191
DBMS_SQLDIAG

The DBMS_SQLDIAG package provides an interface to the SQL Diagnosability functionality.

This chapter contains the following topics:

- DBMS_SQLDIAG Overview
- DBMS_SQLDIAG Security Model
- DBMS_SQLDIAG Constants
- Summary of DBMS_SQLDIAG Subprograms

See Also:

Oracle Database Administrator's Guide for more information about "Managing Diagnostic Data"

DBMS_SQLDIAG Overview

In the rare case that a SQL statement fails with a critical error, you can run the SQL Repair Advisor to try to repair the failed statement by using the `DBMS_SQLDIAG` package subprograms.

The SQL Repair Advisor analyzes the statement and in many cases recommends a patch to repair the statement. If you implement the recommendation, the applied SQL patch circumvents the failure by causing the query optimizer to choose an alternate execution plan for future executions.

See Also:

Oracle Database Administrator's Guide for more information about how to run the SQL Repair Advisor using the `DBMS_SQLDIAG` package subprograms.

DBMS_SQLDIAG Security Model

You must have the `ADVISOR` role to execute the `DBMS_SQLDIAG` package.

DBMS_SQLDIAG Constants

`DBMS_SQLDIAG` defines constants to use when specifying parameter values.

These constants are shown in the following tables:

- Table 191-1 describes the name of SQL repair advisor as seen by the advisor framework

- Table 191-2 describes SQLDIAG advisor task scope parameter values
- Table 191-3 describes SQLDIAG advisor `time_limit` constants
- Table 191-4 describes possible formats for a report
- Table 191-5 describes possible levels of detail in the report
- Table 191-6 describes possible report sections (comma delimited)
- Table 191-7 describes possible values for the `problem_type` parameter of the CREATE_DIAGNOSIS_TASK Functions
- Table 191-8 describes possible values for the `_sql_findings_mode` parameter

Table 191-1 DBMS_SQLDIAG Constants - SQLDIAG Advisor Name

Constant	Type	Value	Description
`ADV_SQL_DIAG_NAME`	`VARCHAR2(18)`	SQL Repair Advisor	Name of SQL repair advisor as seen by the advisor framework

Table 191-2 DBMS_SQLDIAG Constants - SQLDIAG Advisor Task Scope Parameter Values

Constant	Type	Value	Description
`SCOPE_COMPREHENSIVE`	`VARCHAR2(13)`	`COMPREHENSIVE`	Detailed analysis of the problem which may take more time to execute
`SCOPE_LIMITED`	`VARCHAR2(7)`	`LIMITED`	Brief analysis of the problem

Table 191-3 DBMS_SQLDIAG Constants - SQLDIAG Advisor time_limit Constants

Constant	Type	Value	Description
`TIME_LIMIT_DEFAULT`	`NUMBER`	`1800`	Default time limit for analysis of the problem

Table 191-4 DBMS_SQLDIAG Constants - Report Type (possible values) Constants

Constant	Type	Value	Description
`TYPE_HTML`	`VARCHAR2(4)`	`HTML`	Report from the REPORT_DIAGNOSIS_TASK Function in HTML form
`TYPE_TEXT`	`VARCHAR2(4)`	`TEXT`	Report from the REPORT_DIAGNOSIS_TASK Function in text form
`TYPE_XML`	`VARCHAR2(3)`	`XML`	Report from the REPORT_DIAGNOSIS_TASK Function in XML form

Table 191-5 DBMS_SQLDIAG Constants - Report Level (possible values) Constants

Constant	Type	Value	Description
`LEVEL_ALL`	`VARCHAR2(3)`	`ALL`	Complete report including annotations about statements skipped over
`LEVEL_BASIC`	`VARCHAR2(5)`	`BASIC`	Shows information about every statement analyzed, including recommendations not implemented
`LEVEL_TYPICAL`	`VARCHAR2(7)`	`TYPICAL`	Simple report shows only information about the actions taken by the advisor.

Table 191-6 DBMS_SQLDIAG Constants - Report Section (possible values) Constants

Constant	Type	Value	Description
`SECTION_ALL`	`VARCHAR2(3)`	`ALL`	All statements
`SECTION_ERRORS`	`VARCHAR2(6)`	`ERRORS`	Statements with errors
`SECTION_FINDINGS`	`VARCHAR2(8)`	`FINDINGS`	Tuning findings
`SECTION_INFORMATION`	`VARCHAR2(11)`	`INFORMATION`	General information
`SECTION_PLANS`	`VARCHAR2(5)`	`PLANS`	Explain plans
`SECTION_SUMMARY`	`VARCHAR2(7)`	`SUMMARY`	Summary information

Table 191-7 DBMS_SQLDIAG Constants - Problem Type Constants

Constant	Type	Value	Description
`PROBLEM_TYPE_PERFORMANCE`	`NUMBER`	`1`	User suspects this is a performance problem
`PROBLEM_TYPE_WRONG_RESULTS`	`NUMBER`	`2`	User suspects the query is giving inconsistent results
`PROBLEM_TYPE_COMPILATION_ERROR`	`NUMBER`	`3`	User sees a crash in compilation
`PROBLEM_TYPE_EXECUTION_ERROR`	`NUMBER`	`4`	User sees a crash in execution
`PROBLEM_TYPE_ALT_PLAN_GEN`	`NUMBER`	`5`	User to explore all alternative plans

Table 191-8 DBMS_SQLDIAG Constants - Findings Filter Constants

Constant	Type	Value	Description
`SQLDIAG_FINDINGS_ALL`	`NUMBER`	`1`	Show all possible findings

Table 191-8 (Cont.) DBMS_SQLDIAG Constants - Findings Filter Constants

Constant	Type	Value	Description
SQLDIAG_FINDINGS_VALIDATION	NUMBER	2	Show status of validation rules over structures
SQLDIAG_FINDINGS_FEATURES	NUMBER	3	Show only features used by the query
SQLDIAG_FINDINGS_FILTER_PLANS	NUMBER	4	Show the alternative plans generated by the advisor
SQLDIAG_FINDINGS_CR_DIFF	NUMBER	5	Show difference between two plans
SQLDIAG_FINDINGS_MASK_VARIANT	NUMBER	6	Mask info for testing
SQLDIAG_FINDINGS_OBJ_FEATURES	NUMBER	7	Show features usage history
SQLDIAG_FINDINGS_BASIC_INFO	NUMBER	8	Show the alternative plans generated by the advisor

Summary of DBMS_SQLDIAG Subprograms

This table lists the DBMS_SQLDIAG subprograms and briefly describes them.

Table 191-9 DBMS_SQLDIAG Package Subprograms

Subprogram	Description
ACCEPT_SQL_PATCH Function & Procedure	Accepts a recommended SQL patch as recommended by the specified SQL diagnosis task
ALTER_SQL_PATCH Procedure	Alters specific attributes of an existing SQL patch object
CANCEL_DIAGNOSIS_TASK Procedure	Cancels a diagnostic task
CREATE_DIAGNOSIS_TASK Functions	Creates a diagnostic task in order to diagnose a single SQL statement
CREATE_SQL_PATCH Function	Creates an SQL patch based on a set of user specified hints for specific statements identified by SQL text.
CREATE_STGTAB_SQLPATCH Procedure	Creates the staging table used for transporting SQL patches from one system to another
DROP_DIAGNOSIS_TASK Procedure	Drops a diagnostic task
DROP_SQL_PATCH Procedure	Drops the named SQL patch from the database
EXECUTE_DIAGNOSIS_TASK Procedure	Executes a diagnostic task
EXPLAIN_SQL_TESTCASE Function	Explains a SQL test case
EXPORT_SQL_TESTCASE Procedures	Exports a SQL test case to a directory

Table 191-9 (Cont.) DBMS_SQLDIAG Package Subprograms

Subprogram	Description
EXPORT_SQL_TESTCASE_DIR_BY_INC Function	Generates a SQL Test Case corresponding to the incident ID passed as an argument.
EXPORT_SQL_TESTCASE_DIR_BY_TXT Function	Generates a SQL Test Case corresponding to the SQL passed as an argument
GET_FIX_CONTROL Function	Returns the value of fix control for a given bug number
GET_SQL Function	Imports a SQL test case
IMPORT_SQL_TESTCASE Procedures	Imports a SQL test case into a schema
INCIDENTID_2_SQL Procedure	Initializes a `sql_setrow` from an incident ID
INTERRUPT_DIAGNOSIS_TASK Procedure	Interrupts a diagnostic task
LOAD_SQLSET_FROM_TCB Function	Loads a `SQLSET` from Test Case Builder (TCB) file
PACK_STGTAB_SQLPATCH Procedure	SQL patches into the staging table created by the CREATE_STGTAB_SQLPATCH Procedure
REPLAY_SQL_TESTCASE Function	Reports on a diagnostic task
REPORT_SQL Function	Generates a diagnostic report in HTML format for a specific SQL statement.
REPORT_DIAGNOSIS_TASK Function	Reports on a diagnostic task
RESET_DIAGNOSIS_TASK Procedure	Resets a diagnostic task
RESUME_DIAGNOSIS_TASK Procedure	Resumes a diagnostic task
SET_DIAGNOSIS_TASK_PARAMETER Procedure	Sets a diagnosis task parameter
SQL_DIAGNOSE_AND_REPAIR Function	Diagnoses a given SQL statement for a given SQL ID for the given problem type.
UNPACK_STGTAB_SQLPATCH Procedure	Unpacks from the staging table populated by a call to the PACK_STGTAB_SQLPATCH Procedure, using the patch data stored in the staging table to create patches on this system

ACCEPT_SQL_PATCH Function & Procedure

This procedure accepts a recommended SQL patch as recommended by the specified SQL diagnosis task.

Syntax

```
DBMS_SQLDIAG.ACCEPT_SQL_PATCH (
   task_name       IN  VARCHAR2,
   object_id       IN  NUMBER := NULL,
   name            IN  VARCHAR2 := NULL,
   description     IN  VARCHAR2 := NULL,
   category        IN  VARCHAR2 := NULL,
   task_owner      IN  VARCHAR2 := NULL,
   replace         IN  BOOLEAN := FALSE,
   force_match    IN  BOOLEAN := FALSE)
 RETURN VARCHAR2;

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

Parameters

Table 191-10 ACCEPT_SQL_PATCH Function & Procedure Parameters

Parameter	Description
`taskname`	Name of the SQL diagnosis task
`object_id`	Identifier of the advisor framework object representing the SQL statement associated to the diagnosis task
`name`	Name of the patch. It cannot contain double quotation marks. The name is case sensitive. If not specified, the system will generate a unique name for the SQL patch.
`description`	User specified string describing the purpose of this SQL patch. Maximum size of description is `500`.
`category`	Category name which must match the value of the `SQLDIAGNOSE_CATEGORY` parameter in a session for the session to use this patch. It defaults to the value `DEFAULT`. This is also the default of the `SQLDIAGNOSE_CATEGORY` parameter. The category must be a valid Oracle identifier. The category name specified is always converted to upper case. The combination of the normalized SQL text and category name create a unique key for a patch. An accept will fail if this combination is duplicated.
`task_owner`	Owner of the diagnosis task. This is an optional parameter that has to be specified to accept a SQL Patch associated to a diagnosis task owned by another user. The current user is the default value.
`replace`	If the patch already exists, it will be replaced if this argument is `TRUE`. It is an error to pass a name that is already being used for another signature/category pair, even with replace set to `TRUE`.
`force_match`	If `TRUE` this causes SQL Patches to target all SQL statements which have the same text after normalizing all literal values into bind variables. (Note that if a combination of literal values and bind values is used in a SQL statement, no bind transformation occurs.) This is analogous to the matching algorithm used by the `FORCE` option of the `CURSOR_SHARING` parameter. If `FALSE`, literals are not transformed. This is analogous to the matching algorithm used by the `EXACT` option of the `CURSOR_SHARING` parameter.

Return Values

Name of the SQL patch

Usage Notes

Requires `CREATE ANY SQL PROFILE` privilege

ALTER_SQL_PATCH Procedure

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

Syntax

```
DBMS_SQLDIAG.ALTER_SQL_PATCH (
   name            IN  VARCHAR2,
   attribute_name  IN  VARCHAR2,
   attribute_value IN  VARCHAR2);
```

Parameters

Table 191-11 ALTER_SQL_PATCH Procedure Parameters

Parameter	Description
`name`	Name of SQL patch to alter.
`attribute_name`	Name of SQL patch to alter. Possible values: • `STATUS` -> can be set to `ENABLED` or `DISABLED` • `NAME` -> can be reset to a valid name (must be a valid Oracle identifier and must be unique). • `DESCRIPTION` -> can be set to any string of size no more than 500 • `CATEGORY` -> can be reset to a valid category name (must be valid Oracle identifier and must be unique when combined with normalized SQL text) This parameter is mandatory and is case sensitive.
`attribute_value`	New value of the attribute. See `attribute_name` for valid attribute values. This parameter is mandatory.

Usage Notes

Requires `ALTER ANY SQL PATCH` privilege

CANCEL_DIAGNOSIS_TASK Procedure

This procedure cancels a diagnostic task.

Syntax

```
DBMS_SQLDIAG.CANCEL_DIAGNOSIS_TASK (
    taskname        IN   VARCHAR2);
```

Parameters

Table 191-12 CANCEL_DIAGNOSIS_TASK Procedure Parameters

Parameter	Description
`taskname`	Name of task

CREATE_DIAGNOSIS_TASK Functions

This function creates a diagnostic task in order to diagnose a single SQL statement. It returns a SQL diagnosis task unique name

Syntax

Prepares the diagnosis of a single statement given its text:

```
DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK (
    sql_text           IN   CLOB,
    bind_list          IN   sql_binds := NULL,
    user_name          IN   VARCHAR2  := NULL,
    scope              IN   VARCHAR2  := SCOPE_COMPREHENSIVE,
    time_limit         IN   NUMBER    := TIME_LIMIT_DEFAULT,
    task_name          IN   VARCHAR2  := NULL,
    description        IN   VARCHAR2  := NULL,
    problem_type       IN   NUMBER    := PROBLEM_TYPE_PERFORMANCE)
  RETURN VARCHAR2;
```

Prepares the diagnosis of a single statement from the Cursor Cache given its identifier:

```
DBMS_SQLDIAG.CREATE_DIAGNOSIS_TASK (
    sql_id             IN   VARCHAR2,
    plan_hash_value    IN   NUMBER    := NULL,
    scope              IN   VARCHAR2  := SCOPE_COMPREHENSIVE,
    time_limit         IN   NUMBER    := TIME_LIMIT_DEFAULT,
    task_name          IN   VARCHAR2  := NULL,
    description        IN   VARCHAR2  := NULL,
    problem_type       IN   NUMBER    := PROBLEM_TYPE_PERFORMANCE)
  RETURN VARCHAR2;
```

Prepares the diagnosis of a Sqlset:

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

Parameters

Table 191-13 CREATE_DIAGNOSIS_TASK Function Parameters

Parameter	Description
`sql_text`	Text of a SQL statement
`bind_list`	Set of bind values

Table 191-13 (Cont.) CREATE_DIAGNOSIS_TASK Function Parameters

Parameter	Description
`user_name`	Username for who the statement/sqlset will be diagnosed
`scope`	Diagnosis scope (limited/comprehensive)
`time_limit`	Maximum duration in seconds for the diagnosis session
`task_name`	Optional diagnosis task name
description	Maximum of 256 SQL diagnosis session description
`problem_type`	Determines the goal of the task. Possible values are: • `PROBLEM_TYPE_WRONG_RESULTS` • `PROBLEM_TYPE_COMPILATION_ERROR` • `PROBLEM_TYPE_EXECUTION_ERROR`
`sql_id`	Identifier of the statement
`plan_hash_value`	Hash value of the SQL execution plan
`sqlset_name`	Sqlset name
`basic_filter`	SQL predicate to filter the SQL from the SQL tuning set (STS)
`object_filter`	Object filter
`rank(i)`	Order-by clause on the selected SQL
`result_percentage`	Percentage on the sum of a ranking measure
`result_limit`	Top L(imit) SQL from (filtered/ranked) SQL
`plan_filter`	Plan filter. It is applicable in case there are multiple plans (`plan_hash_value`). This filter allows selecting one plan (plan_hash_value) only. Possible values are: • `LAST_GENERATED`: plan with most recent timestamp • `FIRST_GENERATED`: opposite to `LAST_GENERATED` • `LAST_LOADED`: plan with most recent `first_load_time` stat info • `FIRST_LOADED`: opposite to `LAST_LOADED` • `MAX_ELAPSED_TIME`: plan with maximum elapsed time • `MAX_BUFFER_GETS`: plan with maximum buffer gets • `MAX_DISK_READS`: plan with maximum disk reads • `MAX_DIRECT_WRITES`: plan with maximum direct writes • `MAX_OPTIMIZER_COST`: plan with maximum optimum cost
`sqlset_owner`	Owner of the sqlset, or null for current schema owner

CREATE_SQL_PATCH Function

This function creates a SQL patch based on a set of user specified hints for specific statements identified by SQL text.

A SQL patch is usually created automatically by the SQL Repair Advisor to prevent any errors during the compilation or execution of a SQL statement. This function provides a way to manually create a SQL patch based on a set of hints that resolves the error.

Syntax

```
DBMS_SQLDIAG.CREATE_SQL_PATCH (
    sql_text         IN    CLOB,
    hint_text        IN    CLOB,
    name             IN   VARCHAR2   := NULL,
    description      IN   VARCHAR2   := NULL,
    category         IN   VARCHAR2   := NULL,
    validate         IN   BOOLEAN    := TRUE)
RETURN VARCHAR2;

DBMS_SQLDIAG.CREATE_SQL_PATCH (
    sql_id           IN   VARCHAR2,
    hint_text        IN   CLOB,
    name             IN   VARCHAR2   := NULL,
    description      IN   VARCHAR2   := NULL,
    category         IN   VARCHAR2   := NULL,
    validate         IN   BOOLEAN    := TRUE)
RETURN VARCHAR2;
```

Parameters

Table 191-14 CREATE_SQL_PATCH Function Parameters

Parameter	Description
`sql_text`	Text of the SQL statement
`sql_id`	The SQL identifier for the SQL statement
`hint_text`	Hints to include in the SQL patch
`name`	Optional SQL patch name
`description`	Description of the SQL patch
`category`	Category name
`validate`	Whether to validate the provided hints

Return Values

Both functions return the SQL patch name.

CREATE_STGTAB_SQLPATCH Procedure

This procedure creates the staging table used for transporting SQL patches from one system to another.

Syntax

```
DBMS_SQLDIAG.CREATE_STGTAB_SQLPATCH (
   table_name        IN  VARCHAR2,
   schema_name       IN  VARCHAR2 := NULL,
   tablespace_name   IN  VARCHAR2 := NULL);
```

Parameters

Table 191-15 CREATE_STGTAB_SQLPATCH Procedure Parameters

Parameter	Description
`table_name`	(Mandatory) Name of the table to create (case-sensitive)
`schema_name`	Schema to create the table in, or `NULL` for current schema (case-sensitive)

Table 191-15 (Cont.) CREATE_STGTAB_SQLPATCH Procedure Parameters

Parameter	Description
`tablespace_name`	Tablespace to store the staging table within, or `NULL` for current user's default tablespace (case-sensitive)

DROP_DIAGNOSIS_TASK Procedure

This procedure drops a diagnostic task.

Syntax

```
DBMS_SQLDIAG.DROP_DIAGNOSIS_TASK (
    taskname         IN    VARCHAR2);
```

Parameters

Table 191-16 DROP_DIAGNOSIS_TASK Procedure Parameters

Parameter	Description
`taskname`	Name of task

DROP_SQL_PATCH Procedure

This procedure drops the named SQL patch from the database.

Syntax

```
DBMS_SQLDIAG.DROP_SQL_PATCH (
   name      IN   VARCHAR2,    ignore    IN   BOOLEAN := FALSE);
```

Parameters

Table 191-17 DROP_SQL_PATCH Function & Procedure Parameters

Parameter	Description
`name`	Name of patch to be dropped. The name is case sensitive.
`ignore`	Ignore errors due to object not existing.

Usage Notes

Requires `DROP ANY SQL PATCH` privilege

EXECUTE_DIAGNOSIS_TASK Procedure

This procedure executes a diagnostic task.

Syntax

```
DBMS_SQLDIAG.EXECUTE_DIAGNOSIS_TASK (
    taskname         IN    VARCHAR2);
```

Parameters

Table 191-18   EXECUTE_DIAGNOSIS_TASK Procedure Parameters

Parameter	Description
`taskname`	Name of task

EXPLAIN_SQL_TESTCASE Function

This procedure explains a SQL test case.

Syntax

```
DBMS_SQLDIAG.EXPLAIN_SQL_TESTCASE (
    sqlTestCase        IN    CLOB)
  RETURN CLOB;
```

Parameters

Table 191-19   EXPLAIN_SQL_TESTCASE Function Parameters

Parameter	Description
`sqlTestCase`	XML document describing the SQL test case

EXPORT_SQL_TESTCASE Procedures

This procedure exports a SQL test case to a directory.

Syntax

This variant has to be provided with the SQL information.

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE (
    directory                IN              VARCHAR2,
    sql_text                 IN              CLOB,
    user_name                IN              VARCHAR2  :=  NULL,
    bind_list                IN              sql_binds :=  NULL,
    exportEnvironment        IN              BOOLEAN   :=  TRUE,
    exportMetadata           IN              BOOLEAN   :=  TRUE,
    exportData               IN              BOOLEAN   :=  FALSE,
    exportPkgbody            IN              BOOLEAN   :=  FALSE,
    samplingPercent          IN              NUMBER    :=  100,
    ctrlOptions              IN              VARCHAR2  :=  NULL,
    timeLimit                IN              NUMBER    :=  0,
    testcase_name            IN              VARCHAR2  :=  NULL,
    testcase                 IN OUT NOCOPY   CLOB,
    preserveSchemaMapping    IN              BOOLEAN   :=  FALSE,
    version                  IN              VARCHAR2  :=  'COMPATIBLE');
```

This variant extracts the SQL information from an incident file.

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE (
    directory                IN              VARCHAR2,
    incident_id              IN              VARCHAR2,
    exportEnvironment        IN              BOOLEAN   :=  TRUE,
    exportMetadata            IN              BOOLEAN   :=  TRUE,
    exportData                IN              BOOLEAN   :=  FALSE,
    exportPkgbody             IN              BOOLEAN   :=  FALSE,
    samplingPercent           IN              NUMBER    :=  100,
    ctrlOptions               IN              VARCHAR2  :=  NULL,
    timeLimit                 IN              NUMBER    :=
                                                 DBMS_SQLDIAG.TIME_LIMIT_DEFAULT,
    testcase_name             IN              VARCHAR2  :=  NULL,
    testcase                  IN OUT NOCOPY   CLOB,
    preserveSchemaMapping     IN              BOOLEAN   :=  FALSE)
    version                   IN              VARCHAR2  :=  'COMPATIBLE');
```

This variant allow the SQL Test case to be generated from a cursor present in the cursor cache. Use `V$SQL` to get the SQL identifier and the SQL hash value.

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE (
    directory                 IN              VARCHAR2,
    sql_id                    IN              VARCHAR2,
    plan_hash_value           IN              NUMBER    := NULL,
    exportEnvironment         IN              BOOLEAN   :=  TRUE,
    exportMetadata            IN              BOOLEAN   :=  TRUE,
    exportData                IN              BOOLEAN   :=  FALSE,
    exportPkgbody             IN              BOOLEAN   :=  FALSE,
    samplingPercent           IN              NUMBER    :=  100,
    ctrlOptions               IN              VARCHAR2  :=  NULL,
    timeLimit                 IN              NUMBER    :=
                                                 DBMS_SQLDIAG.TIME_LIMIT_DEFAULT,
    testcase_name             IN              VARCHAR2  :=  NULL,
    testcase                  IN OUT NOCOPY   CLOB,
    preserveSchemaMapping     IN              BOOLEAN   :=  FALSE)
    version                   IN              VARCHAR2  :=  'COMPATIBLE');
```

Parameters

Table 191-20 EXPORT_SQL_TESTCASE Procedure Parameters

Parameter	Description
`directory`	Directory to store the various generated files
`sql_text`	Text of the SQL statement to export
`incident_id`	Incident ID containing the offending SQL
`sql_id`	Identifier of the statement in the cursor cache, automatic workload repository, or the automatic SQL tuning set
`username`	Name of the user schema to use to parse the SQL, defaults to `SYS`
`bind_list`	List of bind values associated to the statement
`exportEnvironment`	`TRUE` if the compilation environment should be exported
`exportMetadata`	`TRUE` if the definition of the objects referenced in the SQL should be exported
`exportData`	`TRUE` if the data of the objects referenced in the SQL should be exported
`exportPkgbody`	`TRUE` if the body of the packages referenced in the SQL are exported
`samplingPercent`	If is `TRUE`, specify the sampling percentage to use to create the dump file

Table 191-20 (Cont.) EXPORT_SQL_TESTCASE Procedure Parameters

Parameter	Description
`ctrlOptions`	Opaque control parameters. For example, for SQL execution low trace, set `ctrlOptions` with the following string: `'<parameter name="diag_event">SQLEXEC_LOW</parameter>'`. • `name="capture"` - `BASIC` (default) or `WITH_RUNTIME_INFO`. This parameter defines the mode of TCB capture. `BASIC`: runs as Oracle release 11*g* TCB and captures all the information that is captured in that release as well as AWR reports, SQL monitor reports and parameter information. `WITH_RUNTIME_INFO`: TCB captures runtime information for the SQL, such as dynamic sampling data, list of binds, Dynamic Plan info, along with information captured under `BASIC` mode. Note this must be the same value as used in the IMPORT_SQL_TESTCASE Procedures. • `name="stat_history_since"`–Value is date. The object statistics history is exported using this parameter. Statistics history after date specified will be exported. • `name="compress"`–This option is used to compress the SQL Test Case Builder output files into a zip file by default. The possible values are: – `YES` – `NO` The default value is `YES`. • `name="diag_event"`–This option is used to specify the level of trace information to include in the SQL Test Case Builder output. The possible values are: – `ADS` – `COMPILER` – `SQLEXEC_LOW` – `SQLEXEC_MEDIUM` – `SQLEXEC_HIGH` – `SQLEXEC_HIGHEST` The default value is `ADS + COMPILER`. • `name="problem_type"`–This option is used to assign an issue type for a SQL Test Case Builder test case. For example, if a test case is related to performance regression issue, then you can assign the value of PERFORMANCE to the problem_type option. The possible values are : – `PERFORMANCE` – `WRONG_RESULTS` – `COMPILATION_ERROR` – `EXECUTION_ERROR` The default value is `PERFORMANCE`.
`timeLimit`	How much time should we spend exporting the SQL test case
`testcaseName`	An optional name for the SQL test case. This is used to prefix all the generated scripts
`testcase`	Resulting testcase
`preserveSchemaMapping`	`TRUE` if the schema (or schemas) are not re-mapped from the original environment to the test environment

Table 191-20 (Cont.) EXPORT_SQL_TESTCASE Procedure Parameters

Parameter	Description
`version`	Version of database objects to be extracted. This option is only valid for `EXPORT`. Database objects or attributes incompatible with the version will not be extracted. • `COMPATIBLE` - (default) the version of the metadata corresponds to the database compatibility level and the compatibility release level for feature (as given in the `V$COMPATIBILITY` view). Database compatibility must be set to 9.2 or higher. • `LATEST` - the version of the metadata that specifies the current database version. • A specific database version. For example, if `'10.0.0'`, this cannot be lower than Oracle Database release 10.0.0.

Usage Notes

- A SQL test case generates a set of files needed to help reproduce a SQL failure on a different machine. It contains:
  - a dump file containing schemas objects and statistics (`.dmp`)
  - the explain plan for the statements (in advanced mode)
  - diagnostic information gathered on the offending statement
  - an import script to execute to reload the objects
  - a SQL script to replay system statistics of the source
  - a table of contents file describing the SQL test case
  - metadata. (`xxxxmain.xml`)
  - a `README.txt` file that explain the usage of the TCB
  - the outlines used by the statement (`ol.xml`)
  - a list of parameters set in the exporting db/env (`prmimp.sql`)
  - a SQL monitor report, if any (`smrpt.html`)
  - an AWR report, if any (`awrrpt.html`)
  - a list of binds used in this statement (`bndlst.xml`)
- You should not run Test Case Builder (TCB) under user `SYS`. Instead, use another user who can be granted the `DBA` privilege.
- The default setting for TCB is that data is not exported. However, in some cases data is required, such as to diagnose an outcome with a result that is not optimal. To export data, call `EXPORT_SQL_TESTCASE` with `exportData=>TRUE` and the data will be imported by default, unless turned `OFF` by `importData=>FALSE`.
- TCB includes PL/SQL package spec by default, but not the PL/SQL package body. However, you may need to have the package body as well, for example, to invoke the PL/SQL functions, or because you have a Virtual Private Database (VPD) function defined in a package. To export a PL/SQL package body, call `EXPORT_SQL_TESTCASE` with `exportPkgbody=>TRUE`. To import a PL/SQL package body, call IMPORT_SQL_TESTCASE Procedures with i`mportPkgbody=>TRUE`.

- To export objects statistics history, the database compatibility should be set to 12.0 or higher.
- This procedure does not export data and statistics on a Global Temporary Table (GTT).

Examples

The user can specify multiple parameters in the `ctrlOptions` encapsulated either by using the `<parameters>` parent tag or without the parent tag.

Using the <parameters> tag

```
<parameters>
<parameter name="capture">with_runtime_info</parameter>
<parameter name="diag_event">SQLEXEC_LOW</parameter>
</parameters>
```

Without the <parameters> tag

```
<parameter name="capture">with_runtime_info</parameter>
<parameter name="compress">yes</parameter>
```

EXPORT_SQL_TESTCASE_DIR_BY_INC Function

This function generates a SQL test case corresponding to the incident ID passed as an argument. It creates a set of scripts and dump file in the directory passed as an argument.

Syntax

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE_DIR_BY_INC (
    incident_id        IN   NUMBER,
    directory          IN   VARCHAR2,
    exportEnvironment  IN   VARCHAR2 := 'TRUE',
    exportMetadata     IN   VARCHAR2 := 'TRUE',
    exportData         IN   VARCHAR2 := 'FALSE',
    samplingPercent    IN   VARCHAR2 := '100',
    ctrlOptions        IN   VARCHAR2 := NULL
    version            IN   VARCHAR2 := 'COMPATIBLE')
 RETURN BOOLEAN;
```

Parameters

Table 191-21 EXPORT_SQL_TESTCASE_DIR_BY_INC Function Parameters

Parameter	Description
`incident_id`	Incident ID containing the offending SQL. For more information about Incidents, see *Oracle Database Performance Tuning Guide*.
`directory`	Directory path to the generated files
`exportEnvironment`	`TRUE` if the compilation environment should be exported
`exportMetadata`	`TRUE` if the definition of the objects referenced in the SQL should be exported
`exportData`	`TRUE` if the data of the objects referenced in the SQL should be exported
`samplingPercent`	If is `TRUE`, specify the sampling percentage to use to create the dump file

Table 191-21 (Cont.) EXPORT_SQL_TESTCASE_DIR_BY_INC Function Parameters

Parameter	Description
`ctrlOptions`	Opaque control parameters. For example, to export statistics history from a given date, set `ctrlOptions` as follows: `<parameter name="stat_history_since">26-FEB-25 12.35.03.534278 PM<parameter>`. • `name="capture"` - `BASIC` (default) or `WITH_RUNTIME_INFO`. This parameter defines the mode of TCB capture. `BASIC`: runs as Oracle release 11*g* TCB and captures all the information that is captured in that release as well as AWR reports, SQL monitor reports and parameter information. `WITH_RUNTIME_INFO`: TCB captures runtime information for the SQL, such as dynamic sampling data, list of binds, Dynamic Plan info, along with information captured under `BASIC` mode. • `name="stat_history_since"` - Value is date. The object statistics history is exported using this parameter. Statistics history after date specified will be exported.
`version`	Version of database objects to be extracted. This option is only valid for `EXPORT`. Database objects or attributes incompatible with the version will not be extracted. • `COMPATIBLE` - (default) the version of the metadata corresponds to the database compatibility level and the compatibility release level for feature (as given in the `V$COMPATIBILITY` view). Database compatibility must be set to 9.2 or higher. • `LATEST` - the version of the metadata that specifies the current database version. • A specific database version. For example, if `'10.0.0'`, this cannot be lower than Oracle Database release 10.0.0.

EXPORT_SQL_TESTCASE_DIR_BY_TXT Function

This function generates a SQL Test Case corresponding to the SQL passed as an argument. It creates a set of scripts and dump files in the directory passed as an argument.

Syntax

```
DBMS_SQLDIAG.EXPORT_SQL_TESTCASE_DIR_BY_TXT (
    incident_id        IN   NUMBER,
    directory          IN   VARCHAR2,
    sql_text           IN   CLOB,
    user_name          IN   VARCHAR2 := 'SYS',
    exportEnvironment  IN   VARCHAR2 := 'TRUE',
    exportMetadata     IN   VARCHAR2 := 'TRUE',
    exportData         IN   VARCHAR2 := 'FALSE',
    samplingPercent    IN   VARCHAR2 := '100',
    ctrlOptions        IN   VARCHAR2 := NULL
    version            IN   VARCHAR2 := 'COMPATIBLE')
  RETURN BOOLEAN;
```

Parameters

Table 191-22 EXPORT_SQL_TESTCASE_DIR_BY_TXT Function Parameters

Parameter	Description
`incident_id`	Incident ID containing the offending SQL
`directory`	Directory to store the various generated files
`sql_text`	Text of the SQL statement to explain
`username`	Name of the user schema to use to parse the SQL, defaults to `SYS`
`exportEnvironment`	`TRUE` if the compilation environment should be exported
`exportMetadata`	`TRUE` if the definition of the objects referenced in the SQL should be exported
`exportData`	`TRUE` if the data of the objects referenced in the SQL should be exported
`samplingPercent`	If is `TRUE`, specify the sampling percentage to use to create the dump file
`ctrlOptions`	Opaque control parameters. For example, to export statistics history from a given date, set `ctrlOptions` as follows: `<parameter name="stat_history_since">26-FEB-25 12.35.03.534278 PM<parameter>`. • `capture` - `BASIC` (default) or `WITH_RUNTIME_INFO`. This parameter defines the mode of TCB capture. `BASIC`: runs as Oracle Release 11*g* TCB and captures all the information that is captured in that release as well as AWR reports, SQL monitor reports and parameter information. `WITH_RUNTIME_INFO`: TCB captures runtime information for the SQL, such as dynamic sampling data, list of binds, Dynamic Plan info, along with information captured under `BASIC` mode. • `name=stat_history_since` - Value is date. The object statistics history is exported using this parameter. Statistics history after date specified will be exported.
`version`	Version of database objects to be extracted. This option is only valid for `EXPORT`. Database objects or attributes incompatible with the version will not be extracted. • `COMPATIBLE` - (default) the version of the metadata corresponds to the database compatibility level and the compatibility release level for feature (as given in the `V$COMPATIBILITY` view). Database compatibility must be set to 9.2 or higher. • `LATEST` - the version of the metadata that specifies the current database version. • A specific database version. For example, if `'10.0.0'`, this cannot be lower than Oracle Database Release 10.0.0.

GET_FIX_CONTROL Function

This function returns the value of fix control for a given bug number.

Syntax

```
DBMS_SQLDIAG.GET_FIX_CONTROL (
    bug_number    IN    NUMBER)
  RETURN NUMBER;
```

Parameters

Table 191-23 GET_FIX_CONTROL Function Parameters

Parameter	Description
`bug_number`	Bug number

GET_SQL Function

This function loads a `sql_setrow` from the trace file associated to an the given incident ID.

Syntax

```
DBMS_SQLDIAG.GET_SQL (
    incident_id  IN     VARCHAR2)
  RETURN SQLSET_ROW;
```

Parameters

Table 191-24 GET_SQL Function Parameters

Parameter	Description
`incident_id`	Identifier of the incident

IMPORT_SQL_TESTCASE Procedures

This procedure imports a SQL test case into a schema.

Syntax

This variant requires a source directory and SQL Testcase metadata object (in XML format).

```
DBMS_SQLDIAG.IMPORT_SQL_TESTCASE (
    directory                IN    VARCHAR2,
    sqlTestCase              IN    CLOB,
    importEnvironment        IN    BOOLEAN    :=  TRUE,
    importMetadata           IN    BOOLEAN    :=  TRUE,
    importData               IN    BOOLEAN    :=  TRUE,
    importPkgbody            IN    BOOLEAN    :=  FALSE,
    importDiagnosis          IN    BOOLEAN    :=  TRUE,
    ignoreStorage            IN    BOOLEAN    :=  TRUE,
    ctrlOptions              IN    VARCHAR2   :=  NULL,
    preserveSchemaMapping    IN    BOOLEAN    :=  FALSE);
```

This variant requires a source directory name of SQL Testcase metadata file.

```
DBMS_SQLDIAG.IMPORT_SQL_TESTCASE (
    directory                IN   VARCHAR2,
    filename                 IN   VARCHAR2,
    importEnvironment        IN   BOOLEAN   :=  TRUE,
    importMetadata           IN   BOOLEAN   :=  TRUE,
    importData               IN   BOOLEAN   :=  TRUE,
    importPkgbody            IN   BOOLEAN   :=  FALSE,
    importDiagnosis          IN   BOOLEAN   :=  TRUE,
    ignoreStorage            IN   BOOLEAN   :=  TRUE,
    ctrlOptions              IN   VARCHAR2  :=  NULL,
    preserveSchemaMapping    IN   BOOLEAN   :=  FALSE);
```

Parameters

Table 191-25 IMPORT_SQL_TESTCASE Procedure Parameters

Parameter	Description
`directory`	Directory containing test case files
`filename`	Name of a file containing an XML document describing the SQL test case
`importEnvironment`	`TRUE` if the compilation environment should be imported
`importMetadata`	`TRUE` if the definition of the objects referenced in the SQL should be imported
`importData`	`TRUE` if the data of the objects referenced in the SQL should be imported
`importPkgbody`	`TRUE` if the body of the packages referenced in the SQL are imported
`importDiagnosis`	`TRUE` if the diagnostic information associated to the task should be imported
`ignoreStorage`	`TRUE` if the storage attributes should be ignored
`ctrlOptions`	Opaque control parameters, of which only `capture` is valid for this subprogam. • `capture` - `BASIC` (default) or `WITH_RUNTIME_INFO`. This parameter defines the mode of TCB capture. `BASIC`: runs as Oracle Release 11*g* TCB and captures all the information that is captured in that release as well as AWR reports, SQL monitor reports and parameter information. `WITH_RUNTIME_INFO`: TCB captures runtime information for the SQL, such as dynamic sampling data, list of binds, Dynamic Plan info, along with information captured under `BASIC` mode.
`preserveSchemaMapping`	`TRUE` if the schema (or schemas) are not re-mapped from the original environment to the test environment (schema mapping in the target database will be identical to the source database). Note that when an import is run with `preservesSchemaMapping` set to `TRUE`, if the objects in the schemas exists then the import will overwrite the existing objects.

Usage Notes

- A SQL test case generates a set of files needed to help reproduce a SQL failure on a different machine. It contains:
  - a dump file containing schemas objects and statistics (`.dmp`)

  - the explain plan for the statements (in advanced mode)
  - diagnostic information gathered on the offending statement
  - an import script to execute to reload the objects
  - a SQL script to replay system statistics of the source
  - a table of contents file describing the SQL test case
  - metadata. (`xxxxmain.xml`)
  - a `README.txt` file that explain the usage of the TCB
  - the outlines used by the statement (`ol.xml`)
  - a list of parameters set in the exporting db/env (`prmimp.sql`)
  - a SQL monitor report, if any (`smrpt.html`)
  - an AWR report, if any (`awrrpt.html`)
  - a list of binds used in this statement (`bndlst.xml`)
- You should not run Test Case Builder (TCB) under user `SYS`. Instead, use another user who can be granted the `DBA` privilege
- The default setting for TCB is that data is not exported. However, in some cases data is required, such as to diagnose an outcome with a result that is not optimal. To export data, call EXPORT_SQL_TESTCASE Procedures with `exportData=>TRUE` and the data will be imported by default, unless turned `OFF` by `importData=>FALSE`.
- TCB includes PL/SQL package spec by default, but not the PL/SQL package body. However, you may need to have the package body as well, for example, to invoke the PL/SQL functions, or because you have a Virtual Private Database (VPD) function defined in a package. To export a PL/SQL package body, call EXPORT_SQL_TESTCASE Procedures with `exportPkgbody=>TRUE`. To import a PL/SQL package body, call `IMPORT_SQL_TESTCASE` Procedures with `importPkgbody=>TRUE`.
- The `capture` value used when invoking the EXPORT_SQL_TESTCASE Procedures must be used when calling this procedure.

INCIDENTID_2_SQL Procedure

This procedure initializes a `sql_setrow` from an incident ID.

Syntax

```
DBMS_SQLDIAG.INCIDENTID_2_SQL (
    incident_id   IN     VARCHAR2,
    sql_stmt      OUT    SQLSET_ROW,
    problem_type  OUT    NUMBER,
    err_code      OUT    BINARY_INTEGER,
    err_mesg      OUT    VARCHAR2);
```

Parameters

Table 191-26 INCIDENTID_2_SQL Procedure Parameters

Parameter	Description
`incident_id`	Identifier of the incident

Table 191-26 (Cont.) INCIDENTID_2_SQL Procedure Parameters

Parameter	Description
`sql_stmt`	Resulting SQL
`problem_type`	Tentative type of SQL problem (currently among `PROBLEM_TYPE_COMPILATION_ERROR` and `PROBLEM_TYPE_EXECUTION_ERROR`)
`err_code`	Error code if any otherwise it is set to `NULL`
`err_msg`	Error message if any otherwise it is set to `NULL`

INTERRUPT_DIAGNOSIS_TASK Procedure

This procedure interrupts a diagnostic task.

Syntax

```
DBMS_SQLDIAG.INTERRUPT_DIAGNOSIS_TASK (
    taskname        IN   VARCHAR2);
```

Parameters

Table 191-27 INTERRUPT_DIAGNOSIS_TASK Procedure Parameters

Parameter	Description
`taskname`	Name of task

LOAD_SQLSET_FROM_TCB Function

This function loads a `SQLSET` from a Test Case Builder file.

Syntax

```
DBMS_SQLDIAG.LOAD_SQLSET_FROM_TCB (
    directory        IN     VARCHAR2,
    filename         IN     VARCHAR2,
    sqlset_name      IN     VARCHAR2 DEFAULT NULL)
  RETURN VARCHAR2;
```

Parameters

Table 191-28 LOAD_SQLSET_FROM_TCB Function Parameters

Parameter	Description
`directory`	Name of directory
`filename`	Name of file
`sqlset_name`	Name of `SQLSET`

PACK_STGTAB_SQLPATCH Procedure

This procedure packs SQL patches into the staging table created by a call to the CREATE_STGTAB_SQLPATCH Procedure.

Syntax

```
DBMS_SQLDIAG.PACK_STGTAB_SQLPATCH (
   patch_name             IN  VARCHAR2 := '%',
   patch_category         IN  VARCHAR2 := 'DEFAULT',
   staging_table_name     IN  VARCHAR2,
   staging_schema_owner   IN  VARCHAR2 := NULL);
```

Parameters

Table 191-29 PACK_STGTAB_SQLPATCH Procedure Parameters

Parameter	Description
`patch_name`	Name of patch to pack (% wildcards acceptable, case-sensitive)
`patch_category`	Category to which to pack patches (% wildcards acceptable, case-insensitive)
`staging_table_name`	(Mandatory) Name of the table to use (case-sensitive)
`staging_schema_owner`	Schema where the table resides, or `NULL` for current schema (case-sensitive)

Usage Notes

- Requires: `ADMINISTER SQL PLAN MANAGEMENT OBJECT` privilege and `INSERT` privilege on the staging table
- By default, we move all SQL patches in category `DEFAULT`. Note that the subprogram issues a `COMMIT` after packing each SQL patch, so if an error is raised in mid-execution, some patches may be in the staging table.

Related Topics

- CREATE_STGTAB_SQLPATCH Procedure
  This procedure creates the staging table used for transporting SQL patches from one system to another.

REPLAY_SQL_TESTCASE Function

This function automates the reproduction of the SQL Test Case.

Syntax

```
DBMS_SQLDIAG.REPLAY_SQL_TESTCASE (
   directory        IN   VARCHAR2,
   filename         IN   VARCHAR2,
   ctrlOptions      IN   VARCHAR2  := NULL,
   format           IN   VARCHAR2  := 'TEXT')
   RETURN CLOB;

DBMS_SQLDIAG.REPLAY_SQL_TESTCASE (
   directory        IN   VARCHAR2,
   sqlTestCase      IN   CLOB,
   ctrlOptions      IN   VARCHAR2  := NULL,
   format           IN   VARCHAR2  := 'TEXT')
   RETURN CLOB;
```

Parameters

Table 191-30 REPLAY_SQL_TESTCASE Function Parameters

Parameter	Description
`directory`	Directory containing test case files
`filename`	Name of a file containing an XML document describing the SQL test case
`ctrlOptions`	Opaque control parameters. For example, to replay using outlines, set `ctrlOptions` as follows: `<parameter name="replay">OUTLINE<parameter>`. • `replay` - `EXPLAIN` (default), `OUTLINE`, `EXECUTION` or `OUTLINE EXECUTION`. This parameter defines TCB replay functionality. `EXPLAIN`: Replay explains the statement without using outlines `OUTLINE`: Replay uses outlines mode and explains the statement using outlines `EXECUTION`: Replay executes the statement without using outlines `OUTLINE EXECUTION`: Replay executes the statement using outlines
`sqlTestCase`	SQL test case
`format`	Format of the replay report. Possible formats are: `TEXT`, `XML` and `HTML`..

Examples

```
TCB Replay Mode: Execute
 SELECT /* tcbdynpl_1 */ /*+ gather_plan_statistics */ * FROM (SELECT * FROM emp where
emp.sal > 100) emp, dept WHERE emp.deptno = dept.deptno And emp.sal > 1000 /* tcbdynpl_1
*/

Explain Plan
 Plan Hash Value  : 2219294842

-----------------------------------------------------------------
| Id  | Operation                       | Name            | Rows |
-----------------------------------------------------------------
|   0 | SELECT STATEMENT                |                 |   13 |
| * 1 |   HASH JOIN                     |                 |   13 |
|   2 |    NESTED LOOPS                 |                 |      |
|   3 |     NESTED LOOPS                |                 |   13 |
|   4 |      STATISTICS COLLECTOR       |                 |      |
|   5 |       TABLE ACCESS FULL         | DEPT            |    4 |
| * 6 |      INDEX RANGE SCAN           | EMP_IDX_DEPTNO  |      |
| * 7 |     TABLE ACCESS BY INDEX ROWID | EMP             |    3 |
| * 8 |    TABLE ACCESS FULL            | EMP             |   13 |
-----------------------------------------------------------------

Predicate Information (identified by operation id):
------------------------------------------
* 1 - access("EMP"."DEPTNO"="DEPT"."DEPTNO")
* 6 - access("EMP"."DEPTNO"="DEPT"."DEPTNO")
* 7 - filter("EMP"."SAL">1000)
* 8 - filter("EMP"."SAL">1000)
```

```
Runtime Plan
 Plan Hash Value  : 2219294842

-------------------------------------------------------
| Id  | Operation            | Name | E-Card | A-Card |
-------------------------------------------------------
|   0 | SELECT STATEMENT     |      |        |      0 |
| * 1 |   HASH JOIN          |      |     13 |      0 |
|   2 |    TABLE ACCESS FULL | DEPT |      4 |      0 |
| * 3 |    TABLE ACCESS FULL | EMP  |     13 |      0 |
-------------------------------------------------------

Predicate Information (identified by operation id):
------------------------------------------
* 1 - access("EMP"."DEPTNO"="DEPT"."DEPTNO")
* 3 - filter("EMP"."SAL">1000)

REPLAY Note:
-----------
- Replay used dynamic sampling
- Replay forced Dynamic plan
```

REPORT_SQL Function

Generates a diagnostic report in HTML format for a specific SQL statement.

Syntax

```
DBMS_SQLDIAG.REPORT_SQL (
    sql_id       IN   VARCHAR2,
    directory    IN   VARCHAR2,
    level        IN   VARCHAR2
)
RETURN CLOB;
```

Parameters

Table 191-31 DBMS_SQLDIAG. REPORT_SQL Parameters

Parameter	Description
`sql_id`	ID of the SQL statement.
`directory`	Directory object where the report is written. By default, this parameter is NULL and the report is returned as a CLOB and not written to disk.
`level`	Three options: • `BASIC` – A basic report covering minimal report detail. • `TYPICAL` – A standard (default) report which contains the basic information plus other, more advanced, report sections. • `ALL` – A complete report containing all possible report information.

Usage Notes

By default, the `level` parameter is NULL and the report is returned as a CLOB and not written to disk.

If a directory name is specified for the `DIRECTORY` argument, the file name will be created in the following format: `SQLR_<SQL_ID>_<YYYYMMDDHH24MI>.html`, where `<SQL_ID>` represents the

SQL identifier provided as the `SQL_ID` argument and `<YYYYMMDDHH24MI>` represents the timestamp at which the file was created.

Example 191-1

In this example, `SQL_ID 'gtckcpxmp3ry7'` is passed in, the directory is the standard Data Pump directory, and `ALL` detail is returned in the report.

```
declare my_report clob;

begin
   my_report := dbms_sqldiag.report_sql('gtckcpxmp3ry7',
directory=>'DATA_PUMP_DIR', level=>'ALL');
end;
/
```

REPORT_DIAGNOSIS_TASK Function

This function reports on a diagnostic task. It returns a `CLOB` containing the desired report.

Syntax

```
DBMS_SQLDIAG.REPORT_DIAGNOSIS_TASK (
    taskname           IN   VARCHAR2,
    type               IN   VARCHAR2  := TYPE_TEXT,
    level              IN   VARCHAR2  := LEVEL_TYPICAL,
    section            IN   VARCHAR2  := SECTION_ALL,
    object_id          IN   NUMBER    := NULL,
    result_limit       IN   NUMBER    := NULL,
    owner_name         IN   VARCHAR2  := NULL)
  RETURN CLOB;
```

Parameters

Table 191-32 REPORT_DIAGNOSIS_TASK Function Parameters

Parameter	Description
`taskname`	Name of task to report
`type`	Type of the report. Possible values are: TEXT, HTML, XML (see Table 191-4).
`level`	Format of the recommendations. Possible values are TYPICAL, BASIC, ALL (Table 191-5).
`section`	Particular section in the report. Possible values are: SUMMARY, FINDINGS, PLAN, INFORMATION, ERROR, ALL (Table 191-6).
`object_id`	Identifier of the advisor framework object that represents a given statement in a SQL Tuning Set (STS).
`result_limit`	Number of statements in a STS for which the report is generated
`owner_name`	Name of the task execution to use. If `NULL`, the report will be generated for the last task execution.

RESET_DIAGNOSIS_TASK Procedure

This procedure resets a diagnostic task.

Syntax

```
DBMS_SQLDIAG.RESET_DIAGNOSIS_TASK (
    taskname         IN   VARCHAR2);
```

Parameters

Table 191-33   RESET_DIAGNOSIS_TASK Procedure Parameters

Parameter	Description
`taskname`	Name of task

RESUME_DIAGNOSIS_TASK Procedure

This procedure resumes a diagnostic path.

Syntax

```
DBMS_SQLDIAG.RESUME_DIAGNOSIS_TASK (
    taskname         IN   VARCHAR2);
```

Parameters

Table 191-34   RESUME_DIAGNOSIS_TASK Procedure Parameters

Parameter	Description
`taskname`	Name of task

SET_DIAGNOSIS_TASK_PARAMETER Procedure

This procedure is called to update the value of a SQL diagnosis parameter of type `VARCHAR2`.

The task must be set to its initial state before calling this procedure. The diagnosis parameters that can be set by this procedure are:

- `MODE`: diag scope (comprehensive, limited)
- `_SQLDIAG_FINDING_MODE`: findings in the report (see "Table 191-8" for possible values)

Syntax

```
DBMS_SQLDIAG.SET_DIAGNOSIS_TASK_PARAMETER (
    taskname            IN   VARCHAR2,
    parameter           IN   VARCHAR2,    value              IN   NUMBER);
```

Parameters

Table 191-35 SET_DIAGNOSIS_TASK_PARAMETER Procedure Parameters

Parameter	Description
`taskname`	Identifier of the task to execute
`parameter`	Name of the parameter to set
`value`	New value of the specified parameter

SQL_DIAGNOSE_AND_REPAIR Function

Diagnoses a given SQL statement for a given SQL ID for the given problem type. This function creates an incident, populate incident metadata with required information like, SQL ID, SQL text, compilation environment, and so on. It also creates a diagnostic task, executes it and accepts SQL PATCH recommendation for a given SQL ID.

Syntax

```
DBMS_SQLDIAG.SQL_DIAGNOSE_AND_REPAIR (
    sql_text           IN   CLOB,
    bind_list          IN   sql_binds := NULL,
    scope              IN   VARCHAR2  := SCOPE_COMPREHENSIVE,
    time_limit         IN   NUMBER    := TIME_LIMIT_DEFAULT,
    problem_type       IN   NUMBER    := PROBLEM_TYPE_PERFORMANCE,
    auto_apply_patch   IN   VARCHAR2  := YES)
  RETURN NUMBER;


DBMS_SQLDIAG.SQL_DIAGNOSE_AND_REPAIR (
    sql_id             IN   VARCHAR2,
    plan_hash_value    IN   NUMBER   := NULL,
    scope              IN   VARCHAR2 := SCOPE_COMPREHENSIVE,
    time_limit         IN   NUMBER   := TIME_LIMIT_DEFAULT,
    problem_type       IN   NUMBER   := PROBLEM_TYPE_PERFORMANCE,
    auto_apply_patch   IN   VARCHAR2 := YES)
  RETURN NUMBER;

DBMS_SQLDIAG.SQL_DIAGNOSE_AND_REPAIR (
    incident_id        IN   VARCHAR2,
    scope              IN   VARCHAR2 := SCOPE_COMPREHENSIVE,
    time_limit         IN   NUMBER   := TIME_LIMIT_DEFAULT,
    problem_type       IN   NUMBER   := PROBLEM_TYPE_PERFORMANCE,
    auto_apply_patch   IN   VARCHAR2 := YES)
  RETURN NUMBER;
```

Parameters

Table 191-36 SQL_DIAGNOSE_AND_REPAIR Function Parameters

Parameter	Description
`sql_text`	Text of the SQL statement.
`sql_id`	SQL ID of the SQL query.

Table 191-36 (Cont.) SQL_DIAGNOSE_AND_REPAIR Function Parameters

Parameter	Description
`plan_hash_value`	The plan to be used for diagnosis. The default value is `NULL`.
`bind_list`	Binds to be used for diagnosis. The default value is `NULL`.
`scope`	The scope of diagnostic advisor. Possible values are: • `SCOPE_LIMITED`—only index and plan analyze are invoked for a given SQL. • `SCOPE_COMPREHENSIVE`—besides index and plan analyze, auto-tune is called first to tune the statement. The default value is `SCOPE_COMPREHENSIVE`.
`time_limit`	Time limit for diagnostic task. The default value is `TIME_LIMIT_DEFAULT`.
`problem_type`	Problem type that is being diagnosed. The following problem type are supported: • `PROBLEM_TYPE_PERFORMANCE`—performance problem. • `PROBLEM_TYPE_WRONG_RESULTS`— incorrect results. • `PROBLEM_TYPE_COMPILATION_ERROR`—crash during compilation of the statement. • `PROBLEM_TYPE_EXECUTION_ERROR`— crash during execution of the statement. The default value is `PROBLEM_TYPE_PERFORMANCE`.
`auto_apply_patch`	A value that decides if the recommended SQL patch needs to be accepted. Possible values are: • `YES`—accepts the recommended SQL patch. • `NO`—does not accepts recommended SQL patch automatically. User need to manually accept the SQL patch. The default value is `YES`.

UNPACK_STGTAB_SQLPATCH Procedure

This procedure unpacks from the staging table populated by a call to the PACK_STGTAB_SQLPATCH Procedure. It uses the patch data stored in the staging table to create patches on this system. Users can opt to replace existing patches with patch data when they exist already. In this case, note that it is only possible to replace patches referring to the same statement if the names are the same (see the ACCEPT_SQL_PATCH Function & Procedure).

Syntax

```
DBMS_SQLDIAG.UPPACK_STGTAB_SQLPATCH (
   patch_name             IN  VARCHAR2 := '%',
   patch_category         IN  VARCHAR2 := '%',
   replace                IN  BOOLEAN,
   staging_table_name    IN  VARCHAR2,
   staging_schema_owner  IN  VARCHAR2 := NULL);
```

Parameters

Table 191-37 UPPACK_STGTAB_SQLPATCH Procedure Parameters

Parameter	Description
`patch_name`	Name of patch to unpack (% wildcards acceptable, case-sensitive)
`patch_category`	Category from which to unpack patches (% wildcards acceptable, case-insensitive)
`replace`	Replace patches if they already exist. Note that patches cannot be replaced if there is one in the staging table with the same name as an active patch on different SQL. The subprogram raises an error if there an attempt to create a patch that already exists.
`staging_table_name`	(Mandatory) Name of the table to use (case-sensitive)
`staging_schema_owner`	Schema where the table resides, or `NULL` for current schema (case-sensitive)

Usage Notes

- Requires: `ADMINISTER SQL MANAGEMENT OBJECT` privilege and `SELECT` or `READ` privilege on the staging table
- By default, all SQL patches in the staging table are moved. The function commits after successfully loading each patch. If it fails in creating an individual patch, it raises an error and does not proceed to those remaining in the staging table.

Related Topics

- PACK_STGTAB_SQLPATCH Procedure
  This procedure packs SQL patches into the staging table created by a call to the CREATE_STGTAB_SQLPATCH Procedure.
- ACCEPT_SQL_PATCH Function & Procedure
  This procedure accepts a recommended SQL patch as recommended by the specified SQL diagnosis task.