
  ,   filename => 'script.sql' );
END;
/
```

CREATE_OBJECT Procedure

This procedure creates a new task object.

Syntax

```
DBMS_ADVISOR.CREATE_OBJECT (
   task_name          IN VARCHAR2,
   object_type        IN VARCHAR2,
   attr1              IN VARCHAR2 :=  NULL,
   attr2              IN VARCHAR2 :=  NULL,
   attr3              IN VARCHAR2 :=  NULL,
   attr4              IN CLOB     :=  NULL,
   attr5              IN VARCHAR2 :=  NULL,
   object_id          OUT NUMBER,
   attr6              IN VARCHAR2 :=  NULL,
   attr7              IN VARCHAR2 :=  NULL,
   attr8              IN VARCHAR2 :=  NULL,
   attr9              IN VARCHAR2 :=  NULL,
   attr10             IN VARCHAR2 :=  NULL);
```

Parameters

Table 18-8 CREATE_OBJECT Procedure Parameters

Parameter	Description
`task_name`	A valid Advisor task name that uniquely identifies an existing task.
`object_type`	Specifies the external object type.
`attr1`	Advisor-specific data.
`attr2`	Advisor-specific data.
`attr3`	Advisor-specific data.
`attr4`	Advisor-specific data.
`attr5`	Advisor-specific data.
`object_id`	The advisor-assigned object identifier.
`attr6`	Advisor-specific data.
`attr7`	Advisor-specific data.
`attr8`	Advisor-specific data.
`attr9`	Advisor-specific data.
`attr10`	Advisor-specific data.

The attribute parameters have different values depending upon the object type. See *Oracle Database Administrator's Guide* for details regarding these parameters and object types.

Return Values

Returns the new object identifier.

Usage Notes

Task objects are typically used as input data for a particular advisor. Segment advice can be generated at the object, segment, or tablespace level. If for the object level, advice is generated on all partitions of the object (if the object is partitioned). The advice is not cascaded to any dependent objects. If for the segment level, advice can be obtained on a single segment, such as the partition or subpartition of a table, index, or `LOB` column. If for a tablespace level, target advice for every segment in the tablespace will be generated.

See *Oracle Database Administrator's Guide* for further information regarding the Segment Advisor.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  obj_id NUMBER;
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.CREATE_OBJECT (task_name,'SQL',NULL,NULL,NULL, 'SELECT * FROM
TEST_TAB',NULL,obj_id,NULL,NULL,NULL,NULL,NULL);
END;
/
```

CREATE_SQLWKLD Procedure

This procedure creates a new private SQL Workload object for the user.

A SQL Workload object manages a SQL workload on behalf of the SQL Access Advisor. A SQL Workload object must exist prior to performing any other SQL Workload operations, such as importing or updating SQL statements.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.CREATE_SQLWKLD (
   workload_name            IN OUT VARCHAR2,
   description              IN VARCHAR2 := NULL,
   template                 IN VARCHAR2 := NULL,
   is_template              IN VARCHAR2 := 'FALSE');
```

Parameters

Table 18-9 CREATE_SQLWKLD Procedure Parameters

Parameter	Description
`workload_name`	A name that uniquely identifies the created workload. If not specified, the system will generate a unique name. Names can be up to 30 characters long.
`description`	Specifies an optional workload description. Descriptions can be up to 256 characters.
`template`	An optional SQL Workload name of an existing workload data object or data object template.
`is_template`	An optional value that enables you to set the newly created workload as a template. Valid values are `TRUE` and `FALSE`.

Return Values

The SQL Access Advisor returns a unique workload object identifier number that must be used for subsequent activities within the new SQL Workload object.

Usage Notes

By default, workload objects are created using built-in default settings. To create a workload using the parameter settings of an existing workload or workload template, the user may specify an existing workload name.

After a SQL Workload object is present, it can then be referenced by one or more SQL Access Advisor tasks using the `ADD_SQLWKLD_REF` procedure.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
END;
/
```

CREATE_TASK Procedures

This procedure creates a new Advisor task in the repository.

Syntax

```
DBMS_ADVISOR.CREATE_TASK (
   advisor_name          IN VARCHAR2,
   task_id               OUT NUMBER,
   task_name             IN OUT VARCHAR2,
   task_desc             IN VARCHAR2 := NULL,
   template              IN VARCHAR2 := NULL,
   is_template           IN VARCHAR2 := 'FALSE',
   how_created           IN VARCHAR2 := NULL);

DBMS_ADVISOR.CREATE_TASK (
   advisor_name          IN VARCHAR2,
   task_name             IN VARCHAR2,
   task_desc             IN VARCHAR2 := NULL,
   template              IN VARCHAR2 := NULL,
   is_template           IN VARCHAR2 := 'FALSE',
   how_created           IN VARCHAR2 := NULL);

DBMS_ADVISOR.CREATE_TASK (
   parent_task_name      IN VARCHAR2,
   rec_id                IN NUMBER,
   task_id               OUT NUMBER,
   task_name             IN OUT VARCHAR2,
   task_desc             IN VARCHAR2,
   template              IN VARCHAR2);
```

Parameters

Table 18-10 CREATE_TASK Procedure Parameters

Parameter	Description
`advisor_name`	Specifies the unique advisor name as defined in the view `DBA_ADVISOR_DEFINITIONS`.
`task_id`	A number that uniquely identifies the created task. The number is generated by the procedure and returned to the user.

Table 18-10 (Cont.) CREATE_TASK Procedure Parameters

Parameter	Description
`task_name`	Specifies a new task name. Names must be unique among all tasks for the user. When using the second form of the `CREATE_TASK` syntax listed above (with `OUT`), a unique name can be generated. Names can be up to 30 characters long.
`task_desc`	Specifies an optional task description. Descriptions can be up to 256 characters in length.
`template`	An optional task name of an existing task or task template. To specify built-in SQL Access Advisor templates, use the template name as described earlier.
`is_template`	An optional value that allows the user to set the newly created task as template. Valid values are: `TRUE` and `FALSE`.
`how_created`	An optional value that identifies how the source was created.

Return Values

Returns a unique task ID number and a unique task name if one is not specified.

Usage Notes

A task must be associated with an advisor, and once the task has been created, it is permanently associated with the original advisor. By default, tasks are created using built-in default settings. To create a task using the parameter settings of an existing task or task template, the user may specify an existing task name.

For the SQL Access Advisor, use the identifier `DBMS_ADVISOR.SQLACCESS_ADVISOR` as the `advisor_name`.

The SQL Access Advisor provides three built-in task templates, using the following constants:

- `DBMS_ADVISOR.SQLACCESS_OLTP`

  Parameters are preset to favor an OLTP application environment.
- `DBMS_ADVISOR.SQLACCESS_WAREHOUSE`

  Parameters are preset to favor a data warehouse application environment.
- `DBMS_ADVISOR.SQLACCESS_GENERAL`

  Parameters are preset to favor a hybrid application environment where both OLTP and data warehouse operations may occur. For the SQL Access Advisor, this is the default template.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
```

```
END;
/
```

DELETE_SQLWKLD Procedure

This procedure deletes an existing SQL Workload object from the repository.

> **Note:**
>
> This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.DELETE_SQLWKLD (
   workload_name        IN VARCHAR2);
```

Parameters

Table 18-11 DELETE_SQLWKLD Procedure Parameters

Parameter	Description
`workload_name`	The workload object name that uniquely identifies an existing workload. The wildcard `%` is supported as a `WORKLOAD_NAME`. The rules of use are identical to the `LIKE` operator. For example, to delete all tasks for the current user, use the wildcard `%` as the `WORKLOAD_NAME`. If a wildcard is provided, the `DELETE_SQLWKLD` operation will not delete any workloads marked as `READ_ONLY` or `TEMPLATE`.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See the RESET_TASK Procedure to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.DELETE_SQLWKLD(workload_name);
END;
/
```

DELETE_SQLWKLD_REF Procedure

This procedure removes a link between the current SQL Access task and a SQL Workload data object.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Use `DELETE_STS_REF` instead of `DELETE_SQLWKLD_REF` for all SQL tuning set-based advisor runs. This function is only provided for backward compatibility.

Syntax

```
DBMS_ADVISOR.DELETE_SQLWKLD_REF (
   task_name               IN VARCHAR2,
   workload_name           IN VARCHAR2,
   is_sts                  IN NUMBER :=0);
```

Parameters

Table 18-12 DELETE_SQLWKLD_REF Procedure Parameters

Parameter	Description
`task_name`	The SQL Access task name that uniquely identifies an existing task.
`workload_name`	The name of the workload object to be unlinked. The wildcard `%` is supported as a `workload_name`. The rules of use are identical to the `LIKE` operator. For example, to remove all links to workload objects, use the wildcard `%` as the `workload_name`.
`is_sts`	Indicates the type of workload source. Possible values are: • 0 - SQL workload object • 1 - SQL tuning set

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.DELETE_SQLWKLD_REF(task_name, workload_name);
END;
/
```

DELETE_SQLWKLD_STATEMENT Procedure

This procedure deletes one or more statements from a workload.

Note:

This procedure has been deprecated.

Syntax

```
DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT (
   workload_name          IN VARCHAR2,
   sql_id                 IN NUMBER);

DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT (
   workload_name          IN VARCHAR2,
   search                 IN VARCHAR2,
   deleted                OUT NUMBER);
```

Parameters

Table 18-13 DELETE_SQLWKLD_STATEMENT Procedure Parameters

Parameter	Description
`workload_name`	The workload object name that uniquely identifies an existing workload.
`sql_id`	The Advisor-generated identifier number that is assigned to the statement. To specify all workload statements, use the constant `ADVISOR_ALL`.
`search`	Disabled.
`deleted`	Returns the number of statements deleted by the searched deleted operation.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See the RESET_TASK Procedure to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  deleted NUMBER;
  id NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'YEARLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
```

```
                                       FROM sh.sales');

   SELECT sql_id INTO id FROM USER_ADVISOR_SQLW_STMTS
   WHERE workload_name = 'My Workload';

  DBMS_ADVISOR.DELETE_SQLWKLD_STATEMENT(workload_name, id);
END;
/
```

DELETE_STS_REF Procedure

This procedure removes a link between the current SQL Access Advisor task and a SQL tuning set.

Use `DELETE_STS_REF` for any STS-based advisor runs. The older method of using `DELETE_SQLWKLD_REF` with parameter `IS_STS=1` is only supported for backward compatibility. Furthermore, the `DELETE_STS_REF` function accepts an STS owner name, whereas `DELETE_SQLWKLD_REF` does not.

Syntax

```
DBMS_ADVISOR.DELETE_STS_REF (
  task_name      IN VARCHAR2 NOT NULL,
  sts_owner      IN VARCHAR2,
  workload_name  IN VARCHAR2 NOT NULL);
```

Parameters

Table 18-14 DELETE_STS_REF Procedure Parameters

Parameter	Description
`task_name`	The SQL Access Advisor task name that uniquely identifies an existing task.
`sts_owner`	The owner of the SQL tuning set. The value of this parameter may be `NULL`, in which case the advisor assumes the SQL tuning set to be owned by the currently logged-in user.
`workload_name`	The name of the workload to be unlinked. A workload consists of one or more SQL statements, plus statistics and attributes that fully describe each statement. The database stores a workload as a SQL tuning set. The wildcard `%` is supported as a workload name. The rules of use are identical to the SQL `LIKE` operator. For example, to remove all links to SQL tuning set objects, use the wildcard `%` as the `STS_NAME`.

Examples

```
DBMS_ADVISOR.DELETE_STS_REF ('My task', 'SCOTT', 'My workload');
```

DELETE_TASK Procedure

This procedure deletes an existing task from the repository.

Syntax

```
DBMS_ADVISOR.DELETE_TASK (
   task_name          IN VARCHAR2);
```

Parameters

Table 18-15 DELETE_TASK Procedure Parameters

Parameter	Description
`task_name`	A single Advisor task name that will be deleted from the repository. The wildcard `%` is supported as a `TASK_NAME`. The rules of use are identical to the `LIKE` operator. For example, to delete all tasks for the current user, use the wildcard `%` as the `TASK_NAME`. If a wildcard is provided, the `DELETE_TASK` operation will not delete any tasks marked as `READ_ONLY` or `TEMPLATE`.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.DELETE_TASK(task_name);
END;
/
```

EXECUTE_TASK Procedure

This procedure performs the Advisor analysis or evaluation for the specified task. The procedure is overloaded.

The execution-related arguments are optional and you do not need to set them for advisors that do not allow their tasks to be executed multiple times.

Advisors can execute a task multiple times and use the results for further processing and analysis.

Syntax

```
DBMS_ADVISOR.EXECUTE_TASK (
   task_name          IN VARCHAR2);

DBMS_ADVISOR.EXECUTE_TASK (
   task_name         IN VARCHAR2,
   execution_type    IN VARCHAR2              := NULL,
   execution_name    IN VARCHAR2              := NULL,
   execution_params  IN dbms_advisor.argList := NULL,
   execution_desc    IN VARCHAR2              := NULL,
RETURN VARCHAR2;
```

Parameters

Table 18-16 EXECUTE_TASK Procedure Parameters

Parameter	Description
`task_name`	The task name that uniquely identifies an existing task.
`execution_type`	The type of action to be performed by the function. If `NULL`, it will default to the value of the `DEFAULT_EXECUTION_TYPE` parameter. As an example, the SQL Performance Analyzer accepts the following possible values: • `EXPLAIN PLAN`: Generate an explain plan for a SQL statement. This is similar to an `EXPLAIN PLAN` command. The resulting plans will be stored in the advisor framework in association with the task. • `TEST EXECUTE`: Test execute the SQL statement and collect its execute plan and statistics. The resulting plans and statistics are stored in the advisor framework. • `ANALYZE PERFORMANCE`: Analyze and compare two versions of SQL performance data. The performance data is generated by test executing a SQL statement or generating its explain plan.
`execution_name`	A name to qualify and identify an execution. If not specified, it will be generated by the Advisor and returned by function.
`execution_params`	A list of parameters (name, value) for the specified execution. Note that execution parameters are real task parameters, but they affect only the execution they are specified for. As an example, consider the following: `DBMS_ADVISOR.ARGLIST('time_limit', 12, 'username', 'hr')`
`execution_desc`	A 256-length string describing the execution.

Usage Notes

Task execution is a synchronous operation. Control will not be returned to the caller until the operation has completed, or a user-interrupt was detected.

Upon return, you can check the `DBA_ADVISOR_LOG` table for the execution status.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
END;
/
```

GET_REC_ATTRIBUTES Procedure

This procedure retrieves a specified attribute of a new object as recommended by Advisor analysis.

Syntax

```
DBMS_ADVISOR.GET_REC_ATTRIBUTES (
   workload_name          IN VARCHAR2,
   rec_id                 IN NUMBER,
   action_id              IN NUMBER,
   attribute_name         IN VARCHAR2,
   value                  OUT VARCHAR2,
   owner_name             IN VARCHAR2 := NULL);
```

Parameters

Table 18-17 GET_REC_ATTRIBUTES Procedure Parameters

Parameter	Description
`task_name`	The task name that uniquely identifies an existing task.
`rec_id`	The Advisor-generated identifier number that is assigned to the recommendation.
`action_id`	The Advisor-generated action identifier that is assigned to the particular command.
`attribute_name`	Specifies the attribute to change.
`value`	The buffer to receive the requested attribute value.
`owner_name`	Optional owner name of the target task. This permits access to task data not owned by the current user.

Return Values

The requested attribute value is returned in the `VALUE` argument.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
  attribute VARCHAR2(100);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';
  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                   100,400,5041,103,640445,680000,2,
                                   1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                   FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
  DBMS_ADVISOR.GET_REC_ATTRIBUTES(task_name, 1, 1, 'NAME', attribute);
END;
/
```

GET_TASK_REPORT Function

This function creates and returns a report for the specified task.

Syntax

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

Parameters

Table 18-18 GET_TASK_REPORT Function Parameters

Parameter	Description
`task_name`	The name of the task from which the script will be created.
`type`	The only valid value is `TEXT`.
`level`	The possible values are `BASIC`, `TYPICAL`, and `ALL`.
`section`	Advisor-specific report sections.
`owner_name`	Owner of the task. If specified, the system will check to see if the current user has read privileges to the task data.
`execution_name`	An identifier of a specific execution of the task. It is needed only for advisors that allow their tasks to be executed multiple times.
`object_id`	An identifier of an advisor object that can be targeted by the script.

Return Values

Returns the buffer receiving the script.

GET_TASK_SCRIPT Function

This function creates a SQL*Plus-compatible SQL script and sends the output to a file.

The output script contains all of the accepted recommendations from the specified task.

Syntax

```
DBMS_ADVISOR.GET_TASK_SCRIPT (
   task_name           IN VARCHAR2
   type                IN VARCHAR2 := 'IMPLEMENTATION',
   rec_id              IN NUMBER   := NULL,
   act_id              IN NUMBER   := NULL,
   owner_name          IN VARCHAR2 := NULL,
   execution_name      IN VARCHAR2 := NULL,
   object_id           IN NUMBER   := NULL)
RETURN CLOB;
```

Parameters

Table 18-19 GET_TASK_SCRIPT Function Parameters

Parameter	Description
`task_name`	The task name that uniquely identifies an existing task.
`type`	Specifies the type of script to generate. The possible values are `IMPLEMENTATION` and `UNDO`.
`rec_id`	An optional recommendation identifier number that can be used to extract a subset of the implementation script. A zero or the value `DBMS_ADVISOR.ADVISOR_ALL` indicates all accepted recommendations would be included. The default is to include all accepted recommendations for the task.
`act_id`	Optional action identifier number that can be used to extract a single action as a DDL command. A zero or the value `DBMS_ADVISOR.ADVISOR_ALL` indicates all actions for the recommendation would be included. The default is to include all actions for a recommendation.
`owner_name`	An optional task owner name.
`execution_name`	An identifier of a specific execution of the task. It is needed only for advisors that allow their tasks to be executed multiple times.
`object_id`	An identifier of an advisor object that can be targeted by the script.

Return Values

Returns the script as a `CLOB` buffer.

Usage Notes

Though the script is ready to execute, Oracle recommends that the user review the script for acceptable locations for new materialized views and indexes.

For a recommendation to appear in a generated script, it must be marked as accepted.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
  buf CLOB;
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id, 
task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                   100,400,5041,103,640445,680000,2,
                                   1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                   FROM sh.sales');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
    buf := DBMS_ADVISOR.GET_TASK_SCRIPT(task_name);
END;
/
```

IMPLEMENT_TASK Procedure

This procedure implements the recommendations of the specified Advisor task.

Syntax

```
DBMS_ADVISOR.IMPLEMENT_TASK (
   task_name          IN VARCHAR2,
   rec_id             IN NUMBER := NULL,
   exit_on_error      IN BOOLEAN := NULL);
```

Parameters

Table 18-20   IMPLEMENT_TASK Procedure Parameters

Parameter	Description
`task_name`	The name of the task.
`rec_id`	An optional recommendation ID.
`exit_on_error`	An optional Boolean to exit on the first error.

IMPORT_SQLWKLD_SCHEMA Procedure

This procedure constructs and loads a SQL workload based on schema evidence. The workload is also referred to as a hypothetical workload.

> **Note:**
>
> This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_SCHEMA (
   workload_name          IN VARCHAR2,
   import_mode            IN VARCHAR2 := 'NEW',
   priority               IN NUMBER := 2,
   saved_rows             OUT NUMBER,
   failed_rows            OUT NUMBER);
```

Parameters

Table 18-21 IMPORT_SQLWKLD_SCHEMA Procedure Parameters

Parameter	Description
`workload_name`	The workload object name that uniquely identifies an existing workload.
`import_mode`	Specifies the action to be taken when storing the workload. Possible values are: • `APPEND` Indicates that the collected workload will be added to any existing workload in the task. • `NEW` Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown. • `REPLACE` Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. The default value is `NEW`.
`priority`	Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following: 1-`HIGH`, 2-`MEDIUM`, or 3-`LOW`.
`failed_rows`	Returns the number or rows that were not saved due to syntax or validation errors
`saved_rows`	Returns the number of rows actually saved in the repository.

Return Values

This call returns the number of rows saved and failed as output parameters.

Usage Notes

To successfully import a hypothetical workload, the target schemas must contain dimensions.

If the `VALID_TABLE_LIST` parameter is not set, the search space may become very large and require a significant amount of time to complete. Oracle recommends that you limit your search space to specific set of tables.

If a task contains valid recommendations from a prior run, adding or modifying task will mark the task as invalid, preventing the viewing and reporting of potentially valuable recommendation data.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,'VALID_TABLE_LIST','SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_SCHEMA(workload_name, 'REPLACE', 1, saved,
     failed);
END;
/
```

IMPORT_SQLWKLD_SQLCACHE Procedure

This procedure creates a SQL workload from the current contents of the server's SQL cache.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_SQLCACHE (
   workload_name          IN VARCHAR2,
   import_mode            IN VARCHAR2 := 'NEW',
   priority               IN NUMBER := 2,
   saved_rows             OUT NUMBER,
   failed_rows            OUT NUMBER);
```

Parameters

Table 18-22 IMPORT_SQLWKLD_SQLCACHE Procedure Parameters

Parameter	Description
`workload_name`	The workload object name that uniquely identifies an existing workload.

Table 18-22 (Cont.) IMPORT_SQLWKLD_SQLCACHE Procedure Parameters

Parameter	Description
`import_mode`	Specifies the action to be taken when storing the workload. Possible values are: • `APPEND` Indicates that the collected workload will be added to any existing workload in the task. • `NEW` Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown. • `REPLACE` Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. The default value is `NEW`.
`priority`	Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following 1-`HIGH`, 2-`MEDIUM`, or 3-`LOW`.
`saved_rows`	Returns the number of rows saved as output parameters.
`failed_rows`	Returns the number of rows that were not saved due to syntax or validation errors.

Return Values

This call returns the number of rows saved and failed as output parameters.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See RESET_TASK Procedure to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,'VALID_TABLE_LIST','SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_SQLCACHE(workload_name, 'REPLACE', 1, saved,
     failed);
END;
/
```

IMPORT_SQLWKLD_STS Procedure

This procedure loads a SQL workload from an existing SQL tuning set. A SQL tuning set is typically created from the server workload repository using various time and data filters.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_STS (
   workload_name          IN VARCHAR2,
   sts_name               IN VARCHAR2,
   import_mode            IN VARCHAR2 := 'NEW',
   priority               IN NUMBER := 2,
   saved_rows             OUT NUMBER,
   failed_rows            OUT NUMBER);


DBMS_ADVISOR.IMPORT_SQLWKLD_STS (
   workload_name          IN VARCHAR2,
   sts_owner              IN VARCHAR2,
   sts_name               IN VARCHAR2,
   import_mode            IN VARCHAR2 := 'NEW',
   priority               IN NUMBER := 2,
   saved_rows             OUT NUMBER,
   failed_rows            OUT NUMBER);
```

Parameters

Table 18-23 IMPORT_SQLWKLD_STS Procedure Parameters

Parameter	Description
`workload_name`	The workload object name that uniquely identifies an existing workload.
`sts_owner`	The optional owner of the SQL tuning set.
`sts_name`	The name of an existing SQL tuning set workload from which the data will be imported. If the `sts_owner` value is not provided, the owner will default to the current user.
`import_mode`	Specifies the action to be taken when storing the workload. Possible values are: • `APPEND` Indicates that the collected workload will be added to any existing workload in the task. • `NEW` Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown. • `REPLACE` Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. The default value is `NEW`.

Table 18-23 (Cont.) IMPORT_SQLWKLD_STS Procedure Parameters

Parameter	Description
`priority`	Specifies the application priority for each statement that is saved in the workload object. The value must be one of the following: 1-`HIGH`, 2-`MEDIUM`, or 3-`LOW`. The default value is 2.
`saved_rows`	Returns the number of rows actually saved in the repository.
`failed_rows`	Returns the number of rows that were not saved due to syntax or validation errors.

Return Values

This call returns the number of rows saved and failed as output parameters.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See RESET_TASK Procedure to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,'VALID_TABLE_LIST','SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_STS(workload_name, 'MY_SQLSET', 'REPLACE', 1,
     saved, failed);
END;
/
```

IMPORT_SQLWKLD_SUMADV Procedure

This procedure collects a SQL workload from a Summary Advisor workload.

This procedure is intended to assist Oracle9*i* Database Summary Advisor users in the migration to SQL Access Advisor.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_SUMADV (
   workload_name          IN VARCHAR2,
   import_mode            IN VARCHAR2 := 'NEW',
   priority               IN NUMBER := 2,
   sumadv_id              IN NUMBER,
   saved_rows             OUT NUMBER,
   failed_rows            OUT NUMBER);
```

Parameters

Table 18-24 IMPORT_SQLWKLD_SUMADV Procedure Parameters

Parameter	Description
`workload_name`	The workload object name that uniquely identifies an existing workload.
`import_mode`	Specifies the action to be taken when storing the workload. Possible values are: • `APPEND` Indicates that the collected workload will be added to any existing workload in the task. • `NEW` Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown. • `REPLACE` Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. The default value is `NEW`.
`priority`	Specifies the default application priority for each statement that is saved in the workload object. If a Summary Advisor workload statement contains a priority of zero, the default priority will be applied. If the workload statement contains a valid priority, then the Summary Advisor priority will be converted to a comparable SQL Access Advisor priority. The value must be one of the following: 1-`HIGH`, 2-`MEDIUM`, or 3-`LOW`.
`sumadv_id`	Specifies the Summary Advisor workload identifier number.
`saved_rows`	Returns the number of rows actually saved in the repository.
`failed_rows`	Returns the number of rows that were not saved due to syntax or validation errors.

Return Values

This call returns the number of rows saved and failed as output parameters.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See RESET_TASK Procedure to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
  sumadv_id NUMBER;
BEGIN
  workload_name := 'My Workload';
  sumadv_id := 394;

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,'VALID_TABLE_LIST','SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_SUMADV(workload_name, 'REPLACE', 1, sumadv_id,
    saved, failed);
END;
/
```

IMPORT_SQLWKLD_USER Procedure

This procedure collects a SQL workload from a specified user table.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.IMPORT_SQLWKLD_USER (
   workload_name          IN VARCHAR2,
   import_mode            IN VARCHAR2 := 'NEW',
   owner_name             IN VARCHAR2,
   table_name             IN VARCHAR2,
   saved_rows             OUT NUMBER,
   failed_rows            OUT NUMBER);
```

Parameters

Table 18-25 IMPORT_SQLWKLD_USER Procedure Parameters

Parameter	Description
`workload_name`	The workload object name that uniquely identifies an existing workload.
`import_mode`	Specifies the action to be taken when storing the workload. Possible values are: • `APPEND` Indicates that the collected workload will be added to any existing workload in the task. • `NEW` Indicates that the collected workload will be the exclusive workload for the task. If an existing workload is found, an exception will be thrown. • `REPLACE` Indicates the collected workload will be the exclusive workload for the task. If an existing workload is found, it will be deleted prior to saving the new workload. The default value is `NEW`.
`owner_name`	Specifies the owner name of the table or view from which workload data will be collected.

Table 18-25 (Cont.) IMPORT_SQLWKLD_USER Procedure Parameters

Parameter	Description
`table_name`	Specifies the name of the table or view from which workload data will be collected.
`saved_rows`	Returns the number of rows actually saved in the workload object.
`failed_rows`	Returns the number of rows that were not saved due to syntax or validation errors.

Return Values

This call returns the number of rows saved and failed as output parameters.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See RESET_TASK Procedure to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  saved NUMBER;
  failed NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,'VALID_TABLE_LIST','SH.%');
  DBMS_ADVISOR.IMPORT_SQLWKLD_USER(workload_name, 'REPLACE', 'SH',
    'USER_WORKLOAD', saved, failed);
END;
/
```

INTERRUPT_TASK Procedure

This procedure stops a currently executing task.

The task will end its operations as it would at a normal exit. The user will be able to access any recommendations that exist to this point.

Syntax

```
DBMS_ADVISOR.INTERRUPT_TASK (
   task_name          IN VARCHAR2);
```

Parameters

Table 18-26 INTERRUPT_TASK Procedure Parameters

Parameter	Description
`task_name`	A single Advisor task name that will be interrupted.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
END;
/
```

While this session is executing its task, you can interrupt the task from a second session using the following statement:

```
BEGIN
  DBMS_ADVISOR.INTERRUPT_TASK('My Task');
END;
/
```

MARK_RECOMMENDATION Procedure

This procedure marks a recommendation for import or implementation.

Syntax

```
DBMS_ADVISOR.MARK_RECOMMENDATION (
   task_name           IN VARCHAR2
   id                  IN NUMBER,
   action              IN VARCHAR2);
```

Parameters

Table 18-27 MARK_RECOMMENDATION Procedure Parameters

Parameter	Description
`task_name`	Name of the task.
`id`	The recommendation identifier number assigned by the Advisor.

Table 18-27 (Cont.) MARK_RECOMMENDATION Procedure Parameters

Parameter	Description
`action`	The recommendation action setting. The possible actions are: • `ACCEPT` Marks the recommendation as accepted. With this setting, the recommendation will appear in implementation and undo scripts. • `IGNORE` Marks the recommendation as ignore. With this setting, the recommendation will not appear in an implementation or undo script. • `REJECT` Marks the recommendation as rejected. With this setting, the recommendation will not appear in any implementation or undo scripts.

Usage Notes

For a recommendation to be implemented, it must be marked as accepted. By default, all recommendations are considered accepted and will appear in any generated scripts.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
  attribute VARCHAR2(100);
  rec_id NUMBER;
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                   100,400,5041,103,640445,680000,2,
                                   1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                   FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);

  rec_id := 1;
  DBMS_ADVISOR.MARK_RECOMMENDATION(task_name, rec_id, 'REJECT');
END;
/
```

QUICK_TUNE Procedure

This procedure performs an analysis and generates recommendations for a single SQL statement.

This provides a shortcut method of all necessary operations to analyze the specified SQL statement. The operation creates a task using the specified task name. The task will be created using a specified Advisor task template. Finally, the task will be executed and the results will be saved in the repository.

Syntax

```
DBMS_ADVISOR.QUICK_TUNE (
   advisor_name           IN VARCHAR2,
   task_name              IN VARCHAR2,
   attr1                  IN CLOB,
   attr2                  IN VARCHAR2 := NULL,
   attr3                  IN NUMBER := NULL,
   template               IN VARCHAR2 := NULL,
   implement              IN BOOLEAN := FALSE,
   description            IN VARCHAR2 := NULL);
```

Parameters

Table 18-28 QUICK_TUNE Procedure Parameters

Parameter	Description
`advisor_name`	Name of the Advisor that will perform the analysis.
`task_name`	Name of the task.
`attr1`	Advisor-specific attribute in the form of a `CLOB` variable.
`attr2`	Advisor-specific attribute in the form of a `VARCHAR2` variable.
`attr3`	Advisor-specific attribute in the form of a `NUMBER`.
`template`	Name of an existing task or template from which the initial settings need to be copied.
`implement`	Flag specifying whether to implement the task.
`description`	Description of the task.

Usage Notes

If indicated by the user, the final recommendations can be implemented by the procedure.

The task will be created using either a specified SQL Access task template or the built-in default template of `SQLACCESS_GENERAL`. The workload will only contain the specified statement, and all task parameters will be defaulted.

`attr1` must be the single SQL statement to tune. For the SQL Access Advisor, `attr2` is the user who would execute the single statement. If omitted, the current user will be used.

Examples

```
DECLARE
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.QUICK_TUNE(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_name,
                 'SELECT AVG(amount_sold) FROM sh.sales WHERE promo_id=10');
END;
/
```

RESET_SQLWKLD Procedure

This procedure resets a workload to its initial starting point.

Resetting the workload has the effect of removing all journal and log messages, and recalculating necessary volatility and usage statistics.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.RESET_SQLWKLD (
   workload_name        IN VARCHAR2);
```

Parameters

Table 18-29 RESET_SQLWKLD Procedure Parameters

Parameter	Description
workload_name	The SQL Workload object name that uniquely identifies an existing workload.

Usage Notes

RESET_SQLWKLD should be executed after any workload adjustments such as adding or removing SQL statements.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                     100,400,5041,103,640445,680000,2,
                                     1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                     FROM sh.sales WHERE promo_id = 10');

  DBMS_ADVISOR.RESET_SQLWKLD(workload_name);
END;
/
```

RESET_TASK Procedure

This procedure re-initializes the metadata for the specified task. The task status will be set to `INITIAL`.

Syntax

```
DBMS_ADVISOR.RESET_TASK (
   task_name          IN VARCHAR2);
```

Parameters

Table 18-30 RESET_TASK Procedure Parameters

Parameter	Description
`task_name`	The task name that uniquely identifies an existing task.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.EXECUTE_TASK(task_name);
  DBMS_ADVISOR.RESET_TASK(task_name);
END;
/
```

SET_DEFAULT_SQLWKLD_PARAMETER Procedure

This procedure modifies the default value for a user parameter within a SQL Workload object or SQL Workload object template.

A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting. When a default value is changed for a parameter, workload objects will inherit the new value when they are created.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER (
   parameter             IN VARCHAR2,
   value                 IN VARCHAR2);

DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER (
   parameter             IN VARCHAR2,
   value                 IN NUMBER);
```

Parameters

Table 18-31 SET_DEFAULT_SQLWKLD_PARAMETER Procedure Parameters

Parameter	Description
`parameter`	The name of the data parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the workload object type, but not necessarily unique to all workload object types. Various object types may use the same parameter name for different purposes.
`value`	The value of the specified parameter. The value can be specified as a string or a number. If the value is `DBMS_ADVISOR.DEFAULT`, the value will be reset to the default value.

Usage Notes

A parameter will only affect operations that modify the workload collection. Therefore, parameters should be set prior to importing or adding new SQL statements to a workload. If a parameter is set after data has been placed in a workload object, it will have no effect on the existing data.

Examples

```
BEGIN
  DBMS_ADVISOR.SET_DEFAULT_SQLWKLD_PARAMETER('VALID_TABLE_LIST','SH.%');
END;
/
```

SET_DEFAULT_TASK_PARAMETER Procedure

This procedure modifies the default value for a user parameter within a task or a template.

A user parameter is a simple variable that stores various attributes that affect various Advisor operations. When a default value is changed for a parameter, tasks will inherit the new value when they are created.

A default task is different from a regular task. The default value is the initial value that will be inserted into a newly created task, while setting a task parameter with `SET_TASK_PARAMETER` sets the local value only. Thus, `SET_DEFAULT_TASK_PARAMETER` has no effect on an existing task.

Syntax

```
DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER (
   advisor_name          IN VARCHAR2
   parameter             IN VARCHAR2,
   value                 IN VARCHAR2);

DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER (
   advisor_name          IN VARCHAR2
   parameter             IN VARCHAR2,
   value                 IN NUMBER);
```

Parameters

Table 18-32   SET_DEFAULT_TASK_PARAMETER Procedure Parameters

Parameter	Description
`advisor_name`	Specifies the unique advisor name as defined in the view `DBA_ADVISOR_DEFINITIONS`.
`parameter`	The name of the task parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the task type, but not necessarily unique to all task types. Various task types may use the same parameter name for different purposes.
`value`	The value of the specified task parameter. The value can be specified as a string or a number.

Examples

```
BEGIN
  DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER(DBMS_ADVISOR.SQLACCESS_ADVISOR,
   'VALID_TABLE_LIST', 'SH.%');
END;
/
```

SET_SQLWKLD_PARAMETER Procedure

This procedure modifies a user parameter within a SQL Workload object or SQL Workload object template.

A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.SET_SQLWKLD_PARAMETER (
   workload_name          IN VARCHAR2,
   parameter              IN VARCHAR2,
   value                  IN VARCHAR2);

DBMS_ADVISOR.SET_SQLWKLD_PARAMETER (
   workload_name          IN VARCHAR2,
   parameter              IN VARCHAR2,
   value                  IN NUMBER);
```

Parameters

Table 18-33 SET_SQLWKLD_PARAMETER Procedure Parameters

Parameter	Description
`workload_name`	The SQL Workload object name that uniquely identifies an existing workload.
`parameter`	The name of the data parameter to be modified. Parameter names are not case sensitive.
`value`	The value of the specified parameter. The value can be specified as a string or a number. If the value is `DBMS_ADVISOR.DEFAULT`, the value will be reset to the default value.

Usage Notes

A parameter will only affect operations that modify the workload collection. Therefore, parameters should be set prior to importing or adding new SQL statements to a workload. If a parameter is set after data has been placed in a workload object, it will have no effect on the existing data.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(workload_name,
'VALID_TABLE_LIST','SH.%');
END;
/
```

SET_TASK_PARAMETER Procedure

This procedure modifies a user parameter within an Advisor task or a template. A user parameter is a simple variable that stores various attributes that affect workload collection, tuning decisions and reporting.

Syntax

```
DBMS_ADVISOR.SET_TASK_PARAMETER (
   task_name            IN VARCHAR2
   parameter            IN VARCHAR2,
   value                IN VARCHAR2);

DBMS_ADVISOR.SET_TASK_PARAMETER (
   task_name            IN VARCHAR2
   parameter            IN VARCHAR2,
   value                IN NUMBER);
```

Parameters

Table 18-34 SET_TASK_PARAMETER Procedure Parameters

Parameter	Description
`task_name`	The Advisor task name that uniquely identifies an existing task.
`parameter`	The name of the task parameter to be modified. Parameter names are not case sensitive. Parameter names are unique to the task type, but not necessarily unique to all task types. Various task types may use the same parameter name for different purposes.
`value`	The value of the specified task parameter. The value can be specified as a string or a number. If the value is `DEFAULT`, the value will be reset to the default value.

Usage Notes

A task cannot be modified unless it is in its initial state. See RESET_TASK Procedure to set a task to its initial state. See your Advisor-specific documentation for further information on using this procedure.

SQL Access Advisor Task Parameters

Table 18-35 lists SQL Access Advisor task parameters.

Table 18-35   SQL Access Advisor Task Parameters

Parameter	Description
ANALYSIS_SCOPE	A comma-separated list that specifies the tuning artifacts to consider during analysis. The possible values are: • ALL Short name for specifying INDEX, MVIEW, TABLE, and PARTITION. • EVALUATION Causes a read-only evaluation of the specified workload. No new recommendations will be made. Can only be specified alone. • INDEX Allows the SQL Access Advisor to recommend index structure changes. • MVIEW Allows the SQL Access Advisor to recommend materialized view and log changes. • PARTITION Allows the SQL Access Advisor to recommend partition options. Use this in conjunction with the INDEX, MVIEW, and TABLE options. • TABLE Allows the SQL Access Advisor to make base-table recommendations. In this release, the only base-table recommendation is partitioning. Using the new keywords, the following combinations are valid: • INDEX • MVIEW • INDEX, PARTITION • INDEX, MVIEW, PARTITION • INDEX, TABLE, PARTITION • MVIEW, PARTITION • MIVEW, TABLE, PARTITION • INDEX, MVIEW, TABLE, PARTITION • TABLE, PARTITION • EVALUATION The default value is INDEX. The data type is STRINGLIST.
CREATION_COST	When set to true (default), the SQL Access Advisor will weigh the cost of creation of the access structure (index or materialized view) against the frequency of the query and potential improvement in the query execution time. When set to false, the cost of creation is ignored. The data type is STRING.
DAYS_TO_EXPIRE	Specifies the expiration time in days for the current SQL Access Advisor task. The value is relative to the last modification date. Once the task expires, it will become a candidate for removal by an automatic purge operation. Specifies the expiration time in days for the current Access Advisor task. The value is relative to the last modification date. The data type is NUMBER. Once the task expires, it becomes a candidate for removal by an automatic purge operation. The possible values are: • an integer in the range of 0 to 2147483647 • ADVISOR_UNLIMITED • ADVISOR_UNUSED The default value is 30.
DEF_EM_TEMPLATE	Contains the default task or template name from which the Enterprise Manager SQL Access Advisor Wizard reads its initial values. The default value is SQLACCESS_EMTASK. The data type is STRING.

Table 18-35 (Cont.) SQL Access Advisor Task Parameters

Parameter	Description
`DEF_INDEX_OWNER`	Specifies the default owner for new index recommendations. When a script is created, this value will be used to qualify the index name. Possible values are: • Existing schema name. Quoted identifiers are supported. • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `STRING`.
`DEF_INDEX_TABLESPACE`	Specifies the default tablespace for new index recommendations. When a script is created, this value will be used to specify a tablespace clause. Possible values are: • Existing tablespace name. Quoted identifiers are supported. • `ADVISOR_UNUSED` No tablespace clause will be present in the script for indexes. The default value is `ADVISOR_UNUSED`. The data type is `STRING`.
`DEF_MVIEW_OWNER`	Specifies the default owner for new materialized view recommendations. When a script is created, this value will be used to qualify the materialized view name. Possible values are: • Existing schema name. Quoted identifiers are supported. • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `STRING`.
`DEF_MVIEW_TABLESPACE`	Specifies the default tablespace for new materialized view recommendations. When a script is created, this value will be used to specify a tablespace clause. Possible values are • Existing tablespace name. Quoted identifiers are supported. • `ADVISOR_UNUSED`. No tablespace clause will be present in the script for materialized view logs. The default value is `ADVISOR_UNUSED`. The data type is `STRING`.
`DEF_MVLOG_TABLSPACE`	Specifies the default tablespace for new materialized view log recommendations. When a script is created, this value will be used to specify a tablespace clause. Possible values are: • Existing tablespace name. Quoted identifiers are supported. • `ADVISOR_UNUSED`. No tablespace clause will be present in the script for materialized view logs. The default value is `ADVISOR_UNUSED`. The data type is `STRING`.
`DEF_PARTITION_TABLESPACE`	Specifies the default tablespace for new partitioning recommendations. When a script is created, this value will be used to specify a tablespace clause. Possible values are: • Existing tablespace name. Quoted identifiers are supported. • `ADVISOR_UNUSED`. No tablespace clause will be present in the script for materialized views. The default value is `ADVISOR_UNUSED`. The data type is `STRING`.

Table 18-35 (Cont.) SQL Access Advisor Task Parameters

Parameter	Description
`DML_VOLATILITY`	When set to `TRUE`, the SQL Access Advisor will consider the impact of index maintenance and materialized view refresh in determining the recommendations. It will limit the access structure recommendations involving columns or tables that are frequently updated. For example, if there are too many DML statements on a column, then it may favor a B-tree index over a bitmap index on that column. For this process to be effective, the workload must include DML (insert/update/delete/merge/direct path inserts) statements that represent the update behavior of the application. The data type is `STRING`. See the related parameter `refresh_mode`.
`END_TIME`	Specifies an end time for selecting SQL statements. If the statement did not execute on or before the specified time, it will not be processed. Each date must be in the standard Oracle form of MM-DD-YYYY HH24:MI:SS, where: • DD is the numeric date • MM is the numeric month • YYYY is the numeric year • HH is the hour in 24 hour format • MI is the minute • SS is the second The data type is `STRING`.
`EVALUATION_ONLY`	This parameter is maintained for backward compatibility. All values will be translated and placed into the `ANALYSIS_SCOPE` task parameter. If set to `TRUE`, causes SQL Access Advisor to analyze the workload, but only comment on how well the current configuration is supporting it. No tuning recommendations will be generated. Possible values are: • `FALSE` • `TRUE` The default value is `FALSE`. The data type is `STRING`.
`EXECUTION_TYPE`	This parameter is maintained for backward compatibility. All values will be translated and placed into the `ANALYSIS_SCOPE` task parameter. The translated values are: • `FULL` => `FULL` • `INDEX_ONLY` => `INDEX` • `MVIEW_ONLY` => `MVIEW` • `MVIEW_LOG_ONLY` => `MVIEW_LOG_ONLY` The type of recommendations that is desired. Possible values: • `FULL` All supported recommendation types will be considered. • `INDEX_ONLY` The SQL Access Advisor will only consider index solutions as recommendations. • `MVIEW_ONLY` The SQL Access Advisor will consider materialized view and materialized view log solutions as recommendations. • `MVIEW_LOG_ONLY` The SQL Access Advisor will only consider materialized view log solutions as recommendations. The default value is `FULL`. The data type is `STRINGLIST`.

Table 18-35 (Cont.) SQL Access Advisor Task Parameters

Parameter	Description
`IMPLEMENT_EXIT_ON_ERROR`	When performing an `IMPLEMENT_TASK` operation, this parameter will control behavior when an action fails to implement. If set to `TRUE`, `IMPLEMENT_TASK` will stop on the first unexpected error. The possible values are: • `TRUE` • `FALSE` The default value is `TRUE`. The data type is `STRING`.
`INDEX_NAME_TEMPLATE`	Specifies the method by which new index names are formed. If the *TASK_ID* is omitted from the template, names generated by two concurrently executing SQL Access Advisor tasks may conflict and cause undesirable effects. So it is recommended that you include the *TASK_ID* in the template. Once formatted, the maximum size of a name is 30 characters. Valid keywords are: • Any literal value up to 22 characters. • `TABLE` Causes the parent table name to be substituted into the index name. If the name is too long, it will be trimmed to fit. • `TASK_ID` Causes the current task identifier number to be inserted in hexadecimal form. • `SEQ` Causes a sequence number to be inserted in hexadecimal form. Because this number is used to guarantee uniqueness, it is a required token. The default template is *`table`*`_IDX$$_`*`task_idsequence`*. The data type is `STRING`.
`INVALID_ACTION_LIST`	Contains a fully qualified list of actions that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness. During a task execution, if a SQL statement's action matches a name in the action list, it will not be processed by the task. An action name is case sensitive. The possible values are: • single action • comma-delimited action list • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `STRINGLIST`.
`INVALID_MODULE_LIST`	Contains a fully qualified list of modules that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness. During a task execution, if a SQL statement's module matches a name in the list, it will not be processed by the task. A module name is case sensitive. The possible values are: • single application • comma-delimited module list • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `STRINGLIST`.

Table 18-35 (Cont.) SQL Access Advisor Task Parameters

Parameter	Description
`INVALID_SQLSTRING_LIST`	Contains a fully qualified list of text strings that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted values are supported. A SQL string can be any string. If a string is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A SQL string is not scanned for correctness. During a task execution, if a SQL statement contains a string in the SQL string list, it will not be processed by the task. The possible values are: • single string • comma-delimited string list • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `STRINGLIST`.
`INVALID_USERNAME_LIST`	Contains a fully qualified list of user names that are not eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. During a task execution, if a SQL statement's user name matches a name in the user name list, it will not be processed by the task. A user name is not case sensitive unless it is quoted. The possible values are: • single user name • comma-delimited user name list • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `STRINGLIST`.
`JOURNALING`	Controls the logging of messages to the journal (`DBA_ADVISOR_JOURNAL` and `USER_ADVISOR_JOURNAL` views). The higher the setting, the more information is logged to the journal. Possible values are: • `UNUSED`: no journal messages • `FATAL`: explanation of fatal conditions • `ERROR`: explanation of errors • `WARNING`: explanation of warnings • `INFORMATION`: information message • `INFORMATION2`: common information • `INFORMATION3`: common information • `INFORMATION4`: common information • `INFORMATION5`: common information • `INFORMATION6`: common information Each journal value represents all recorded messages at that level or lower. For example, when choosing `WARNING`, all messages marked `WARNING` as well as `ERROR` and `FATAL` will be recorded in the repository. `INFORMATION6` represents the most thorough message recording and `UNUSED` is the least. The default value is `INFORMATION`. The data type is `NUMBER`.

Table 18-35 (Cont.) SQL Access Advisor Task Parameters

Parameter	Description
`LIMITED_PARTITION_SCHEMES`	User can suggest that the Partition Expert cut off the number of partitioning schemes to investigate. This can help with cutting down the run time of the advisor. Possible values are: • An integer in the range of 1 to 10 • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `NUMBER`.
`MAX_NUMBER_PARTITIONS`	Limits the number of partitions the advisor will recommend for any base table, index, or materialized view. Possible values are: • An integer in the range of 1 to 4294967295 • `ADVISOR_UNLIMITED` • `ADVISOR_UNUSED` The default value is `ADVISOR_UNLIMITED`. The data type is `NUMBER`.
`MODE`	Specifies the mode by which Access Advisor will operate during an analysis. Valid values are: • `LIMITED` Indicates the Advisor will attempt to a quick job by limiting the search-space of candidate recommendations, and correspondingly, the results may be of a low quality. • `COMPREHENSIVE` Indicates the Advisor will search a large pool of candidates that may take long to run, but the resulting recommendations will be of the highest quality. The default value is `COMPREHENSIVE`. The data type is `STRING`.
`MVIEW_NAME_TEMPLATE`	Specifies the method by which new materialized view names are formed. If the *TASK_ID* is omitted from the template, names generated by two concurrently executing SQL Access Advisor tasks may conflict and cause undesirable effects. So it is recommended that you include the *TASK_ID* in the template. The format is any combination of keyword tokens and literals. However, once formatted, the maximum size of a name is 30 characters. Valid tokens are: • Any literal value up to 22 characters. • *TASK_ID* Causes the current task identifier number to be inserted in hexadecimal form. • *SEQ* Causes a sequence number to be inserted in hexadecimal form. Because this number is used to guarantee uniqueness, it is a required token. The default template is: `MV$$_`*`task_idsequence`*. The data type is `STRING`.

Table 18-35 (Cont.) SQL Access Advisor Task Parameters

Parameter	Description
`ORDER_LIST`	This parameter has been deprecated. Contains the primary natural order in which the Access Advisor processes workload elements during the analysis operation. To determine absolute natural order, Access Advisor sorts the workload using `ORDER_LIST` values. A comma must separate multiple order keys. Possible values are: • `BUFFER_GETS` Sets the order using the SQL statement's buffer-get count value. • `CPU_TIME` Sets the order using the SQL statement's CPU time value. • `DISK_READS` Sets the order using the SQL statement's disk-read count value. • `ELAPSED_TIME` Sets the order using the SQL statement's elapsed time value. • `EXECUTIONS` Sets the order using the SQL statement's execution frequency value. • `OPTIMIZER_COST` Sets the order using the SQL statement's optimizer cost value. • `I/O` Sets the order using the SQL statement's I/O count value. • `PRIORITY` Sets the order using the user-supplied business priority value. All values are accessed in descending order, where a high value is considered more interesting than a low value. The default value is `PRIORITY`, `OPTIMIZER_COST`. The data type is `STRINGLIST`.
`PARTITION_NAME_TEMPLATE`	Specifies the method by which new partition names are formed. The format is any combination of keyword tokens and literals. However, once formatted, the maximum size of a name is 30 characters. Valid tokens are: • Any literal value up to 22 characters. • *`table`* - Causes the parent table name to be substituted into the partition name. If the name is too long, it will be trimmed to fit. • *`task_id`* - Causes the current task identifier number to be inserted in hexadecimal form. • *`sequence`* - Causes a sequence number to be inserted in hexadecimal form. Because this number is used to guarantee uniqueness, it is a required token. The default template is `PTN$$_`*`table_task_idsequence`*. The data type is `STRING`.
`PARTITIONING_GOAL`	Specifies the approach used to make partitioning recommendations. One possible value is `PERFORMANCE`, which is the default. The data type is `STRING`.
`PARTITIONING_TYPES`	Specifies the type of partitioning used. Possible values are `RANGE` and `HASH`. The data type is `STRING`.

Table 18-35 (Cont.) SQL Access Advisor Task Parameters

Parameter	Description
RANKING_MEASURE	Contains the primary natural order in which the SQL Access Advisor processes workload elements during the analysis operation. To determine absolute natural order, SQL Access Advisor sorts the workload using RANKING_MEASURE values. A comma must separate multiple order keys. Possible values are: • BUFFER_GETS Sets the order using the SQL statement's buffer-get count value. • CPU_TIME Sets the order using the SQL statement's CPU time value. • DISK_READS Sets the order using the SQL statement's disk-read count value. • ELAPSED_TIME Sets the order using the SQL statement's elapsed time value. • EXECUTIONS Sets the order using the SQL statement's elapsed time value. • OPTIMIZER_COST Sets the order using the SQL statement's optimizer cost value. • PRIORITY Sets the order using the user-supplied business priority value. All values are accessed in descending order, where a high value is considered more interesting than a low value. The default value is PRIORITY, OPTIMIZER_COST. The data type is STRINGLIST.
RECOMMEND_MV_EXACT_TEXT_MATCH	When considering candidate materialized views, exact text match solutions will only be included if this parameter contains TRUE. The possible values are: • TRUE • FALSE The default value is TRUE. The data type is STRING.
RECOMMENDED_TABLESPACES	Allows the SQL Access Advisor to recommend optimal tablespaces for any partitioning scheme. If this is not set, the SQL Access Advisor will simply recommend a partitioning method but give no advice on physical storage. Possible values are: • TRUE • FALSE (the default) The data type is STRING.
REFRESH_MODE	Specifies whether materialized views are refreshed ON_DEMAND or ON_COMMIT. This will be used to weigh the impact of materialized view refresh when the parameter dml_volatility is set to TRUE. Possible values are: • ON_DEMAND • ON_COMMIT The default value is ON_DEMAND. The data type is STRING.
REPORT_DATE_FORMAT	This is the default date and time formatting template. The default format is DD/MM/YYYYHH24:MI. The data type is STRING.
SHOW_RETAINS	Controls the display of RETAIN actions within an implementation script and the SQL Access Advisor wizard. The possible values are: • TRUE • FALSE The default value is TRUE. The data type is STRING.

Table 18-35 (Cont.) SQL Access Advisor Task Parameters

Parameter	Description
`SQL_LIMIT`	Specifies the number of SQL statements to be analyzed. The `SQL_LIMIT` filter is applied after all other filters have been applied. For example, if only statements referencing the table `hr.employees` are to be accepted, the `SQL_LIMIT` value will be only apply to those statements. When used in conjunction with the parameter `ORDER_LIST`, SQL Access Advisor will process the most interesting SQL statements by ordering the statements according to the specified sort keys. The possible values are: • An integer in the range of 1 to 2147483647 • `ADVISOR_UNLIMITED` • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `NUMBER`.
`START_TIME`	Specifies a start time for selecting SQL statements. If the statement did not execute on or before the specified time, it will not be processed. Each date must be in the standard Oracle form of MM-DD-YYYY HH24:MI:SS, where: • DD is the numeric date • MM is the numeric month • YYYY is the numeric year • HH is the hour in 24 hour format • MI is the minute • SS is the second The data type is `STRING`.
`STORAGE_CHANGE`	Contains the amount of space adjustment that can be consumed by SQL Access Advisor recommendations. Zero or negative values are only permitted if the workload scope is marked as `FULL`. When the SQL Access Advisor produces a set of recommendations, the resultant physical structures must be able to fit into the budgeted space. A space budget is computed by adding the `STORAGE_CHANGE` value to the space quantity currently used by existing access structures. A negative `STORAGE_CHANGE` value may force SQL Access Advisor to remove existing structures in order to shrink space demand. Possible values: • Any valid integer including negative values, zero and positive values. The default value is `ADVISOR_UNLIMITED`. The data type is `NUMBER`.
`TIME_LIMIT`	Specifies the time in minutes that the SQL Access Advisor can use to perform an analysis operation. If the SQL Access Advisor reaches a specified recommendation quality or all input data has been analyzed, processing will terminate regardless of any remaining time. Possible values: • An integer in the range of 1 to 10,000 • `ADVISOR_UNLIMITED` The default value is 720 (12 hours). The data type is `NUMBER`. Note that specifying `ADVISOR_UNLIMITED` has the same effect as setting the parameter to the maximum of 10,000 (about one week). The SQL Access Advisor will never run for more than 10,000 minutes.

Table 18-35 (Cont.) SQL Access Advisor Task Parameters

Parameter	Description
`VALID_ACTION_LIST`	Contains a fully qualified list of actions that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. An action can be any string. If an action is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. An action string is not scanned for correctness. During a task execution, if a SQL statement's action does not match a name in the action list, it will not be processed by the task. An action name is case sensitive. The possible values are: • single action • comma-delimited action list • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `STRINGLIST`.
`VALID_MODULE_LIST`	Contains a fully qualified list of application modules that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. A module can be any string. If a module is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A module string is not scanned for correctness. During a task execution, if a SQL statement's module does not match a name in the module list, it will not be processed by the task. A module name is case sensitive. The possible values are: • single application • comma-delimited module list • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `STRINGLIST`.
`VALID_SQLSTRING_LIST`	Contains a fully qualified list of text strings that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. A SQL string can be any string. If a string is not quoted, it will be changed to uppercase lettering and stripped of leading and trailing spaces. A SQL string is not scanned for correctness. During a task execution, if a SQL statement does not contain string in the SQL string list, it will not be processed by the task. The possible values are: • single string • comma-delimited string list • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `STRINGLIST`.

Table 18-35 (Cont.) SQL Access Advisor Task Parameters

Parameter	Description
`VALID_TABLE_LIST`	Contains a fully qualified list of tables that are eligible for tuning. The list elements are comma-delimited, and quoted identifiers are supported. Wildcard specifications are supported for tables. The default value is all tables within the user's scope are eligible for tuning. Supported wildcard character is `%`. A `%` wildcard matches any set of consecutive characters. When a SQL statement is processed, it will not be accepted unless at least one referenced table is specified in the valid table list. If the list is unused, then all table references within a SQL statement are considered valid. The valid syntax for a table reference is: • `schema.table` • `schema` • `schema.%` (equivalent to `schema`) • comma-delimited action list • `ADVISOR_UNUSED` The possible values are: • single table reference • comma-delimited reference list • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `TABLELIST`.
`VALID_USERNAME_LIST`	Contains a fully qualified list of user names that are eligible for processing in a SQL workload object. The list elements are comma-delimited, and quoted names are supported. During a task execution, if a SQL statement's user name does not match a name in the user name list, it will not be processed by the task. A user name is not case sensitive unless it is quoted. The possible values are: • single user name • comma-delimited user name list • `ADVISOR_UNUSED` The default value is `ADVISOR_UNUSED`. The data type is `STRINGLIST`.
`WORKLOAD_SCOPE`	Describes the level of application coverage the workload represents. Possible values are `FULL` and `PARTIAL`. `FULL` Should be used if the workload contains all interesting application SQL statements for the targeted tables. `PARTIAL` (default) Should be used if the workload contains anything less than a full representation of the interesting application SQL statements for the targeted tables. The data type is `STRING`.

Segment Advisor Parameters

Table 18-36 lists the input task parameters that can be set in the Segment Advisor using the `SET_TASK_PARAMETER` procedure.

Table 18-36 Segment Advisor Task Parameters

Parameter	Description
`MODE`	The data to use for analysis. The default value is `COMPREHENSIVE`, and the possible values are: • `LIMITED`: Analysis restricted to statistics available in the Automatic Workload Repository • `COMPREHENSIVE`: Analysis based on sampling and Automatic Workload Repository statistics
`TIME_LIST`	The time limit for which the Advisor should run. It is specified in seconds, and the default and possible values are `UNLIMITED`.
`RECOMMEND_ALL`	Whether to generate recommendations for all segments. The default value is `TRUE`. If set to `TRUE`, it generates recommendations all segments specified by the user. If set to `FALSE`, it generates recommendations for only those objects that are eligible for shrink.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

   DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
   DBMS_ADVISOR.SET_TASK_PARAMETER(task_name, 'VALID_TABLELIST',
      'SH.%,SCOTT.EMP');
END;
/
```

Undo Advisor Task Parameters

Table 18-37 lists the input task parameters that can be set in the Undo Advisor using the `SET_TASK_PARAMETER` procedure.

Table 18-37 Undo Advisor Task Parameters

Parameter	Description
`TARGET_OBJECTS`	The undo tablespace of the system. There is no default value, and the possible value is `UNDO_TBS`.
`START_SNAPSHOT`	The starting time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value and the possible values are the valid snapshot numbers in the AWR repository.
`END_SNAPSHOT`	The ending time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value and the possible values are the valid snapshot numbers in the AWR repository.
`BEGIN_TIME_SEC`	The number of seconds between the beginning time of the period and now. Describes a period of time for the system to perform analysis. `BEGIN_TIME_SEC` should be greater than `END_TIME_SEC`. There is no default value and the possible values are any positive integer.

Table 18-37 (Cont.) Undo Advisor Task Parameters

Parameter	Description
`END_TIME_SEC`	The number of seconds between the ending time of the period and now. `END_TIME_SEC` should be less than `BEGIN_TIME_SEC`. There is no default value and the possible values are any positive integer.

Examples

```
DECLARE
   tname  VARCHAR2(30);
   oid    NUMBER;
   BEGIN
     DBMS_ADVISOR.CREATE_TASK('Undo Advisor', tid, tname, 'Undo Advisor
Task');
     DBMS_ADVISOR.CREATE_OBJECT(tname, 'UNDO_TBS', null, null, null, 'null',
oid);
     DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'TARGET_OBJECTS', oid);
     DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'START_SNAPSHOT', 1);
     DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'END_SNAPSHOT', 2);
     DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'INSTANCE', 1);
     DBMS_ADVISOR.EXECUTE_TASK(tname);
   END;
/
```

Automatic Database Diagnostic Monitor (ADDM) Task Parameters

Table 18-38 lists the input task parameters that can be set in ADDM using the `SET_TASK_PARAMETER` procedure.

Table 18-38 ADDM Task Parameters

Parameter	Description
`START_SNAPSHOT`	The starting time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value, and the possible values are the valid snapshot numbers in the AWR repository.
`END_SNAPSHOT`	The ending time for the system to perform analysis using the snapshot numbers in the AWR repository. There is no default value, and the possible values are the valid snapshot numbers in the AWR repository.
`DB_ID`	The database for `START_SNAPSHOT` and `END_SNAPSHOT`. The default value is the current database ID.
`INSTANCE`	The instance for `START_SNAPSHOT` and `END_SNAPSHOT`. The default value is 0 or `UNUSED`, and the possible values are all positive integers. By default, all instances are analyzed.
`INSTANCES`	If the `INSTANCE` parameter has been set, `INSTANCES` is ignored. The default value is `UNUSED`, and the possible values are comma-separated list of instance numbers (for example, "1, 3, 5"). By default, all instances are analyzed.
`DBIO_EXPECTED`	The average time to read the database block in microseconds. The default value is 10 milliseconds, and the possible values are system-dependent.

Examples

The following creates and executes an ADDM task for the current database and an AWR snapshot range between 19 and 26. Note that this example will analyze all instances, whether you have only one or an Oracle RAC database.

```
DECLARE
   tid     NUMBER;
   tname VARCHAR2(30) := 'ADDM_TEST';
BEGIN
   DBMS_ADVISOR.CREATE_TASK('ADDM', tid, tname, 'my test');
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'START_SNAPSHOT', '19');
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'END_SNAPSHOT', '26');
   DBMS_ADVISOR.EXECUTE_TASK(tname);
END;
/
```

See Also:

- *Oracle Database Performance Tuning Guide* to learn more about using ADDM
- The DBMS_ADDM package for details on how to create and execute ADDM tasks

SQL Tuning Advisor Task Parameters

See the DBMS_SQLTUNE package and *Oracle Database SQL Tuning Guide* for more information.

TUNE_MVIEW Procedure

This procedure shows how to decompose a materialized view into multiple views and to restate the materialized view to be optimized for fast refresh and query rewrite. It also shows how to fix materialized view logs and to enable query rewrite.

Syntax

```
DBMS_ADVISOR.TUNE_MVIEW (
   task_name        IN OUT VARCHAR2,
   mv_create_stmt   IN     [CLOB | VARCHAR2]);
```

Parameters

Table 18-39 TUNE_MVIEW Procedure Parameters

Parameter	Description
`task_name`	The task name for querying the results in a catalog view. If not specified, the database generates a task name, and then returns.
`mv_create_stmt`	The original materialized view creation statement.

Usage Notes

Executing TUNE_MVIEW generates two sets of output results: one for the implementation, and the other for undoing the implementation. The output is accessible through USER_TUNE_MVIEW and DBA_TUNE_MVIEW views. You can also use DBMS_ADVISOR.GET_TASK_SCRIPT and DBMS_ADVISOR.CREATE_FILE to print the TUNE_MVIEW results into a script file for later execution.

Table 18-40 USER_TUNE_MVIEW and DBA_TUNE_MVIEW Views

Column Name	Column Description
OWNER	The name of the materialized view owner.
TASK_NAME	The name of the task. This name serves as a key to access the set of recommendations.
SCRIPT_TYPE	Recommendation ID that indicates whether the row is for the IMPLEMENTATION or UNDO script.
ACTION_ID	Action ID used as the command order number.
STATEMENT	For TUNE_MVIEW output, this column represents the following statements, and includes statement properties such as REFRESH and REWRITE options: • CREATE MATERIALIZED VIEW LOG • ALTER MATERIALIZED VIEW LOG FORCE • [CREATE | DROP] MATERIALIZED VIEW

Examples

The following example shows how to use TUNE_MVIEW to optimize a CREATE MATERIALIZED VIEW statement:

```
DECLARE
  v_tname VARCHAR2(30);
BEGIN
  v_tname := 'mview_task';
  DBMS_ADVISOR.TUNE_MVIEW(
      task_name      => v_tname
  ,   mv_create_stmt =>
       'CREATE MATERIALIZED VIEW omv REFRESH WITH ROWID AS SELECT * FROM
orders');
END;
```

You can view the results by querying USER_TUNE_MVIEW or DBA_TUNE_MVIEW as the following example (sample output included):

```
SET LINESIZE 120
COL TASK_NAME FORMAT a20
COL STATEMENT FORMAT a40

SELECT *
FROM   USER_TUNE_MVIEW
WHERE  TASK_NAME='mview_task'
AND    SCRIPT_TYPE='IMPLEMENTATION';

TASK_NAME             ACTION_ID SCRIPT_TYPE    STATEMENT
-------------------- ---------- --------------
```

```
----------------------------------------
mview_task                    1 IMPLEMENTATION CREATE MATERIALIZED VIEW LOG
ON "OE"."OR
                                               DERS" WITH ROWID

mview_task                    2 IMPLEMENTATION ALTER MATERIALIZED VIEW LOG
FORCE ON "OE
                                               "."ORDERS" ADD ROWID

mview_task                    3 IMPLEMENTATION CREATE MATERIALIZED VIEW
OE.OMV REFRESH
                                               FAST WITH ROWID DISABLE QUERY
REWRITE
```

Alternatively, you can save the output results in an external script file as in the following example:

```
CREATE DIRECTORY TUNE_RESULTS_DIR AS  '/tmp';
GRANT READ, WRITE ON DIRECTORY TUNE_RESULTS_DIR TO PUBLIC;
BEGIN
  DBMS_ADVISOR.CREATE_FILE(
      buffer     => DBMS_ADVISOR.GET_TASK_SCRIPT( task_name => 'mview_task')
  ,   location   => 'TUNE_RESULTS_DIR'
  ,   filename   => 'mview_create.sql' );
END;
```

The preceding statement will save the results in `/tmp/mview_create.sql`.

See Also:

*Oracle Database SQL Tuning Guide*for more information about using the `TUNE_MVIEW` procedure

UPDATE_OBJECT Procedure

This procedure updates an existing task object.

Task objects are typically used as input data for a particular advisor. Segment advice can be generated at the object, segment, or tablespace level.

Syntax

```
DBMS_ADVISOR.UPDATE_OBJECT (
   task_name        IN VARCHAR2
   object_id        IN NUMBER,
   attr1            IN VARCHAR2 := NULL,
   attr2            IN VARCHAR2 := NULL,
   attr3            IN VARCHAR2 := NULL,
   attr4            IN CLOB := NULL,
   attr5            IN VARCHAR2 := NULL);
```

Parameters

Table 18-41 UPDATE_OBJECT Procedure Parameters

Parameter	Description
`task_name`	A valid advisor task name that uniquely identifies an existing task.
`object_id`	The advisor-assigned object identifier.
`attr1`	Advisor-specific data. If set to `NULL`, there will be no effect on the target object.
`attr2`	Advisor-specific data. If set to `NULL`, there will be no effect on the target object.
`attr3`	Advisor-specific data. If set to `NULL`, there will be no effect on the target object.
`attr4`	Advisor-specific data. If set to `NULL`, there will be no effect on the target object.
`attr5`	Advisor-specific data. If set to null, there will be no effect on the target object.

The attribute parameters have different values depending upon the object type. See *Oracle Database Administrator's Guide* for details regarding these parameters and object types.

Usage Notes

If for the object level, advice is generated on all partitions of the object (if the object is partitioned). The advice is not cascaded to any dependent objects. If for the segment level, advice can be obtained on a single segment, such as the partition or subpartition of a table, index, or lob column. If for a tablespace level, target advice for every segment in the tablespace will be generated.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  obj_id NUMBER;
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.CREATE_OBJECT (task_name,'SQL',NULL,NULL,NULL,
                             'SELECT * FROM SH.SALES',obj_id);
  DBMS_ADVISOR.UPDATE_OBJECT (task_name, obj_id,NULL,NULL,NULL,
                             'SELECT count(*) FROM SH.SALES');
END;
/
```

See Also:

Oracle Database Administrator's Guide for further information regarding the Segment Advisor

UPDATE_REC_ATTRIBUTES Procedure

This procedure updates the owner, name, and tablespace for a recommendation.

Syntax

```
DBMS_ADVISOR.UPDATE_REC_ATTRIBUTES (
   task_name               IN VARCHAR2
   rec_id                  IN NUMBER,
   action_id               IN NUMBER,
   attribute_name          IN VARCHAR2,
   value                   IN VARCHAR2);
```

Parameters

Table 18-42 UPDATE_REC_ATTRIBUTES Procedure Parameters

Parameter	Description
`task_name`	The task name that uniquely identifies an existing task.
`rec_id`	The Advisor-generated identifier number that is assigned to the recommendation.
`action_id`	The Advisor-generated action identifier that is assigned to the particular command.
`attribute_name`	Name of the attribute to be changed. The valid values are: • `owner` The new owner of the object. • `name` The new name of the object. • `tablespace` The new tablespace for the object.
`value`	Specifies the new value for the recommendation attribute.

Usage Notes

Recommendation attributes cannot be modified unless the task has successfully executed.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
  workload_name VARCHAR2(30);
  attribute VARCHAR2(100);
BEGIN
  task_name := 'My Task';
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_REF(task_name, workload_name);
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                   100,400,5041,103,640445,680000,2,
                                   1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                   FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.EXECUTE_TASK(task_name);

attribute := 'SH';

  DBMS_ADVISOR.UPDATE_REC_ATTRIBUTES(task_name, 1, 3, 'OWNER', attribute);
END;
/
```

UPDATE_SQLWKLD_ATTRIBUTES Procedure

This procedure changes various attributes of a SQL Workload object or template.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.UPDATE_SQLWKLD_ATTRIBUTES (
   workload_name         IN VARCHAR2,
   new_name              IN VARCHAR2 := NULL,
   description           IN VARCHAR2 := NULL,
   read_only             IN VARCHAR2 := NULL,
   is_template           IN VARCHAR2 := NULL,
   how_created           IN VARCHAR2 := NULL);
```

Parameters

Table 18-43 UPDATE_SQLWKLD_ATTRIBUTES Procedure Parameters

Parameter	Description
`workload_name`	The workload object name that uniquely identifies an existing workload.
`new_name`	The new workload object name. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, the workload will not be renamed. A task name can be up to 30 characters long.
`description`	A new workload description. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, the description will not be changed. Names can be up to 256 characters long.
`read_only`	Set to `TRUE` so it cannot be changed.
`is_template`	`TRUE` if workload is to be used as a template.

Table 18-43 (Cont.) UPDATE_SQLWKLD_ATTRIBUTES Procedure Parameters

Parameter	Description
`how_created`	Indicates a source application name that initiated the workload creation. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, the source will not be changed.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                   100,400,5041,103,640445,680000,2,
                                   1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                   FROM sh.sales WHERE promo_id = 10');
  DBMS_ADVISOR.UPDATE_SQLWKLD_ATTRIBUTES(workload_name,'New workload name');
END;
/
```

UPDATE_SQLWKLD_STATEMENT Procedure

This procedure updates an existing SQL statement in a specified SQL workload.

Note:

This procedure is deprecated starting in Oracle Database 11*g*.

Syntax

```
DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT (
   workload_name      IN VARCHAR2,
   sql_id             IN NUMBER,
   application        IN VARCHAR2 := NULL,
   action             IN VARCHAR2 := NULL,
   priority           IN NUMBER := NULL,
   username           IN VARCHAR2 := NULL);

DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT (
   workload_name      IN VARCHAR2,
   search             IN VARCHAR2,
   updated            OUT NUMBER,
   application        IN VARCHAR2 := NULL,
   action             IN VARCHAR2 := NULL,
   priority           IN NUMBER := NULL,
   username           IN VARCHAR2 := NULL);
```

Parameters

Table 18-44 UPDATE_SQLWKLD_STATEMENT Procedure Parameters

Parameter	Description
`workload_name`	The SQL Workload object name that uniquely identifies an existing workload.
`sql_id`	The Advisor-generated identifier number that is assigned to the statement. To specify all workload statements, use the constant `DBMS_ADVISOR.ADVISOR_ALL`.
`updated`	Returns the number of statements changed by a searched update.
`application`	Specifies a business application name that will be associated with the SQL statement. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, then the column will not be updated in the repository.
`action`	Specifies the application action for the statement. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, then the column will not be updated in the repository.
`priority`	The relative priority of the SQL statement. The value must be one of the following: 1 - `HIGH`, 2 - `MEDIUM`, or 3 - `LOW`. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, then the column will not be updated in the repository.
`username`	The Oracle user name that executed the SQL statement. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, then the column will not be updated in the repository. Because a user name is an Oracle identifier, the `username` value must be entered exactly like it is stored in the database. For example, if the user `SCOTT` is the executing user, then you must provide the user identifier `SCOTT` in all uppercase letters. The database does not recognize the user `scott` as a match for `SCOTT`.
`search`	Disabled.

Usage Notes

A workload cannot be modified or deleted if it is currently referenced by an active task. A task is considered active if it is not in its initial state. See RESET_TASK Procedure to set a task to its initial state.

Examples

```
DECLARE
  workload_name VARCHAR2(30);
  updated NUMBER;
  id NUMBER;
BEGIN
  workload_name := 'My Workload';

  DBMS_ADVISOR.CREATE_SQLWKLD(workload_name, 'My Workload');
  DBMS_ADVISOR.ADD_SQLWKLD_STATEMENT(workload_name, 'MONTHLY', 'ROLLUP',
                                   100,400,5041,103,640445,680000,2,
                                   1,SYSDATE,1,'SH','SELECT AVG(amount_sold)
                                   FROM sh.sales WHERE promo_id = 10');

   SELECT sql_id INTO id FROM USER_ADVISOR_SQLW_STMTS
   WHERE workload_name = 'My Workload';

  DBMS_ADVISOR.UPDATE_SQLWKLD_STATEMENT(workload_name, id);
END;
/
```

UPDATE_TASK_ATTRIBUTES Procedure

This procedure changes various attributes of a task or a task template.

Syntax

```
DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES (
   task_name          IN VARCHAR2
   new_name           IN VARCHAR2 := NULL,
   description        IN VARCHAR2 := NULL,
   read_only          IN VARCHAR2 := NULL,
   is_template        IN VARCHAR2 := NULL,
   how_created        IN VARCHAR2 := NULL);
```

Parameters

Table 18-45 UPDATE_TASK_ATTRIBUTES Procedure Parameters

Parameter	Description
`task_name`	The Advisor task name that uniquely identifies an existing task.
`new_name`	The new Advisor task name. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, the task will not be renamed. A task name can be up to 30 characters long.
`description`	A new task description. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, the description will not be changed. Names can be up to 256 characters long.
`read_only`	Sets the task to read-only. Possible values are: `TRUE` and `FALSE`. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, the setting will not be changed.
`is_template`	Marks the task as a template. Physically, there is no difference between a task and a template; however, a template cannot be executed. Possible values are: `TRUE` and `FALSE`. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, the setting will not be changed.
`how_created`	Indicates a source application name that initiated the task creation. If the value is `NULL` or contains the value `ADVISOR_UNUSED`, the source will not be changed.

Examples

```
DECLARE
  task_id NUMBER;
  task_name VARCHAR2(30);
BEGIN
  task_name := 'My Task';

  DBMS_ADVISOR.CREATE_TASK(DBMS_ADVISOR.SQLACCESS_ADVISOR, task_id,
task_name);
  DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES(task_name,'New Task Name');
  DBMS_ADVISOR.UPDATE_TASK_ATTRIBUTES('New Task Name',NULL,'New description');
END;
/
```