16 DBMS_ADDM

The DBMS_ADDM package facilitates the use of Advisor functionality regarding the Automatic Database Diagnostic Monitor.

Note:

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

This chapter contains the following topics:

- Security Model
- Summary of DBMS_ADDM Subprograms

See Also:

- *Oracle Real Application Clusters Administration and Deployment Guide* for more information about "Automatic Workload Repository in Oracle Real Application Clusters Environments"
- *Oracle Database Performance Tuning Guide* for more information about "Automatic Performance Diagnostics"

DBMS_ADDM Security Model

The DBMS_ADDM package runs with the caller's permission, not the definer's, and then applies the security constraints required by the DBMS_ADVISOR package.

See Also:

The DBMS_ADVISOR package for more information about "Security Model".

Summary of DBMS_ADDM Subprograms

The table in this topic lists and describes the DBMS_ADDM subprograms.

Table 16-1 DBMS_ADDM Package Subprograms

Subprogram	Description
ANALYZE_DB Procedure	Creates an ADDM task for analyzing in database analysis mode and executes it
ANALYZE_INST Procedure	Creates an ADDM task for analyzing in instance analysis mode and executes it.
ANALYZE_PARTIAL Procedure	Creates an ADDM task for analyzing a subset of instances in partial analysis mode and executes it
COMPARE_CAPTURE_REPLAY_REPORT Function	Produces a Compare Period ADDM report comparing the performance of a capture to a replay
COMPARE_DATABASES Function	Produces a Compare Period ADDM report for a database-wide performance comparison
COMPARE_INSTANCES Function	Produces a Compare Period ADDM report for an instance-level performance comparison
COMPARE_REPLAY_REPLAY_REPORT Function	Produces a Compare Period ADDM report comparing the performance of a replay to another replay
DELETE Procedure	Deletes an already created ADDM task (of any kind)
DELETE_FINDING_DIRECTIVE Procedure	Deletes a finding directive
DELETE_PARAMETER_DIRECTIVE Procedure	Deletes a parameter directive
DELETE_SEGMENT_DIRECTIVE Procedure	Deletes a segment directive
DELETE_SQL_DIRECTIVE Procedure	Deletes a SQL directive
FAILED_AUTO_TASKS_REPORT Function	Creates a plain text, user-readable report for the failed tasks registered in the `DBA_ADDM_PENDING_AUTOTASKS` views.
GET_ASH_QUERY Function	Returns a string containing the SQL text of an ASH query identifying the rows in ASH with impact for the finding
GET_REPORT Function	Retrieves the default text report of an executed ADDM task
INSERT_FINDING_DIRECTIVE Procedure	Creates a directive to limit reporting of a specific finding type.
INSERT_PARAMETER_DIRECTIVE Procedure	Creates a directive to prevent ADDM from creating actions to alter the value of a specific system parameter
INSERT_SEGMENT_DIRECTIVE Procedure	Creates a directive to prevent ADDM from creating actions to "run Segment Advisor" for specific segments
INSERT_SQL_DIRECTIVE Procedure	Creates a directive to limit reporting of actions on specific SQL
REAL_TIME_ADDM_REPORT Function	Produces a real-time report of ADDM activity
REEXECUTE_FAILED_AUTO_TASKS Procedure	Re-executes the tasks registered in the `DBA_ADDM_PENDING_AUTOTASKS` views.

ANALYZE_DB Procedure

This procedure creates an ADDM task for analyzing in database analysis mode and executes it.

Syntax

```
DBMS_ADDM.ANALYZE_DB (
   task_name                 IN OUT VARCHAR2,
   begin_snapshot            IN     NUMBER,
   end_snapshot              IN     NUMBER,
   read_only_type_override   IN     VARCHAR2,
   db_id                     IN     NUMBER := NULL);
```

Parameters

Table 16-2 ANALYZE_DB Procedure Parameters

Parameter	Description
`task_name`	Name of the task to be created.
`begin_snapshot`	Number of the snapshot that starts the analysis period.
`end_snapshot`	Number of the snapshot that ends the analysis period.
`read_only_type_override`	Overrides the type of `CDB` `ADDM` determines for analysis. The possible values are: • `READ-WRITE`—a regular database or the primary database in a data guard configuration • `READ-ONLY`—a database open in read-only mode, such as an active data guard standby • `AUTO`—allows `ADDM` to decide the type of `CDB` to override based on the data
`db_id`	Database ID for the database you to analyze. By default, this is the database currently connected.

Return Values

The name of the created task is returned in the `task_name` parameter. It may be different from the value that is given as input (only in cases that name is already used by another task).

Examples

To create an ADDM task in database analysis mode and execute it, with its name in variable `tname`:

```
var tname VARCHAR2(60);
BEGIN
  :tname := 'my_database_analysis_mode_task';
  DBMS_ADDM.ANALYZE_DB(:tname, 1, 2);
END
```

To see a report:

```
SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

Note that the return type of a report is a `CLOB`, formatted to fit line size of 80.

ANALYZE_INST Procedure

This procedure creates an ADDM task for analyzing in instance analysis mode and executes it.

Syntax

```
DBMS_ADDM.ANALYZE_INST (
   task_name                 IN OUT VARCHAR2,
   begin_snapshot            IN     NUMBER,
   end_snapshot              IN     NUMBER,
   cdb_type_override         IN     VARCHAR2,
   read_only_type_override   IN     VARCHAR2,
   instance_number           IN     NUMBER := NULL,
   db_id                     IN     NUMBER := NULL);
```

Parameters

Table 16-3 ANALYZE_INST Procedure Parameters

Parameter	Description
`task_name`	Name of the task to be created
`begin_snapshot`	Number of the snapshot that starts the analysis period
`end_snapshot`	Number of the snapshot that ends the analysis period
`cdb_type_override`	Overrides the type of `CDB` that `ADDM` determines for doing analysis. The possible values are: • `AUTONOMOUS OLTP`—autonomous OLTP inside a PDB • `AUTONOMOUS DATA WAREHOUSE`—autonomous data warehouse (ADWH) inside a PDB • PDB —a regular PDB • CDB ROOT —the root of a CDB • NON-CDB —a system that is not CDB or PDB • AUTO —allows `ADDM` to decide the type of `CDB` to override based on the data
`read_only_type_override`	Overrides the type of `CDB` `ADDM` determines for analysis. The possible values are: • `READ-WRITE`—a regular database or the primary database in a data guard configuration • `READ-ONLY`—a database open in read-only mode, such as an active data guard standby • `AUTO`—allows `ADDM` to decide the type of `CDB` to override based on the data
`instance_number`	Number of the instance to analyze. By default it is the instance currently connected
`db_id`	Database ID for the database you to analyze. By default, this is the database currently connected

Return Values

The name of the created task is returned in the `task_name` parameter. It may be different from the value that is given as input (only in cases that name is already used by another task).

Usage Notes

On single instance systems (when not using Oracle RAC) the resulting task is identical to using the `ANALYZE_DB` procedure.

Examples

To create an ADDM task in instance analysis mode and execute it, with its name in variable `tname`:

```
var tname VARCHAR2(60);
BEGIN
  :tname := 'my_instance_analysis_mode_task';
  DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END
```

To see a report:

```
SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

Note that the return type of a report is a `CLOB`, formatted to fit line size of 80.

ANALYZE_PARTIAL Procedure

This procedure creates an ADDM task for analyzing a subset of instances in partial analysis mode and executes it.

Syntax

```
DBMS_ADDM.ANALYZE_PARTIAL (
   task_name                 IN OUT VARCHAR2,
   instance_numbers          IN     VARCHAR2,
   begin_snapshot            IN     NUMBER,
   end_snapshot              IN     NUMBER,
   cdb_type_override         IN     VARCHAR2,
   read_only_type_override   IN     VARCHAR2,
   db_id                     IN     NUMBER := NULL);
```

Parameters

Table 16-4 ANALYZE_PARTIAL Procedure Parameters

Parameter	Description
`task_name`	Name of the task to be created
`instance_numbers`	Comma separated list of instance numbers to analyze
`begin_snapshot`	Number of the snapshot that starts the analysis period
`end_snapshot`	Number of the snapshot that ends the analysis period

Table 16-4 (Cont.) ANALYZE_PARTIAL Procedure Parameters

Parameter	Description
`cdb_type_override`	Overrides the type of `CDB` that `ADDM` determines for doing analysis. The possible values are: • `AUTONOMOUS OLTP`—autonomous OLTP inside a PDB • `AUTONOMOUS DATA WAREHOUSE`—autonomous data warehouse (ADWH) inside a PDB • PDB —a regular PDB • CDB ROOT —the root of a CDB • NON-CDB —a system that is not CDB or PDB • AUTO —allows `ADDM` to decide the type of `CDB` to override based on the data
`read_only_type_override`	Overrides the type of `CDB` `ADDM` determines for analysis. The possible values are: • `READ-WRITE`—a regular database or the primary database in a data guard configuration • `READ-ONLY`—a database open in read-only mode, such as an active data guard standby • `AUTO`—allows `ADDM` to decide the type of `CDB` to override based on the data
`db_id`	Database ID for the database you to analyze. By default, this is the database currently connected

Return Values

The name of the created task is returned in the `task_name` parameter. It may be different from the value that is given as input (only in cases that name is already used by another task).

Examples

To create an ADDM task in partial analysis mode and execute it, with its name in variable `tname`:

```
var tname VARCHAR2(60);
BEGIN
  :tname := 'my_partial_analysis_modetask';
  DBMS_ADDM.ANALYZE_PARTIAL(:tname, '1,2,3', 1, 2);
END
```

To see a report:

```
SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

Note that the return type of a report is a `CLOB`, formatted to fit line size of 80.

COMPARE_CAPTURE_REPLAY_REPORT Function

This function produces a Compare Period ADDM report comparing the performance of a capture to a replay.

The AWR data must reside in the same database, but it can originate from different databases. The function generates a report in either XML or HTML(Active Report) format.

Syntax

```
DBMS_ADDM.COMPARE_CAPTURE_REPLAY_REPORT (
   replay_id                 IN NUMBER,
   cdb_type_override         IN VARCHAR2,
   read_only_type_override   IN VARCHAR2,
   report_type               IN VARCHAR2 := 'HTML')
  RETURN CLOB;
```

Parameters

Table 16-5 COMPARE_CAPTURE_REPLAY_REPORT Function Parameters

Parameter	Description
`replay_id`	Replay ID to use as the base period. The base period is the baseline period to compare in order to determine improvement or regression.
`cdb_type_override`	Overrides the type of `CDB` that `ADDM` determines for doing analysis. The possible values are: • `AUTONOMOUS OLTP`—autonomous OLTP inside a PDB • `AUTONOMOUS DATA WAREHOUSE`—autonomous data warehouse (ADWH) inside a PDB • PDB —a regular PDB • CDB ROOT —the root of a CDB • NON-CDB —a system that is not CDB or PDB • AUTO —allows `ADDM` to decide the type of `CDB` to override based on the data
`read_only_type_override`	Overrides the type of `CDB` `ADDM` determines for analysis. The possible values are: • `READ-WRITE`—a regular database or the primary database in a data guard configuration • `READ-ONLY`—a database open in read-only mode, such as an active data guard standby • `AUTO`—allows `ADDM` to decide the type of `CDB` to override based on the data
`report_type`	`HTML` (the default) for an HTML active report, `'XML'` for an XML report

Return Values

A `CLOB` containing a compare period ADDM report

COMPARE_DATABASES Function

This function produces a Compare Period ADDM report comparing the performance of a database over two different time periods or the performance of two different databases over two different time periods.

The AWR data must reside in the same database, but it can originate from different databases. The function generates a report in either XML or HTML(Active Report) format.

Syntax

```
DBMS_ADDM.COMPARE_DATABASES (
   base_dbid             IN NUMBER := NULL,
   base_begin_snap_id    IN NUMBER,
   base_end_snap_id      IN NUMBER,
   comp_dbid             IN NUMBER := NULL,
   comp_begin_snap_id    IN NUMBER,
   comp_end_snap_id      IN NUMBER,
   cdb_type_override         IN      VARCHAR2,
   read_only_type_override   IN      VARCHAR2,
   report_type           IN VARCHAR2 := 'HTML')
  RETURN CLOB;
```

Parameters

Table 16-6 COMPARE_DATABASES Function Parameters

Parameter	Description
`base_dbid`	Database id (`DBID`) of the base period. The base period is the baseline period that we compare to in order to determine improvement or regression.
`base_begin_snap_ids`	Begin AWR snapshot ID of the base period.
`base_end_snap_id`	End AWR snapshot ID of the base period.
`comp_dbid`	Database id (`DBID`) of the comparison period. The comparison period is the period we compare to the base period.
`comp_begin_snap_id`	Begin AWR snapshot ID of the comparison period
`comp_end_snap_id`	End AWR snapshot ID of the comparison period
`cdb_type_override`	Overrides the type of `CDB` that `ADDM` determines for doing analysis. The possible values are: • `AUTONOMOUS OLTP`—autonomous OLTP inside a PDB • `AUTONOMOUS DATA WAREHOUSE`—autonomous data warehouse (ADWH) inside a PDB • PDB —a regular PDB • CDB ROOT —the root of a CDB • NON-CDB —a system that is not CDB or PDB • AUTO —allows `ADDM` to decide the type of `CDB` to override based on the data

Table 16-6 (Cont.) COMPARE_DATABASES Function Parameters

Parameter	Description
`read_only_type_override`	Overrides the type of `CDB ADDM` determines for analysis. The possible values are: • `READ-WRITE`—a regular database or the primary database in a data guard configuration • `READ-ONLY`—a database open in read-only mode, such as an active data guard standby • `AUTO`—allows `ADDM` to decide the type of `CDB` to override based on the data
`report_type`	`'HTML'` (the default) for an HTML active report, `'XML'` for an XML report

Return Values

A `CLOB` containing a compare period ADDM report

COMPARE_INSTANCES Function

This function produces a Compare Period ADDM report comparing the performance of a single instance over two different time periods or the performance of two different instances over two different time periods.

The AWR data must reside in the same database, but it can originate from different databases. The function generates a report in either XML or HTML(Active Report) format.

Syntax

```
DBMS_ADDM.COMPARE_INSTANCES (
   base_dbid                  IN NUMBER := NULL,
   base_instance_id           IN NUMBER,
   base_begin_snap_id         IN NUMBER,
   base_end_snap_id           IN NUMBER,
   comp_dbid                  IN NUMBER := NULL,
   comp_instance_id           IN NUMBER,
   comp_begin_snap_id         IN NUMBER,
   comp_end_snap_id           IN NUMBER,
   cdb_type_override          IN VARCHAR2,
   read_only_type_override    IN VARCHAR2,
   report_type                IN VARCHAR2 := 'HTML')
  RETURN CLOB;
```

Parameters

Table 16-7 COMPARE_INSTANCES Function Parameters

Parameter	Description
`base_dbid`	Database id (`DBID`) of the base period. The base period is the baseline period that we compare to in order to determine improvement or regression.
`base_instance_id`	Instance number of the database instance to include from the base period
`base_begin_snap_id`	Begin AWR snapshot ID of the base period.

Table 16-7 (Cont.) COMPARE_INSTANCES Function Parameters

Parameter	Description
`base_end_snap_id`	End AWR snapshot ID of the base period.
`comp_dbid`	Database id (`DBID`) of the comparison period. The comparison period is the period we compare to the base period.
`comp_instance_id`	Instance number of the database instance to include from the comparison period
`comp_begin_snap_id`	Begin AWR snapshot ID of the comparison period
`comp_end_snap_id`	End AWR snapshot ID of the comparison period
`cdb_type_override`	Overrides the type of `CDB` that `ADDM` determines for doing analysis. The possible values are: • `AUTONOMOUS OLTP`—autonomous OLTP inside a PDB • `AUTONOMOUS DATA WAREHOUSE`—autonomous data warehouse (ADWH) inside a PDB • PDB —a regular PDB • CDB ROOT —the root of a CDB • NON-CDB —a system that is not CDB or PDB • AUTO —allows `ADDM` to decide the type of `CDB` to override based on the data
`read_only_type_override`	Overrides the type of `CDB` `ADDM` determines for analysis. The possible values are: • `READ-WRITE`—a regular database or the primary database in a data guard configuration • `READ-ONLY`—a database open in read-only mode, such as an active data guard standby • `AUTO`—allows `ADDM` to decide the type of `CDB` to override based on the data
`report_type`	`'HTML'` (the default) for an HTML active report, `'XML'` for an XML report

Return Values

A `CLOB` containing a compare period ADDM report

COMPARE_REPLAY_REPLAY_REPORT Function

This function produces a Compare Period ADDM report comparing the performance of a replay to another replay.

The AWR data must reside in the same database, but it can originate from different databases. The function generates a report in either XML or HTML(Active Report) format.

Syntax

```
DBMS_ADDM.COMPARE_CAPTURE_REPLAY_REPORT (
   replay_id1                IN     NUMBER,
   replay_id2                IN     NUMBER,
   cdb_type_override          IN     VARCHAR2,
   read_only_type_override    IN     VARCHAR2,
   report_type                IN     VARCHAR2 := 'HTML')
  RETURN CLOB;
```

Parameters

Table 16-8 COMPARE_REPLAY_REPLAY_REPORT Function Parameters

Parameter	Description
`replay_id1`	Replay ID to use as the base period. The base period is the baseline period to compare in order to determine improvement or regression.
`replay_id2`	Replay ID to use as the comparison period. The comparison period is the period to compare to the base period in order to determine improvement or regression.
`cdb_type_override`	Overrides the type of `CDB` that `ADDM` determines for doing analysis. The possible values are: • `AUTONOMOUS OLTP`—autonomous OLTP inside a PDB • `AUTONOMOUS DATA WAREHOUSE`—autonomous data warehouse (ADWH) inside a PDB • PDB —a regular PDB • CDB ROOT —the root of a CDB • NON-CDB —a system that is not CDB or PDB • AUTO —allows `ADDM` to decide the type of `CDB` to override based on the data
`read_only_type_override`	Overrides the type of `CDB` `ADDM` determines for analysis. The possible values are: • `READ-WRITE`—a regular database or the primary database in a data guard configuration • `READ-ONLY`—a database open in read-only mode, such as an active data guard standby • `AUTO`—allows `ADDM` to decide the type of `CDB` to override based on the data
`report_type`	`'HTML'` (the default) for an HTML active report, `'XML'` for an XML report

Return Values

A `CLOB` containing a compare period ADDM report

DELETE Procedure

This procedure deletes an already created ADDM task (of any kind). For database analysis mode and partial analysis mode this deletes the local tasks associated with the main task.

Syntax

```
DBMS_ADDM.DELETE (
   task_name           IN VARCHAR2);
```

Parameters

Table 16-9 DELETE Procedure Parameters

Parameter	Description
`task_name`	Name of the task to be deleted

Examples

```
BEGIN
  DBMS_ADDM.DELETE ('my_partial_analysis_mode_task');
END
```

DELETE_FINDING_DIRECTIVE Procedure

This procedure deletes a finding directive.

Syntax

```
DBMS_ADDM.DELETE_FINDING_DIRECTIVE (
   task_name           IN VARCHAR2,
   dir_name            IN VARCHAR2);
```

Parameters

Table 16-10 DELETE_FINDING_DIRECTIVE Procedure Parameters

Parameter	Description
`task_name`	Name of the task this directive applies to. If the value is `NULL`, it is a system directive.
`dir_name`	Name of the directive. All directives must be given unique names.

DELETE_PARAMETER_DIRECTIVE Procedure

This procedure deletes a parameter directive. This removes a specific system directive for parameters. Subsequent ADDM tasks are not affected by this directive.

Syntax

```
DBMS_ADDM.DELETE_PARAMETER_DIRECTIVE (
   task_name           IN VARCHAR2,
   dir_name            IN VARCHAR2);
```

Parameters

Table 16-11 DELETE_PARAMETER_DIRECTIVE Procedure Parameters

Parameter	Description
`task_name`	Name of the task this directive applies to. If the value is `NULL`, it is a system directive.
`dir_name`	Name of the directive. All directives must be given unique names.

Examples

```
BEGIN
   DBMS_ADDM.DELETE_PARAMETER_DIRECTIVE (NULL,'my Parameter directive');
END;
```

DELETE_SEGMENT_DIRECTIVE Procedure

This procedure deletes a segment directive.

Syntax

```
DBMS_ADDM.DELETE_SEGMENT_DIRECTIVE (
   task_name            IN VARCHAR2,
   dir_name             IN VARCHAR2);
```

Parameters

Table 16-12   DELETE_SEGMENT_DIRECTIVE Procedure Parameters

Parameter	Description
`task_name`	Name of the task this directive applies to. If the value is `NULL`, it is a system directive.
`dir_name`	Name of the directive. All directives must be given unique names.

DELETE_SQL_DIRECTIVE Procedure

This procedure deletes a SQL directive.

Syntax

```
DBMS_ADDM.DELETE_SQL_DIRECTIVE (
   task_name            IN VARCHAR2,
   dir_name             IN VARCHAR2);
```

Parameters

Table 16-13   DELETE_SQL_DIRECTIVE Procedure Parameters

Parameter	Description
`task_name`	Name of the task this directive applies to. If the value is `NULL`, it is a system directive.
`dir_name`	Name of the directive. All directives must be given unique names.

FAILED_AUTO_TASKS_REPORT Function

This function creates a plain text, user-readable report for the failed tasks registered in the `DBA_ADDM_PENDING_AUTOTASKS` views.

Syntax

```
DBMS_ADDM.FAILED_AUTO_TASKS_REPORT(
   instance_number      IN   NUMBER :=NULL,
   begin_snapshot       IN   NUMBER :=NULL,
   end_snapshot         IN   NUMBER :=NULL,
   dbid                 IN   NUMBER :=NULL)
```

Parameters

Table 16-14 FAILED_AUTO_TASKS_REPORT Function Parameters

Parameter	Description
`instance_number`	Instance number for the tasks to be reported.
`begin_snapshot`	Earliest begin snapshot ID for the tasks to be reported.
`end_snapshot`	Latest end snapshot ID for the tasks to be reported.
`dbid`	Database ID for the tasks to be reported.

GET_ASH_QUERY Function

The function returns a string containing the SQL text of an ASH query identifying the rows in ASH with impact for the finding.

For most types of findings this identifies the exact rows in ASH corresponding to the finding. For some types of findings the query is an approximation and should not be used for exact identification of the finding's impact or the finding's specific activity.

Syntax

```
DBMS_ADDM.GET_ASH_QUERY (
   task_name           IN    VARCHAR2,
   finding_id          IN    NUMBER)
  RETURN VARCHAR2;
```

Parameters

Table 16-15 GET_ASH_QUERY Function Parameters

Parameter	Description
`task_name`	Name of the task
`finding`	ID of the finding within the task

Return Values

A `VARCHAR` containing an ASH query identifying the rows in ASH with impact for the finding

GET_REPORT Function

This function retrieves the default text report of an executed ADDM task.

Syntax

```
DBMS_ADDM.GET_REPORT (
   task_name           IN VARCHAR2)
  RETURN CLOB;
```

Parameters

Table 16-16   GET_REPORT Function Parameters

Parameter	Description
`task_name`	Name of the task

Examples

```
Set long 1000000
Set pagesize 50000
SELECT DBMS_ADDM.GET_REPORT('my_partial_analysis_mode_task') FROM DUAL;
```

INSERT_FINDING_DIRECTIVE Procedure

This procedure creates a directive to limit reporting of a specific finding type. The directive can be created for a specific task (only when the task is in `INITIAL` status), or for all subsequently created ADDM tasks (such as a system directive).

Syntax

```
DBMS_ADDM.INSERT_FINDING_DIRECTIVE (
   task_name              IN VARCHAR2,
   dir_name               IN VARCHAR2,
   finding_name           IN VARCHAR2,
   min_active_sessions    IN NUMBER := 0,
   min_perc_impact        IN NUMBER := 0);
```

Parameters

Table 16-17   INSERT_FINDING_DIRECTIVE Procedure Parameters

Parameter	Description
`task_name`	Name of the task this directive applies to. If the value is `NULL`, it applies to all subsequently created ADDM Tasks.
`dir_name`	Name of the directive. All directives must be given unique names.
`finding_name`	Name of an ADDM finding to which this directive applies. All valid findings names appear in the `NAME` column of view `DBA_ADVISOR_FINDING_NAMES`.
`min_active_sessions`	Minimal number of active sessions for the finding. If a finding has less than this number, it is filtered from the ADDM result.

Table 16-17 (Cont.) INSERT_FINDING_DIRECTIVE Procedure Parameters

Parameter	Description
`min_perc_impact`	Minimal number for the "percent impact" of the finding relative to total database time in the analysis period. If the finding's impact is less than this number, it is filtered from the ADDM result.

Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for 'Undersized SGA' findings. The result of `GET_REPORT` shows only an 'Undersized SGA' finding if the finding is responsible for at least 2 average active sessions during the analysis period, and this constitutes at least 10% of the total database time during that period.

```
var tname VARCHAR2(60);
BEGIN
  DBMS_ADDM.INSERT_FINDING_DIRECTIVE(
   NULL,
   'Undersized SGA directive',
   'Undersized SGA',
   2,
   10);
  :tname := 'my_instance_analysis_mode_task';
  DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
```

To see a report containing 'Undersized SGA' findings regardless of the directive:

```
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```

INSERT_PARAMETER_DIRECTIVE Procedure

This procedure creates a directive to prevent ADDM from creating actions to alter the value of a specific system parameter. The directive can be created for a specific task (only when the task is in `INITIAL` status), or for all subsequently created ADDM tasks (such as a system directive).

Syntax

```
DBMS_ADDM.INSERT_PARAMETER_DIRECTIVE (
   task_name              IN VARCHAR2,
   dir_name               IN VARCHAR2,
   parameter_name         IN VARCHAR2);
```

Parameters

Table 16-18 INSERT_PARAMETER_DIRECTIVE Procedure Parameters

Parameter	Description
`task_name`	Name of the task this directive applies to. If the value is `NULL`, it applies to all subsequently created ADDM Tasks.
`dir_name`	Name of the directive. All directives must be given unique names.
`parameter_name`	Specifies the parameter to use. Valid parameter names appear in `V$PARAMETER`.

Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for all actions that recommend modifying the parameter '`sga_target`'. The result of `GET_REPORT` does not show these actions.

```
var tname varchar2(60);
BEGIN
  DBMS_ADDM.INSERT_PARAMETER_DIRECTIVE(
   NULL,
   'my Parameter directive',
   'sga_target');
  :tname := 'my_instance_analysis_mode_task';
  DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
```

To see a report containing all actions regardless of the directive:

```
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```

INSERT_SEGMENT_DIRECTIVE Procedure

This procedure creates a directive to prevent ADDM from creating actions to "run Segment Advisor" for specific segments. The directive can be created for a specific task (only when the task is in `INITIAL` status), or for all subsequently created ADDM tasks (such as a system directive).

Syntax

```
DBMS_ADDM.INSERT_SEGMENT_DIRECTIVE (
   task_name              IN VARCHAR2,
   dir_name               IN VARCHAR2,
   owner_name             IN VARCHAR2,
   object_name            IN VARCHAR2 := NULL,
   sub_object_name        IN VARCHAR2 := NULL);


DBMS_ADDM.INSERT_SEGMENT_DIRECTIVE (
   task_name              IN VARCHAR2,
   dir_name               IN VARCHAR2,
   object_number          IN NUMBER);
```

Parameters

Table 16-19 INSERT_SEGMENT_DIRECTIVE Procedure Parameters

Parameter	Description
`task_name`	Name of the task this directive applies to. If the value is `NULL`, it applies to all subsequently created ADDM Tasks.
`dir_name`	Name of the directive. All directives must be given unique names.
`owner_name`	Specifies the owner of the segment/s to be filtered. A wildcard is allowed in the same syntax used for "like" constraints.
`object_name`	Name of the main object to be filtered. Again, wildcards are allowed. The default value of `NULL` is equivalent to a value of '%'.

Table 16-19 (Cont.) INSERT_SEGMENT_DIRECTIVE Procedure Parameters

Parameter	Description
`sub_object_name`	Name of the part of the main object to be filtered. This could be a partition name, or even sub partitions (separated by a '.'). Again, wildcards are allowed. The default value of `NULL` is equivalent to a value of '%'.
`object_number`	Object number of the `SEGMENT` that this directive is to filter, found in views `DBA_OBJECTS` or `DBA_SEGMENTS`

Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for all segments that belong to user `SCOTT`. The result of `GET_REPORT` does not show actions for running Segment advisor for segments that belong to `SCOTT`.

```
var tname VARCHAR2(60);
BEGIN
  DBMS_ADDM.INSERT_SEGMENT_DIRECTIVE(NULL,
                                     'my Segment directive',
                                     'SCOTT');
  :tname := 'my_instance_analysis_mode_task';
  DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
```

To see a report containing all actions regardless of the directive:

```
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```

INSERT_SQL_DIRECTIVE Procedure

This procedure creates a directive to limit reporting of actions on specific SQL. The directive can be created for a specific task (only when the task is in `INITIAL` status), or for all subsequently created ADDM tasks (such as a system directive).

Syntax

```
DBMS_ADDM.INSERT_SQL_DIRECTIVE (
   task_name              IN VARCHAR2,
   dir_name               IN VARCHAR2,
   sql_id                 IN VARCHAR2,
   min_active_sessions    IN NUMBER := 0,
   min_response_time      IN NUMBER := 0);
```

Parameters

Table 16-20 INSERT_SQL_DIRECTIVE Procedure Parameters

Parameter	Description
`task_name`	Name of the task this directive applies to. If the value is `NULL`, it applies to all subsequently created ADDM Tasks.
`dir_name`	Name of the directive. All directives must be given unique names.
`sql_id`	Identifies which SQL statement to filter. A valid value contains exactly 13 characters from '0' to '9' and 'a' to 'z'.

Table 16-20 (Cont.) INSERT_SQL_DIRECTIVE Procedure Parameters

Parameter	Description
`min_active_sessions`	Minimal number of active sessions for the SQL. If a SQL action has less than this number, it is filtered from the ADDM result.
`min_response_time`	Minimal value for response time of the SQL (in microseconds). If the SQL had lower response time, it is filtered from the ADDM result.

Examples

A new ADDM task is created to analyze a local instance. However, it has special treatment for SQL with id 'abcd123456789'. The result of `GET_REPORT` shows only actions for that SQL (actions to tune the SQL, or to investigate application using it) if the SQL is responsible for at least 2 average active sessions during the analysis period, and the average response time was at least 1 second.

```
var tname VARCHAR2(60);
BEGIN
  DBMS_ADDM.INSERT_SQL_DIRECTIVE(
      NULL,
      'my SQL directive',
      'abcd123456789',
      2,
      1000000);
  :tname := 'my_instance_analysis_mode_task';
  DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
```

To see a report containing all actions regardless of the directive:

```
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```

REAL_TIME_ADDM_REPORT Function

This function produces a real-time ADDM report for ADDM-related activity for the last five minutes. In an Oracle Real Application Clusters (Oracle RAC) environment, the function assumes that executing SQL over `GV$` is possible.

Syntax

```
DBMS_ADDM.REAL_TIME_ADDM_REPORT (
   cdb_type_override         IN     VARCHAR2,
   read_only_type_override   IN     VARCHAR2)
 RETURN CLOB;
```

Parameters

Table 16-21 REAL_TIME_ADDM_REPORT Function Parameters

Parameter	Description
`cdb_type_override`	Overrides the type of `CDB` that `ADDM` determines for doing analysis. The possible values are: • `AUTONOMOUS OLTP`—autonomous OLTP inside a PDB • `AUTONOMOUS DATA WAREHOUSE`—autonomous data warehouse (ADWH) inside a PDB • PDB —a regular PDB • CDB ROOT —the root of a CDB • NON-CDB —a system that is not CDB or PDB • AUTO —allows `ADDM` to decide the type of `CDB` to override based on the data
`read_only_type_override`	Overrides the type of `CDB` `ADDM` determines for analysis. The possible values are: • `READ-WRITE`—a regular database or the primary database in a data guard configuration • `READ-ONLY`—a database open in read-only mode, such as an active data guard standby • `AUTO`—allows `ADDM` to decide the type of `CDB` to override based on the data

Return Values

`CLOB` containing a real-time ADDM report

REEXECUTE_FAILED_AUTO_TASKS Procedure

This procedure re-executes the tasks registered in the `DBA_ADDM_PENDING_AUTOTASKS` views.

This API can be called from both the CDB and PDB level. The user specifies the range for the snapshots or a time interval. The duration of the re-execution is time constrained by a user-specified parameter or a default value. Once a task has executed successfully, its corresponding row is removed from the `DBA_ADDM_PENDING_AUTOTASKS` view.

Syntax

```
DBMS_ADDM.REXECUCUTE_FAILED_AUTO_TASKS(
   instance_number     IN   NUMBER :=NULL,
   begin_snapshot      IN   NUMBER :=NULL,
   end_snapshot        IN   NUMBER :=NULL,
   dbid                IN   NUMBER :=NULL,
   time_budget_in_sec  IN   NUMBER :=NULL,
   max_attempts        IN   NUMBER :=NULL);
```

Parameters

Table 16-22 REEXECUTE_FAILED_AUTO_TASKS Function Parameters

Parameter	Description
`instance_number`	Instance number for the tasks to be re executed.
`begin_snapshot`	Earliest begin snapshot ID for the tasks to be re executed.
`end_snapshot`	Latest end snapshot ID for the tasks to be re executed.
`dbid`	Database ID for the tasks to be re executed.
`time_budget_in_sec`	Forced time out to run the procedure after exiting.
`max_attempts`	Maximum number of attempts to re execute a task.