194 DBMS_SQLSET

The `DBMS_SQLSET` package provides an interface to manage SQL tuning sets.

This package provides the same subprograms, although in some cases with slightly different names, as the SQL tuning set subprograms in `DBMS_SQLTUNE`. The difference is that `DBMS_SQLSET` does not require the Oracle Tuning Pack.

This chapter contains the following topics:

DBMS_SQLSET Overview

Use this package to manage SQL tuning sets.

> **Note:**
>
> All `DBMS_SQLSET` subprograms have equivalents in the `DBMS_SQLTUNE` package.

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

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

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

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

The complete group of subprograms listed in Summary of DBMS_SQLSET Subprograms facilitates this functionality. As examples:

- The CREATE_SQLSET Procedure and Function creates a SQL tuning set object in the database.
- The LOAD_SQLSET Procedure populates the SQL tuning set with a set of selected SQL.
- The CAPTURE_CURSOR_CACHE Procedure collects SQL statements from the shared SQL area over a specified time interval, attempting to build a realistic picture of database workload.

DBMS_SQLSET Security Model

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

SQL tuning set subprograms require either the `ADMINISTER SQL TUNING SET` or the `ADMINISTER ANY SQL TUNING SET` privilege. Users having the `ADMINISTER SQL TUNING SET` privilege can only create and modify a SQL tuning set that they own. The `ADMINISTER ANY SQL TUNING SET` privilege allows users to manipulate all SQL tuning sets, even those owned by other users. For example, you can use CREATE_SQLSET Procedure and Function to create a SQL tuning set to be owned by a different user. In this case, the different user need not have the `ADMINISTER SQL TUNING SET` privilege to manipulate the SQL tuning set.

DBMS_SQLSET Data Structures

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

Object Types

- SQLSET_ROW Object Type

SQLSET_ROW Object Type

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

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

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

Syntax

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

Attributes

Table 194-1 SQLSET_ROW Attributes

Attribute	Description
`sql_id`	Unique SQL ID.
`forcing_matching_signature`	Signature with literals, case, and whitespace removed.
`sql_text`	Full text for the SQL statement.
`object_list`	Currently not implemented.
`bind_data`	Bind data as captured for this SQL. Note that you cannot stipulate an argument for this parameter and also for `bind_list` - they are mutually exclusive.
`parsing_schema_name`	Schema where the SQL is parsed.
`module`	Last application module for the SQL.
`action`	Last application action for the SQL.
`elapsed_time`	Sum total elapsed time for this SQL statement.
`cpu_time`	Sum total CPU time for this SQL statement.
`buffer_gets`	Sum total number of buffer gets.
`disk_reads`	Sum total number of disk reads.
`direct_writes`	Sum total number of direct path writes.
`rows_processed`	Sum total number of rows processed by this SQL.
`fetches`	Sum total number of fetches.
`executions`	Total executions of this SQL statement.

Table 194-1 (Cont.) SQLSET_ROW Attributes

Attribute	Description
`end_of_fetch_count`	Number of times the SQL statement was fully executed with all of its rows fetched.
`optimizer_cost`	Optimizer cost for this SQL.
`optimizer_env`	Optimizer environment for this SQL statement.
`priority`	User-defined priority (1,2,3).
`command_type`	Statement type, such as `INSERT` or `SELECT`.
`first_load_time`	Load time of the parent cursor.
`stat_period`	Period of time (seconds) when the statistics of this SQL statement were collected.
`active_stat_period`	Effective period of time (in seconds) during which the SQL statement was active.
`other`	Other column for user-defined attributes.
`plan_hash_value`	Plan hash value of the plan.
`sql_plan`	Execution plan for the SQL statement.
`bind_list`	List of user-specified binds for the SQL statement. This is used for user-specified workloads. Note that you cannot stipulate an argument for this parameter and also for `bind_data`: they are mutually exclusive.
`con_dbid`	DBID of the PDB or CDB root.
`last_exec_start_time`	Most recent execution start time of this SQL statement.

Summary of DBMS_SQLSET Subprograms

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

Table 194-2 DBMS_SQLSET Package Subprograms

Subprogram	Description
ADD_REFERENCE Function	This procedure adds a new reference to an existing SQL tuning set to indicate its use by a client.
CAPTURE_CURSOR_CACHE Procedure	This procedure captures a workload from the shared SQL area into a SQL tuning set.
CREATE_SQLSET Procedure and Function	This procedure or function creates a SQL tuning set object in the database.
CREATE_STGTAB Procedure	This procedure creates a staging table through which SQL tuning sets are imported and exported.
DELETE_SQLSET Procedure	This procedure deletes a set of SQL statements from a SQL tuning set.
DROP_SQLSET Procedure	This procedure drops a SQL tuning set if it is not active.

Table 194-2 (Cont.) DBMS_SQLSET Package Subprograms

Subprogram	Description
LOAD_SQLSET Procedure	This procedure populates the SQL tuning set with a set of selected SQL statements. You can call the procedure multiple times to add new SQL statements or replace attributes of existing statements.
PACK_STGTAB Procedure	This procedure copies one or more SQL tuning sets from their location in the `SYS` schema to a staging table created by the `CREATE_STGTAB` procedure.
REMAP_STGTAB Procedure	This procedure changes the tuning set names and owners in the staging table so that they can be unpacked with different values.
REMOVE_REFERENCE Procedure	This procedure deactivates a SQL tuning set to indicate that it is no longer used by the client.
SELECT_CURSOR_CACHE Function	This function collects SQL statements from the workload repository.
SELECT_SQL_TRACE Function	This table function reads the content of one or more trace files and returns the SQL statements it finds in the format of `sqlset_row`.
SELECT_SQLPA_TASK Function	This function collects SQL statements from a SQL Performance Analyzer comparison task.
SELECT_SQLSET Function	This is a table function that reads the contents of a SQL tuning set.
SELECT_WORKLOAD_REPOSITORY Function	This function collects SQL statements from the workload repository.
UNPACK_STGTAB Procedure	This procedure copies one or more SQL tuning sets from their location in the staging table into the SQL tuning sets schema, making them proper SQL tuning sets.
UPDATE_SQLSET Procedures	This overloaded procedure updates selected fields for SQL statements in a SQL tuning set.

ADD_REFERENCE Function

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

Syntax

```
DBMS_SQLSET.ADD_REFERENCE (
   sqlset_name  IN  VARCHAR2,
   description  IN  VARCHAR2 := NULL,
   sqlset_owner IN  VARCHAR2 :=NULL)
 RETURN NUMBER;
```

Parameters

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

Table 194-3 ADD_SQLSET_REFERENCE and ADD_REFERENCE Function Parameters

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

Return Values

The identifier of the added reference.

Usage Notes

Adding a reference to a SQL tuning set prevents the tuning set from being modified while it is being used. Invoking SQL Tuning Advisor on the SQL tuning set adds a reference automatically, so use `ADD_REFERENCE` only when the automatically generated reference is not sufficient. The `ADD_REFERENCE` function returns a reference ID that you can later supply to the `REMOVE_SQLSET_REFERENCE` procedure. Query the `DBA_SQLSET_REFERENCES` view to find all references to a specified SQL tuning set.

Examples

This example generates a reference to the SQL tuning set named `my_workload` and stores it in the `b_rid` variable.

```
VARIABLE b_rid NUMBER;
EXEC :b_rid := DBMS_SQLSET.ADD_REFERENCE(sqlset_name => 'my_workload',
description => 'my sts ref');
```

CAPTURE_CURSOR_CACHE Procedure

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

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

Syntax

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

Parameters

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

Table 194-4 CAPTURE_CURSOR_CACHE_SQLSET and CAPTURE_CURSOR_CACHE Procedure Parameters

Parameter	Description
`sqlset_name`	Specifies the SQL tuning set name
`time_limit`	Defines the total amount of time, in seconds, to execute.
`repeat_interval`	Defines the amount of time, in seconds, to pause between sampling.
`capture_option`	Specifies whether to insert new statements, update existing statements, or both. Values are `INSERT`, `UPDATE`, or `MERGE`. The values are the same as for `load_option` in `load_sqlset`.
`capture_mode`	Specifies the capture mode (`UPDATE` and `MERGE` capture options). Possible values: • `MODE_REPLACE_OLD_STATS` — Replaces statistics when the number of executions is greater than the number stored in the SQL tuning set • `MODE_ACCUMULATE_STATS` — Adds new values to current values for SQL that is already stored. Note that this mode detects if a statement has been aged out, so the final value for a statistics is the sum of the statistics of all cursors that statement existed under.
`basic_filter`	Defines a filter to apply to the shared SQL area for each sample. If `basic_filter` is not set by the caller, then the subprogram captures only statements of type `CREATE TABLE`, `INSERT`, `SELECT`, `UPDATE`, `DELETE`, and `MERGE`.
`sqlset_owner`	Specifies the owner of the SQL tuning set or `NULL` for current schema owner
`recursive_sql`	Defines a filter that includes recursive SQL in the SQL tuning set (`HAS_RECURSIVE_SQL`) or excludes it (`NO_RECURSIVE_SQL`).

Examples

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

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

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

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

```
EXEC DBMS_SQLSET.CAPTURE_CURSOR_CACHE( -
                         sqlset_name     => 'my_workload', -
                         time_limit      => 30, -
                         repeat_interval => 5, -
                         capture_mode    =>
DBMS_SQLSET.MODE_ACCUMULATE_STATS);
```

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

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

CREATE_SQLSET Procedure and Function

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

Syntax

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

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

Parameters

Table 194-5 CREATE_SQLSET Procedure Parameters

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

Return Values

Specifies the name of the created SQL tuning set. The name is the name passed to the function. If no name is passed to the function, then the function generates an automatic name.

Examples

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

CREATE_STGTAB Procedure

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

Syntax

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

Parameters

Table 194-6 CREATE_STGTAB_SQLSET and CREATE_STGTAB Procedure Parameters

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

Examples

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

```
EXEC DBMS_SQLSET.CREATE_STGTAB(table_name => 'STGTAB_SQLSET');
```

Create a staging table to pack a SQL tuning set in Oracle Database 12*c* Release 1 (12.1.0.2) format:

```
BEGIN
  DBMS_SQLSET.CREATE_STGTAB(
      table_name => 'STGTAB_SQLSET_121'
  ,   db_version => DBMS_SQLSET.STS_STGTAB_12_1_VERSION );
END;
```

DELETE_SQLSET Procedure

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

Syntax

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

Parameters

Table 194-7 DELETE_SQLSET Procedure Parameters

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

Examples

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

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

DROP_SQLSET Procedure

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

Syntax

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

Parameters

Table 194-8 DROP_SQLSET Procedure Parameters

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

Usage Notes

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

Examples

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

LOAD_SQLSET Procedure

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

Syntax

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

Parameters

Table 194-9 LOAD_SQLSET Procedure Parameters

Parameter	Description
`sqlset_name`	Specifies the name of SQL tuning set to be loaded.
`populate_cursor`	Specifies the cursor reference to the SQL tuning set to be loaded.
`load_option`	Specifies which statements are loaded into the SQL tuning set. The possible values are: • `INSERT` (default) — Adds only new statements. • `UPDATE` — Updates existing the SQL statements and ignores any new statements. • `MERGE` — Inserts new statements and updates the information of the existing ones.

Table 194-9 (Cont.) LOAD_SQLSET Procedure Parameters

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

Table 194-9 (Cont.) LOAD_SQLSET Procedure Parameters

Parameter	Description
`sqlset_owner`	Defines the owner of the SQL tuning set, or the current schema owner (or `NULL` for the current owner).

Exceptions

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

Usage Notes

Rows in the input `populate_cursor` must be of type `SQLSET_ROW`.

Examples

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

```
-- create the tuning set
EXEC DBMS_SQLSET.CREATE_SQLSET('my_workload');

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

DBMS_SQLSET.LOAD_SQLSET(sqlset_name     => 'my_workload',
                        populate_cursor => cur);

END;
/
```

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

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

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

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

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

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

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

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

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

PACK_STGTAB Procedure

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

Syntax

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

Parameters

The parameters are identical for the DBMS_SQLTUNE.PACK_STGTAB_SQLSET and DBMS_SQLSET.PACK_STGTAB procedures.

Table 194-10 PACK_STGTAB_SQLSET and PACK_STGTAB Procedure Parameters

Parameter	Description
sqlset_name	Specifies the name of the SQL tuning set to pack. The name is case sensitive. Wildcard characters (%) are permitted.
sqlset_owner	Specifies the category from which to pack SQL tuning sets. The name is case sensitive. Wildcard characters (%) are permitted.
staging_table_name	Specifies the name of the table to use. The value is case sensitive.
staging_schema_owner	Specifies the schema where the table resides, or NULL for the current schema. The value is case sensitive.
db_version	Specifies the database version that determines the format of the staging table. You can also create an older database version staging table to export an STS to an older database version. Use any of the following values: • NULL (default) — Specifies the current database version. • STS_STGTAB_10_2_VERSION — Specifies the 10.2 database version. • STS_STGTAB_11_1_VERSION — Specifies the 11.1 database version. • STS_STGTAB_11_2_VERSION — Specifies the 11.2 database version.

Usage Notes

- To move more than one SQL tuning set, call this procedure multiple times. You can then move the populated staging table to a destination database using any method, such as a database link or Oracle Data Pump, and then unpack the SQL tuning set in the destination database.
- This function issues a COMMIT after packing each SQL tuning set. If an error is raised mid-execution, then clear the staging table by deleting its rows.

Examples

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

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

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

```
BEGIN
  DBMS_SQLSET.PACK_STGTAB(
      sqlset_name        => '%'
  ,   staging_table_name => 'STGTAB_SQLSET');
END;
```

Pack a specific SQL tuning set:

```
BEGIN
  DBMS_SQLSET.PACK_STGTAB(
      sqlset_name          => 'my_workload'
  ,   staging_table_name   => 'STGTAB_SQLSET');
END;
```

Pack a second SQL tuning set:

```
BEGIN
  DBMS_SQLSET.PACK_STGTAB(
      sqlset_name          => 'workload_subset'
  ,   staging_table_name   => 'STGTAB_SQLSET');
END;
```

Pack the STS `my_workload_subset` into a staging table `stgtab_sqlset` created for Oracle Database 11*g* Release 2 (11.2):

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

REMAP_STGTAB Procedure

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

Syntax

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

Parameters

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

Table 194-11 REMAP_STGTAB_SQLSET and REMAP_SQLSET Procedure Parameters

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

Usage Notes

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

Examples

```
-- Change the name of an STS in the staging table before unpacking it.
BEGIN
  DBMS_SQLSET.REMAP_STGTAB(
      old_sqlset_name    =>
'my_workload'
  ,   old_sqlset_owner   => 'SH'
  ,   new_sqlset_name    =>
'imp_workload'
  ,   staging_table_name => 'STGTAB_SQLSET');

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

REMOVE_REFERENCE Procedure

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

Syntax

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

Parameters

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

Table 194-12 REMOVE_SQLSET_REFERENCE and REMOVE_REFERENCE Procedure Parameters

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

Examples

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

```
EXEC DBMS_SQLSET.REMOVE_REFERENCE(sqlset_name   => 'my_workload', -
                                  reference_id  => :rid);
```

Use the `DBA_SQLSET_REFERENCES` view to find all references to a given SQL tuning set.

SELECT_CURSOR_CACHE Function

This function collects SQL statements from the shared SQL area.

Syntax

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

Parameters

Table 194-13 SELECT_CURSOR_CACHE Function Parameters

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

Table 194-13 (Cont.) SELECT_CURSOR_CACHE Function Parameters

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

Return Values

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

Usage Notes

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

Example 194-1 Statements with 500 or More Buffer Gets

This query obtains the SQL IDs and SQL text for statements with 500 buffer gets:

```
SELECT SQL_ID, SQL_TEXT
FROM   TABLE(DBMS_SQLSET.SELECT_CURSOR_CACHE('buffer_gets > 500'))
ORDER BY sql_id;
```

Example 194-2 All Information About a Statement

The following query obtains all information about the SQL statement with the SQL ID `4rm4183czbs7j`:

```
SELECT * FROM TABLE(DBMS_SQLSET.SELECT_CURSOR_CACHE('sql_id =
''4rm4183czbs7j'''));
```

Example 194-3   Multiple Plans for a SQL Statement

A data source may store multiple plans for each SQL statement. The output of the `SELECT_CURSOR_CACHE` function is a SQL row set object that is uniquely identified by SQL ID and plan hash value. This example queries the plan hash values for the statement with the SQL ID `ay1m3ssvtrh24`:

```
SELECT sql_id, plan_hash_value
FROM table(DBMS_SQLSET.select_cursor_cache('sql_id = ''ay1m3ssvtrh24'''))
ORDER BY sql_id, plan_hash_value;
```

Example 194-4   Processing All Statements in the Shared SQL Area

This example processes all statements in the shared SQL area:

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(p)
    FROM TABLE(DBMS_SQLSET.SELECT_CURSOR_CACHE) p;

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

  CLOSE cur;
END;
/
```

Example 194-5   Process Statements Not Parsed by SYS

This example processes all statements not parsed in the `SYS` schema:

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur for
    SELECT VALUE(p)
    FROM TABLE(
     DBMS_SQLSET.SELECT_CURSOR_CACHE('parsing_schema_name <> ''SYS''')) p;

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

  CLOSE cur;
end;
/
```

Example 194-6   All Statements from an Application Module and Action

This example processes all statements from a specified application module and action:

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(p)
    FROM TABLE(
      DBMS_SQLSET.SELECT_CURSOR_CACHE(
         'module = ''MY_APPLICATION'' and action = ''MY_ACTION''')) p;

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

  CLOSE cur;
END;/
```

Example 194-7   All Statements Whose Elapsed Time Is At Least Five Seconds

This example processes all statements that ran for at least five seconds:

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLSET.SELECT_CURSOR_CACHE('elapsed_time > 5000000')) P;

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

  CLOSE cur;
END;
/
```

Example 194-8   Statements Parsed in the APPS Schema

This example processes all SQL statements that were parsed in the `APPS` schema and had more than 100 buffer gets:

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(p)
    FROM TABLE(
      DBMS_SQLSET.SELECT_CURSOR_CACHE(
        'buffer_gets > 100 and parsing_schema_name = ''APPS'''))p;

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

  CLOSE cur;
END;
/
```

Example 194-9   Plans and SQL Statements

This example processes all SQL statements exceeding 5 seconds. It also selects the plans for these statements. For performance reasons, the example selects execution statistics and SQL binds. The `SQL_PLAN` attribute of `sqlset_row` is `NULL`.

```
-- select all statements exceeding 5 seconds in elapsed time, but also
-- select the plans (by default we only select execution stats and binds
-- for performance reasons - in this case the SQL_PLAN attribute of sqlset_row
-- is NULL)
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(p)
    FROM TABLE(DBMS_SQLSET.SELECT_CURSOR_CACHE(
      basic_filter      => 'elapsed_time > 5000000',
      object_filter     => NULL,
      ranking_measure1  => NULL,
      ranking_measure2  => NULL,
      ranking_measure3  => NULL,
      result_percentage => 1,
      result_limit      => NULL,
      attribute_list    => 'EXECUTION_STATISTICS, SQL_BINDS, SQL_PLAN',
      recursive_sql     => HAS_RECURSIVE_SQL)) p;

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

  CLOSE cur;
END;/
```

Example 194-10   Top 100 Statements Ordered by Elapsed Time

This example selects the top 100 statements in the shared SQL area, ordered by elapsed time:

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(p)
    FROM TABLE(DBMS_SQLSET.SELECT_CURSOR_CACHE(
      basic_filter      => NULL,
      object_filter     => NULL,
      ranking_measure_1 => 'ELAPSED_TIME',
      ranking_measure_2 => NULL,
      ranking_measure_3 => NULL,
      result_percentage => 1,
      result_limit      => 100,
      attribute_list    => 'TYPICAL',
      recursive_sql     => HAS_RECURSIVE_SQL))) p;

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

  CLOSE cur;
END;
/
```

Example 194-11 Statements Responsible for Most Buffer Gets

This example processes statements that cumulatively account for 90% of the buffer gets in the shared SQL area. The buffer gets of all statements added together is approximately 90% of the sum of all statements currently in the shared SQL area.

```
DECLARE
  cur sys_refcursor;
BEGIN
  OPEN cur FOR
    SELECT VALUE(P)
    FROM table(DBMS_SQLSET.SELECT_CURSOR_CACHE(
      basic_filter       => NULL,
      object_filter      => NULL,
      ranking_measure_1 => 'BUFFER_GETS',
      ranking_measure_2 => NULL,
      ranking_measure_3 => NULL,
      result_percentage => .9,
      result_limit       => NULL,
      attribute_list     => 'TYPICAL',
      recursive_sql      => HAS_RECURSIVE_SQL))) p;

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

  CLOSE cur;
END;
/
```

SELECT_SQL_TRACE Function

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

Syntax

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

Parameters

Table 194-14 SELECT_SQL_TRACE Function Parameters

Parameter	Description
`directory`	Defines the directory object containing the trace files. This field is mandatory.

Table 194-14 (Cont.) SELECT_SQL_TRACE Function Parameters

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

Return Values

This function returns a `SQLSET_ROW` object.

Usage Notes

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

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

Examples

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

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

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

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

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

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

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

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

See Also:

Oracle Database SQL Language Reference to learn more about the `PATH_PREFIX` clause

SELECT_SQLPA_TASK Function

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

See Also:

Oracle Database Testing Guide for a `SELECT_SQLPA_TASK` example

Syntax

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

Parameters

Table 194-15 SELECT_SQLPA_TASK Function Parameters

Parameter	Description
`task_name`	Specifies the name of the SQL Performance Analyzer task.
`task_owner`	Specifies the owner of the SQL Performance Analyzer task. If `NULL`, then assume the current user.
`execution_name`	Specifies the name of the SQL Performance Analyzer task execution (type `COMPARE PERFORMANCE`) from which the provided filters will be applied. If `NULL`, then assume the most recent `COMPARE PERFORMANCE` execution.
`level_filter`	Specifies which subset of SQL statements to include. Same format as `DBMS_SQLPA.REPORT_ANALYSIS_TASK.LEVEL`, with some possible strings removed. • `IMPROVED` includes only improved SQL. • `REGRESSED` includes only regressed SQL (default). • `CHANGED` includes only SQL with changed performance. • `UNCHANGED` includes only SQL with unchanged performance. • `CHANGED_PLANS` includes only SQL with plan changes. • `UNCHANGED_PLANS` includes only SQL with unchanged plans. • `ERRORS` includes only SQL with errors only. • `MISSING_SQL` includes only missing SQL statements (across STS). • `NEW_SQL` includes only new SQL statements (across STS).
`basic_filter`	Specifies the SQL predicate to filter the SQL in addition to the level filters.
`object_filter`	Currently not supported.

Table 194-15 (Cont.) SELECT_SQLPA_TASK Function Parameters

Parameter	Description
`attribute_list`	Defines the SQL statement attributes to return in the result. Possible values are: • `TYPICAL` — Returns `BASIC` plus the SQL plan (without row source statistics) and without an object reference list. This is the default. • `BASIC` — Returns all attributes (such as execution statistics and binds) except the plans. The execution context is always part of the result. • `ALL` — Returns all attributes. • Comma-separated list of attribute names this allows to return only a subset of SQL attributes: `EXECUTION_STATISTICS`, `SQL_BINDS`, `SQL_PLAN_STATISTICS` (similar to `SQL_PLAN` + row source statistics).

Return Values

This function returns a SQL tuning set object.

Usage Notes

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

SELECT_SQLSET Function

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

Syntax

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

Parameters

Table 194-16 SELECT_SQLSET Function Parameters

Parameter	Description
`sqlset_name`	Specifies the name of the SQL tuning set to query.

Table 194-16 (Cont.) SELECT_SQLSET Function Parameters

Parameter	Description
`basic_filter`	Specifies the SQL predicate to filter the SQL from the SQL tuning set defined on attributes of the `SQLSET_ROW`.
`object_filter`	Currently not supported.
`ranking_measure(n)`	Specifies an `ORDER BY` clause on the selected SQL.
`result_percentage`	Specifies a filter that picks the top *n*% according to the supplied ranking measure. Note that this parameter applies only if one ranking measure is supplied.
`result_limit`	The top limit SQL from the filtered source, ranked by the ranking measure.
`attribute_list`	Defines the SQL statement attributes to return in the result. The possible values are: • `BASIC` — Returns all attributes (such as execution statistics and binds) except the plans. The execution context is included in the result. • `TYPICAL` — Returns `BASIC` plus the SQL plan, but without row source statistics and without the object reference list. This is the default. • `ALL` — Returns all attributes. • Comma-separated list of attribute names. This value enables the function to return only a subset of SQL attributes: – `EXECUTION_STATISTICS` – `SQL_BINDS` – `SQL_PLAN_STATISTICS` (similar to `SQL_PLAN` plus row source statistics)
`plan_filter`	Specifies the plan filter. This parameter enables you to select a single plan when a statement has multiple plans. Possible values are: • `LAST_GENERATED` — Returns the plan with the most recent timestamp. • `FIRST_GENERATED` — Returns the plan with the least recent timestamp. • `LAST_LOADED` — Returns the plan with the most recent `FIRST_LOAD_TIME` statistical information. • `FIRST_LOADED` — Returns the plan with the least recent `FIRST_LOAD_TIME` statistical information. • `MAX_ELAPSED TIME` — Returns the plan with the maximum elapsed time. • `MAX_BUFFER_GETS` — Returns the plan with the maximum buffer gets. • `MAX_DISK_READS` — Returns the plan with the maximum disk reads. • `MAX_DIRECT_WRITES` — Returns the plan with the maximum direct writes. • `MAX_OPTIMIZER_COST` — Returns the plan with the maximum optimizer cost value.
`sqlset_owner`	Specifies the owner of the SQL tuning set, or `NULL` for the current schema owner.
`recursive_sql`	Specifies that the filter must include recursive SQL in the SQL tuning set (`HAS_RECURSIVE_SQL`, which is the default) or exclude it (`NO_RECURSIVE_SQL`).

Return Values

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

Usage Notes

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

Examples

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

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

  CLOSE cur;
END;
/
```

SELECT_WORKLOAD_REPOSITORY Function

This function collects SQL statements from the workload repository.

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

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

Syntax

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

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

Parameters

Table 194-17 SELECT_WORKLOAD_REPOSITORY Function Parameters

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

Return Values

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

Usage Notes

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

Examples

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

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

  CLOSE cur;
END;
/
```

UNPACK_STGTAB Procedure

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

Syntax

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

Parameters

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

Table 194-18 UNPACK_STGTAB_SQLSET and UNPACK_STGTAB Procedure Parameters

Parameter	Description
`sqlset_name`	Specifies the name of the tuning set to unpack (not null). Wildcard characters (`%`) are supported to unpack multiple tuning sets in a single call. For example, specify `%` to unpack all tuning sets from the staging table.

Table 194-18 (Cont.) UNPACK_STGTAB_SQLSET and UNPACK_STGTAB Procedure Parameters

Parameter	Description
`sqlset_owner`	Specifies the name of tuning set owner, or `NULL` for the current schema owner. Wildcard characters (`%`) are supported.
`replace`	Specifies whether to replace an existing SQL tuning set. If `FALSE`, then this procedure raises errors when you try to create a tuning set that already exists.
`staging_table_name`	Specifies the name of the staging table, moved after a call to the `DBMS_SQLTUNE.PACK_STGTAB_SQLSET` or `DBMS_SQLSET.PACK_STGTAB` procedure (case-sensitive).
`staging_schema_owner`	Specifies the name of staging table owner, or `NULL` for the current schema owner (case-sensitive).

Examples

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

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

UPDATE_SQLSET Procedures

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

Syntax

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

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

Parameters

Table 194-19 UPDATE_SQLSET Procedure Parameters

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