186
DBMS_SPM

The `DBMS_SPM` package supports the SQL plan management feature by providing an interface for the DBA or other user to perform controlled manipulation of plan history and SQL plan baselines maintained for various SQL statements.

This chapter contains the following topics:

> **See Also:**
>
> For more information about "Using SQL Plan Management" in the *Oracle Database SQL Tuning Guide*

DBMS_SPM Overview

The `DBMS_SPM` package allows the user to manage SQL execution plans using SQL plan management.

SQL plan management prevents performance regressions resulting from sudden changes to the execution plan of a SQL statement by recording and evaluating the execution plans of SQL statements over time, and builds SQL plan baselines composed of a set of existing plans known to be efficient. The SQL plan baselines are then used to preserve performance of corresponding SQL statements, regardless of changes occurring in the system. Common usage scenarios where SQL plan management can improve or preserve SQL performance include:

- A database upgrade that installs a new optimizer version usually results in plan changes for a small percentage of SQL statements, with most of the plan changes resulting in either no performance change or improvement. However, certain plan changes may cause performance regressions. The use of SQL plan baselines significantly minimizes potential performance regressions resulting from a database upgrade.
- Ongoing system and data changes can impact plans for some SQL statements, potentially causing performance regressions. The use of SQL plan baselines helps to minimize performance regressions and stabilize SQL performance.
- Deployment of new application modules means introducing new SQL statements into the system. The application software may use appropriate SQL execution plans developed under a standard test configuration for the new SQL statements. If the system production

configuration differs significantly from the test configuration, SQL plan baselines can be evolved over time to produce better performance.

DBMS_SPM Security Model

The package is owned by `SYS`. The `EXECUTE` package privilege is required to execute its procedures. Any user granted the `ADMINISTER SQL MANAGEMENT OBJECT` privilege is able to execute the `DBMS_SPM` package.

DBMS_SPM Constants

The `DBMS_SPM` package provides constants that can be used for specifying parameter values.

These are shown in the following table. These constants are defined as standard input for the `time_limit` parameter of the EVOLVE_SQL_PLAN_BASELINE Function.

Table 186-1 DBMS_SPM Constants

Constant	Type	Value	Description
AUTO_LIMIT	INTEGER	2147483647	Oracle determines the appropriate time spent by the EVOLVE_SQL_PLAN_BASELINE Function.
NO_LIMIT	INTEGER	2147483647 -1	There is no limit to the time spent by the EVOLVE_SQL_PLAN_BASELINE Function.

DBMS_SPM Examples

These examples will help you understand use of `DBMS_SPM`.

Detailed examples are located under the following topics:

- Migrating Stored Outlines to SQL Plan Baselines
- Migrating Outlines to Utilize SQL Plan Management Features
- Migrating Outlines to Preserve Stored Outline Behavior
- Performing Follow-Up Tasks After Stored Outline Migration

DBMS_SPM Data Structures

The `DBMS_SPM` package defines a `TABLE` type.

Table Types

- DBMS_SPM NAMELIST Table Type

DBMS_SPM NAMELIST Table Type

This type allows for a list of names as an input parameter.

Syntax

```
TYPE  name_list  IS TABLE OF VARCHAR2(30);
```

Summary of DBMS_SPM Subprograms

This table lists and briefly describes the `DBMS_SPM` package subprograms.

Table 186-2 DBMS_SPM Package Subprograms

Subprogram	Description
ACCEPT_SQL_PLAN_BASELINE Procedure	Accepts a plan based on the recommendation of an evolve task
ADD_VERIFIED_SQL_PLAN_BASELINE Function	Finds the plans available for the given SQL_ID in different sources such Cursor Cache, Auto SQL Tuning Set, and Automatic Workload Repository.
ALTER_SQL_PLAN_BASELINE Function	Changes an attribute of a single plan or all plans associated with a SQL statement using the attribute name/value format
CANCEL_EVOLVE_TASK Procedure	Cancels a currently executing evolve task
CONFIGURE Procedure	Sets configuration options for SQL management base, in parameter/value format
CREATE_EVOLVE_TASK Function	Creates an advisor task and sets its parameters
CREATE_STGTAB_BASELINE Procedure	Creates a staging table that used for transporting SQL plan baselines from one system to another
DROP_EVOLVE_TASK Procedure	Drops an evolved task
DROP_MIGRATED_STORED_OUTLINE Function	Drops all stored outlines that have been migrated to SQL plan baselines.
DROP_SQL_PLAN_BASELINE Function	Drops a single plan, or all plans associated with a SQL statement
EVOLVE_SQL_PLAN_BASELINE Function	Evolves SQL plan baselines associated with one or more SQL statements
EXECUTE_EVOLVE_TASK Function	Executes a previously created evolve task
INTERRUPT_EVOLVE_TASK Procedure	Interrupts a currently executing evolve task
LOAD_PLANS_FROM_CURSOR_CACHE Functions	Loads one or more plans present in the cursor cache for a SQL statement
LOAD_PLANS_FROM_AWR Function	Loads the SQL Management Base (SMB) with SQL plan baselines for a set of SQL statements using the plans from the AWR, and returns the number of plans loaded
LOAD_PLANS_FROM_SQLSET Function	Loads plans stored in a SQL tuning set (STS) into SQL plan baselines

Table 186-2 (Cont.) DBMS_SPM Package Subprograms

Subprogram	Description
MIGRATE_STORED_OUTLINE Functions	Migrates existing stored outlines to SQL plan baselines
PACK_STGTAB_BASELINE Function	Packs (exports) SQL plan baselines from SQL management base into a staging table
RESET_EVOLVE_TASK Procedure	Resets an evolve task to its initial state
RESUME_EVOLVE_TASK Procedure	Resumes a previously interrupted task
REPORT_AUTO_EVOLVE_TASK Function	Displays the results of an execution of an automatic evolve task.
REPORT_EVOLVE_TASK Function	Displays the results of an evolved task
SET_EVOLVE_TASK_PARAMETER Procedure	Sets a parameter of an evolve task
UNPACK_STGTAB_BASELINE Function	Unpacks (imports) SQL plan baselines from a staging table into SQL management base

ACCEPT_SQL_PLAN_BASELINE Procedure

The procedure accepts a plan based on the recommendation of an evolve task.

Syntax

```
DBMS_SPM.ACCEPT_SQL_PLAN_BASELINE  (
   task_name        IN  VARCHAR2,
   object_id        IN  NUMBER    := NULL,
   task_owner       IN  VARCHAR2  := NULL,
   force            IN  BOOLEAN   := FALSE);
```

Parameters

Table 186-3 ACCEPT_SQL_PLAN_BASELINE Procedure Parameters

Parameter	Description
task_name	Identifier of task to implement
object_id	Identifier of the advisor framework object that represents a single plan. If NULL, the report is generated for all objects.
task_owner	Owner of the evolve task. Defaults to the current schema owner.
force	Accept the plan even if the advisor did not recommend such an action. The default is FALSE requiring acceptance of the plan only if the plan is verified and shows sufficient improvement in benefit.

ADD_VERIFIED_SQL_PLAN_BASELINE Function

This function finds the plans available for the given SQL_ID in different sources such Cursor Cache, Auto SQL Tuning Set, and Automatic Workload Repository.

Syntax

```
DBMS_SPM.ADD_VERIFIED_SQL_PLAN_BASELINE (
    sql_id IN VARCHAR2
);
```

Parameters

Table 186-4 ADD_VERIFIED_SQL_PLAN_BASELINE Function Parameters

Parameter	Description
sql_id	The SQL statement identifier, which is used to identify the plans in different sources, such as the cursor cache.

Return Value

A CLOB containing a report of SQL plan baselines verified, reproduced and accepted.

Usage Notes

SQL plan baselines are created to ensure that the best-known plans are used for the selected SQL statement. These are the steps that the function executes:

1. Loads plans from Cursor Cache, Automatic Workload Repository, and Auto SQL Tuning Set (SYS_AUTO_STS) into SQL Plan Management SQL plan history in a non-accepted state.
2. Uses the SQL Plan Management Evolve Advisor internally to identify the best-performing execution plans. The best-performing plans are accepted.

> **Note:**
>
> "Administer SQL Management Object" privilege. is required.

Examples

Example 1: 36k32wnz0v0fd923079310

```
select sql_id, plan_hash_value, sql_text
from   v$sql
where  sql_text like 'select /* SPM_TEST_QUERY%'
/

SQL_ID          PLAN_HASH_VALUE  SQL_TEXT
--------------- ---------------
--------------------------------------------------------------------
36k32wnz0v0fd   923079310        select /* SPM_TEST_QUERY */ num from
example_spm_table where id = 100
```

Example 2:9230793102448381833

```
select sql_text, sqlset_name, plan_hash_value
from   dba_sqlset_statements
where  sql_text like 'select /* SPM_TEST_QUERY%';

SQL_TEXT
SQLSET_NAME   PLAN_HASH_VALUE
------------------------------------------------------------------------------
------------ ---------------
select /* SPM_TEST_QUERY */ num from example_spm_table where id = 100
SYS_AUTO_STS   923079310
select /* SPM_TEST_QUERY */ num from example_spm_table where id = 100
SYS_AUTO_STS   2448381833
```

Example 3:

```
set tab off
set serveroutput on
set pagesize 100
set linesize 250
set  long 100000
column report format a200

var rep clob;

BEGIN
   :rep := DBMS_SPM. ADD_VERIFIED_SQL_PLAN_BASELINE('36k32wnz0v0fd');
END;
/

select :rep report from dual;

SQL Plan Baselines verified for SQL ID: 36k32wnz0v0fd
------------------------------------------------------------------------------
------------

Plan Hash Value   Plan Name                        Reproduced   Accepted
Source
---------------   ------------------------------   ----------   --------
-------------
923079310         SQL_PLAN_163tr5qgzwmgt05ce4c2e   YES          YES
CURSOR CACHE
2448381833        SQL_PLAN_163tr5qgzwmgt1f191f3e   YES          NO
SQL TUNING SET
------------------------------------------------------------------------------
-----------

SQL Handle    : SQL_130f372d9ffe4df9
SQL Text      : select /* SPM_TEST_QUERY */ num from example_spm_table where
id = 100
------------------------------------------------------------------------------
-----------
```

Example 4:

```
select sql_text, accepted, enabled, sql_handle, plan_name
from   dba_sql_plan_baselines
where  sql_text like 'select /* SPM_TEST_QUERY%';

SQL_TEXT                                                                  ACC
ENA SQL_HANDLE           PLAN_NAME
------------------------------------------------------------------------ ---
--- -------------------- ------------------------------
select /* SPM_TEST_QUERY */ num from example_spm_table where id = 100  YES
YES SQL_130f372d9ffe4df9 SQL_PLAN_163tr5qgzwmgt05ce4c2e
select /* SPM_TEST_QUERY */ num from example_spm_table where id = 100  NO
YES SQL_130f372d9ffe4df9 SQL_PLAN_163tr5qgzwmgt1f191f3e
```

Example 5

```
select * from
table(dbms_xplan.display_sql_plan_baseline('SQL_130f372d9ffe4df9'));

--------------------------------------------------------------------------------
--
SQL handle: SQL_130f372d9ffe4df9
SQL text: select /* SPM_TEST_QUERY */ num from example_spm_table where id =
100
--------------------------------------------------------------------------------
--

--------------------------------------------------------------------------------
--
Plan name: SQL_PLAN_163tr5qgzwmgt05ce4c2e         Plan id: 923079310
Enabled: YES     Fixed: NO      Accepted: YES     Origin: EVOLVE-LOAD-FROM-
CURSOR-CACHE
Plan rows: From Auto SQL Tuning Set
--------------------------------------------------------------------------------
--

Plan hash value: 923079310

--------------------------------------------------------------------------------
------------------
| Id  | Operation                    | Name              | Rows  | Bytes |
Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
------------------
|   0 | SELECT STATEMENT             |                   |       |       |
2 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| EXAMPLE_SPM_TABLE |     1 |     8 |
2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | SPM_TAB_PK        |     1 |       |
1   (0)| 00:00:01 |
--------------------------------------------------------------------------------
------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("ID"=100)

-------------------------------------------------------------------------------
--
Plan name: SQL_PLAN_163tr5qgzwmgt1f191f3e         Plan id: 2448381833
Enabled: YES     Fixed: NO      Accepted: NO      Origin: EVOLVE-LOAD-FROM-STS
Plan rows: From Auto SQL Tuning Set
-------------------------------------------------------------------------------
--

Plan hash value: 2448381833

-------------------------------------------------------------------------------
---------
| Id  | Operation         | Name              | Rows  | Bytes | Cost (%CPU)|
Time     |
-------------------------------------------------------------------------------
---------
|   0 | SELECT STATEMENT  |                   |       |       |     3
(100)|          |
|*  1 |  TABLE ACCESS FULL| EXAMPLE_SPM_TABLE |     1 |     8 |     3   (0)|
00:00:01 |
-------------------------------------------------------------------------------
---------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("ID"=100)
```

ALTER_SQL_PLAN_BASELINE Function

This function changes an attribute of a single plan or all plans associated with a SQL statement using the attribute name/value format.

Syntax

```
DBMS_SPM.ALTER_SQL_PLAN_BASELINE (
   sql_handle        IN VARCHAR2 := NULL,
   plan_name         IN VARCHAR2 := NULL,
   attribute_name    IN VARCHAR2,
   attribute_value   IN VARCHAR2)
 RETURN PLS_INTEGER;
```

Parameters

Table 186-5 ALTER_SQL_PLAN_BASELINE Function Parameters

Parameter	Description
sql_handle	SQL statement handle. It identifies plans associated with a SQL statement for an attribute change. If NULL then plan_name must be specified.
plan_name	Name of a specific plan. Default NULL means set the attribute for all plans associated with a SQL statement identified by sql_handle. If NULL then sql_handle must be specified.
attribute_name	Name of the plan attribute to set (see table below).
attribute_value	Value of the plan attribute to use (see table below)

Table 186-6 Names & Values for ALTER_SQL_PLAN_BASELINE Function Parameters

Name	Description	Possible Values
enabled	'YES' means the plan is available for use by the optimizer. It may or may not be used, depending on accepted status.	'YES' or 'NO'
fixed	'YES' means the SQL plan baseline is not evolved over time. A fixed plan takes precedence over a non-fixed plan.	'YES' or 'NO'
autopurge	'YES' means the plan is purged if it is not used for a time period. 'NO' means it is never purged.	'YES' or 'NO'
plan_name	Name of the plan	String of up to 30 characters
description	Plan description.	String of up to 500 bytes

Return Values

The number of plans altered.

Usage Notes

When a single plan is specified, one of the various statuses, or plan name, or description can be altered. When all plans for a SQL statement are specified, one of various statuses, or description can be altered. This function can be called numerous times, each time setting a different plan attribute of same plan(s) or different plan(s).

CANCEL_EVOLVE_TASK Procedure

The procedure cancels a currently executing evolve task. All intermediate results are removed from the task.

Syntax

```
DBMS_SPM.CANCEL_EVOLVE_TASK  (
   task_name        IN  VARCHAR2);
```

Parameters

Table 186-7 CANCEL_EVOLVE_TASK Procedure Parameters

Parameter	Description
task_name	Identifier of task to cancel

CONFIGURE Procedure

This procedure sets configuration options for the SQL management base and for the maintenance of SQL plan baselines. You can call this function multiple times, setting a different configuration option each time.

Syntax

```
DBMS_SPM.CONFIGURE (
   parameter_name    IN VARCHAR2,
   parameter_value   IN VARCHAR2 := NULL,
   allow             IN BOOLEAN := TRUE);
```

Parameters

Table 186-8 CONFIGURE Procedure Parameters

Parameter	Description
parameter_name	Name of parameter to set (see table below).
parameter_value	Value of parameter to use (see table below). The maximum length of parameter_value is 1000 characters.
allow	Whether to include (true) or exclude (false) matching SQL statements and plans for theauto_capture_* parameters. If null, then the procedure ignores the specified parameter.

Table 186-9 Names and Values for CONFIGURE Procedure Parameters

Parameter Name	Description	Possible Values
auto_capture_action	Action to include (=) or exclude (<>) for SQL plan management automatic capture, depending on whether allow is TRUE or FALSE. A null value removes the filter for parameter_name entirely.	Action name, for example, R%

Table 186-9 (Cont.) Names and Values for CONFIGURE Procedure Parameters

Parameter Name	Description	Possible Values
auto_capture_module	Module to include (=) or exclude (<>) for SQL plan management auto capture, depending on whether allow is TRUE or FALSE. A null value removes the filter for parameter_name entirely. The database only uses this filter when OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is TRUE.	Module name, for example, LOGGER
auto_capture_parsing_schema_name	Parsing schema to include (=) or exclude (<>) for SQL plan management auto capture, depending on whether allow is TRUE or FALSE. A null value removes the filter for parameter_name entirely. The database only uses this filter when OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is TRUE.	Schema name, for example, HR
auto_capture_sql_text	Search pattern to apply to SQL text of LIKE or NOT LIKE, depending on whether allow is TRUE or FALSE. A null value removes the filter for parameter_name entirely. The database only uses this filter when OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is TRUE.	Text of a SQL statement, for example, SELECT a%
auto_spm_evolve_task	Setting to enable or disable the high-frequency SPM Evolve Advisor task. OFF -- disables background-verified SQL plan management and real-time SQL plan management. ON -- enables background-verified SQL plan management. AUTO -- enables real-time SQL plan management. **Note:** See About Automatic SQL Plan Management in the *Oracle Database SQL Tuning Guide*, which discusses the high frequency SPM Evolve task and explains the difference between real-time and background-verified Automatic SQL Plan Management.	ON, OFF, AUTO
plan_retention_weeks	Number of weeks to retain unused plans before the database purges them. A null value resets to the default value of 53 weeks, or 1 year plus 1 week. (This retains plans for annually executing queries.) The value of allow is ignored.	5–523 (default is 53)
space_budget_percent	Maximum percent of SYSAUX space that can be used for SQL management base. The database issues alerts when this amount is exceeded. A null value resets the percentage to the default value of 10%. The value of allow is ignored.	1–50 (default is 10)

Exceptions

Table 186-10 CONFIGURE Exceptions

Error Number	Description
ORA-38133	Invalid parameter name
ORA-38134	Invalid parameter value
ORA-38150	Not enough space for new filter
ORA-38151	Module name too long
ORA-38152	Action name too long
ORA-38304	Missing or invalid user name

Usage Notes

- When `parameter_name` is `auto_capture_sql_text`, the `parameter_value` is an automatic search filter. The filter uses the search pattern of `LIKE` *`parameter_name`* when `allow=>true`. The filter uses the pattern `NOT LIKE` *`parameter_name`* when `allow=>false`.

  For all other non-null `parameter_name` values, the search pattern depends on the `allow` setting. The parameter uses an equal sign (=) when `allow=>true`. The parameter uses a not-equal sign (<>) when `allow=>false`.
- You can configure multiple automatic capture parameters of different types. You cannot specify multiple values for the same parameter. Instead, the values specified for a particular parameter are combined. For example, specifying `auto_capture_sql_text` to be '`%TABLE1%`', `TRUE`, and '`%TABLE2%`', `FALSE` will result in matching SQL text `LIKE '%TABLE1%'` and `NOT LIKE '%TABLE2%'`. The database uses these configuration settings only when the initialization parameter `OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES` is set to `TRUE`.
- A null value for `parameter_value` removes the filter for `parameter_name` entirely. By using `paramter_value=>'%'` in combination with `allow=FALSE`, you can filter out all values for a parameter, and then create a separate filter to include only specified values. The `DBA_SQL_MANAGEMENT_CONFIG` view shows the current filters.
- The default space budget for SQL management base is no more than ten percent of the size of `SYSAUX` tablespace. The space budget can be set to a maximum of 50%. The default unused plan retention period is one year and one week, which means a plan will be automatically purged if it has not been used for more than a year. The retention period can be set to a maximum of 523 weeks (i.e. a little over 10 years).
- When the space occupied by SQL management base exceeds the defined space budget limit, a weekly database alert is generated.

Examples

The following example creates a filter for SQL text that is like `SELECT a%`:

```
EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_SQL_TEXT', 'select a%', 'TRUE');
```

The following example filters out the `HR` parsing schema:

```
EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_PARSING_SCHEMA_NAME', 'HR', 'FALSE');
```

The following example removes any existing filters for SQL text:

```
EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_SQL_TEXT', NULL, NULL);
```

The following example removes any `LIKE` or `NOT LIKE` filters for the SQL text `select a%`:

```
EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_SQL_TEXT', 'select a%', NULL);
```

The following example creates a filter with the predicate `(action LIKE 'R%') OR (action LIKE '%E_')`:

```
EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_ACTION', 'R%', 'TRUE');
EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_ACTION', '%E_', 'TRUE');
```

The following example creates a filter with the predicate `NOT(module LIKE 'LOGGER') AND NOT(module LIKE 'UTIL__')`:

```
EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_MODULE', 'LOGGER', 'FALSE');
EXEC DBMS_SPM.CONFIGURE('AUTO_CAPTURE_MODULE', 'UTIL__', 'FALSE');
```

CREATE_EVOLVE_TASK Function

The function has two overloads, both of which create an advisor task and sets its parameters. This version which takes a SQL handle creates an evolve task in order to evolve one or more plans for a given SQL statement.

Syntax

```
DBMS_SPM.CREATE_EVOLVE_TASK (
   sql_handle     IN  VARCHAR2  := NULL,
   plan_name      IN  VARCHAR2  := NULL,
   time_limit     IN  NUMBER    := DBMS_SPM.AUTO_LIMIT,
   task_name      IN  VARCHAR2  := NULL,
   description    IN  VARCHAR2  := NULL)
  RETURN VARCHAR2;

DBMS_SPM.CREATE_EVOLVE_TASK (
   plan_list      IN  DBMS_SPM.NAME_LIST,
   time_limit     IN  NUMBER    := DBMS_SPM.AUTO_LIMIT,
   task_name      IN  VARCHAR2  := NULL,
   description    IN  VARCHAR2  := NULL)
  RETURN VARCHAR2;
```

Parameters

Table 186-11 CREATE_EVOLVE_TASK Function Parameters

Parameter	Description
`sql_handle`	Handle of a SQL statement. The default `NULL` considers all SQL statements with non-accepted plans.
`plan_list`	List of plan names. The plans may belong to different SQL statements.

Table 186-11 (Cont.) CREATE_EVOLVE_TASK Function Parameters

Parameter	Description
plan_name	Plan identifier. The default NULL considers all non-accepted plans of the specified SQL handle or all SQL statements if the SQL handle is NULL.
time_limit	Time limit in number of minutes. The time limit is global and it is used in the following manner. The time limit for first non-accepted plan is equal to the input value. The time limit for the second non-accepted plan is equal to (input value - time spent in first plan verification) and so on. The default DBMS_SPM.AUTO_LIMIT means let the system choose an appropriate time limit based on the number of plan verifications required to be done. The value DBMS_SPM.NO_LIMIT means no time limit.
task_name	Evolve task name
description	Description of the task (maximum 256 characters)

Return Values

SQL evolve task unique name

CREATE_STGTAB_BASELINE Procedure

This procedure creates a staging table used for transporting SQL plan baselines from one system to another.

Syntax

```
DBMS_SPM.CREATE_STGTAB_BASELINE (
   table_name        IN VARCHAR2,
   table_owner       IN VARCHAR2 := NULL,
   tablespace_name   IN VARCHAR2 := NULL);
```

Parameters

Table 186-12 CREATE_STGTAB_BASELINE Procedure Parameters

Parameter	Description
table_name	Name of staging table to create for the purpose of packing and unpacking SQL plan baselines
table_owner	Name of owner of the staging table. The default, NULL, means that the current schema is the table owner.
tablespace_name	Name of the tablespace. The default, NULL, results in creating the staging table in the default tablespace.

Usage Notes

The creation of staging table is the first step. To migrate SQL plan baselines from one system to another, the user or DBA has to perform a series of steps as follows:

1. Create a staging table in the source system

2. Select SQL plan baselines in the source system and pack them into the staging table
3. Copy the staging table from the source system to the target system. For example, you can use Oracle Data Pump to export and import the staging table.
4. Select SQL plan baselines from the staging table and unpack them into the target system

DROP_EVOLVE_TASK Procedure

The procedure drops an evolved task.

Syntax

```
DBMS_SPM.DROP_EVOLVE_TASK  (
   task_name         IN  VARCHAR2);
```

Parameters

Table 186-13 DROP_EVOLVE_TASK Procedure Parameters

Parameter	Description
task_name	Identifier of the task that you want to drop

DROP_MIGRATED_STORED_OUTLINE Function

This function drops all stored outlines that have been migrated to SQL plan baselines.

Syntax

```
DBMS_SPM.DROP_MIGRATED_STORED_OUTLINE ()
RETURN PLS_INTEGER;
```

Requires the following privileges:

- Administer SQL Management Object
- DROP ANY OUTLINE
- select on dba_outlines

Return Values

The number of outlines dropped.

DROP_SQL_PLAN_BASELINE Function

This function drops a single plan, or all plans associated with a SQL statement.

Syntax

```
DBMS_SPM.DROP_SQL_PLAN_BASELINE (
   sql_handle     IN VARCHAR2 := NULL,
   plan_name      IN VARCHAR2 := NULL)
RETURN PLS_INTEGER;
```

Parameters

Table 186-14 DROP_SQL_PLAN_BASELINE Function Parameters

Parameter	Description
sql_handle	SQL statement handle. It identifies plans associated with a SQL statement that are to be dropped. If NULL then plan_name must be specified.
plan_name	Plan name. It identifies a specific plan. Default NULL means to drop all plans associated with the SQL statement identified by sql_handle.

Return Values

The number of plans dropped

EVOLVE_SQL_PLAN_BASELINE Function

This function evolves SQL plan baselines associated with one or more SQL statements. A SQL plan baseline is evolved when one or more of its non-accepted plans is changed to an accepted plan or plans.

If interrogated by the user (parameter verify = 'YES'), the execution performance of each non-accepted plan is compared against the performance of a plan chosen from the associated SQL plan baseline. If the non-accepted plan performance is found to be better than SQL plan baseline performance, the non-accepted plan is changed to an accepted plan provided such action is permitted by the user (parameter commit = 'YES').

The second form of the function employs a plan list format.

Syntax

```
DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE (
   sql_handle    IN VARCHAR2 := NULL,
   plan_name     IN VARCHAR2 := NULL,
   time_limit    IN INTEGER  := DBMS_SPM.AUTO_LIMIT,
   verify        IN VARCHAR2 := 'YES',
   commit        IN VARCHAR2 := 'YES')
  RETURN CLOB;

DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE (
   plan_list     IN DBMS_SPM.NAME_LIST,
   time_limit    IN INTEGER  := DBMS_SPM.AUTO_LIMIT,
   verify        IN VARCHAR2 := 'YES',
   commit        IN VARCHAR2 := 'YES')
  RETURN CLOB;
```

Parameters

Table 186-15 EVOLVE_SQL_PLAN_BASELINE Function Parameters

Parameter	Description
sql_handle	SQL statement identifier. Unless plan_name is specified, NULL means to consider all statements with non-accepted plans in their SQL plan baselines.

Table 186-15 (Cont.) EVOLVE_SQL_PLAN_BASELINE Function Parameters

Parameter	Description
plan_name	Plan identifier. Default NULL means to consider all non- accepted plans in the SQL plan baseline of either the identified SQL statement or all SQL statements if sql_handle is NULL.
plan_list	A list of plan names. Each plan in the list can belong to same or different SQL statement.
time_limit	Time limit in number of minutes. This applies only if verify = 'YES'. The time limit is global and it is used as follows: The time limit for first non-accepted plan verification is set equal to the input value; the time limit for second non-accepted plan verification is set equal to (input value - time spent in first plan verification); and so on. • DBMS_SPM.AUTO_LIMIT (Default) lets the system choose an appropriate time limit based on the number of plan verifications required to be done. • DBMS_SPM.NO_LIMIT means there is no time limit. • A positive integer value represents a user specified time limit.
verify	Specifies whether to execute the plans and compare the performance before changing non-accepted plans into accepted plans. A performance verification involves executing a non-accepted plan and a plan chosen from corresponding SQL plan baseline and comparing their performance statistics. If non-accepted plan shows performance improvement, it is changed to an accepted plan. • 'YES' (Default) - verifies that a non-accepted plan gives better performance before changing it to an accepted plan • 'NO' - directs not to execute plans but only to change non-accepted plans into accepted plans
commit	Specifies whether to update the ACCEPTED status of non-accepted plans from 'NO' to 'YES'. • 'YES' (Default) - perform updates of qualifying non-accepted plans and generate a report that shows the updates and the result of performance verification when verify = 'YES'. • 'NO' - generate a report without any updates. Note that commit = 'NO' together with verify = 'NO' represents a no-op.

Return Values

A CLOB containing a formatted text report showing non-accepted plans in sequence, each with a possible change of its ACCEPTED status, and if verify = 'YES' the result of their performance verification.

Usage Notes

Invoking this subprogram requires the ADMINISTER SQL MANAGEMENT OBJECT privilege.

EXECUTE_EVOLVE_TASK Function

The function executes a previously created evolve task.

Syntax

```
DBMS_SPM.EXECUTE_EVOLVE_TASK  (
   task_name        IN  VARCHAR2,
```

```
   execution_name    IN  VARCHAR2  := NULL,
   execution_desc    IN  VARCHAR2  := NULL);
 RETURN VARCHAR2;
```

Parameters

Table 186-16 EXECUTE_EVOLVE_TASK Function Parameters

Parameter	Description
task_name	Evolve task name
execution_name	Name to qualify and identify an execution. If not specified, it is generated by the advisor and returned by the function.
execution_desc	Description of the execution (maximum 256 characters)

Return Values

Name of the new execution

INTERRUPT_EVOLVE_TASK Procedure

The procedure interrupts a currently executing evolve task. The task ends its operations as at a normal exit and the user can access the intermediate results. The task can be resumed later.

Syntax

```
DBMS_SPM.INTERRUPT_EVOLVE_TASK  (
   task_name        IN  VARCHAR2);
```

Parameters

Table 186-17 INTERRUPT_EVOLVE_TASK Procedure Parameters

Parameter	Description
task_name	Identifier of task to interrupt

LOAD_PLANS_FROM_AWR Function

This function loads the SQL Management Base (SMB) with SQL plan baselines for a set of SQL statements using the plans from the AWR, and returns the number of plans loaded.

Syntax

```
DBMS_SPM.LOAD_PLANS_FROM_AWR
   begin_snap      IN  NUMBER,
   end_snap        IN  NUMBER,
   basic_filter    IN  VARCHAR2 := NULL,
   fixed           IN  VARCHAR2 := 'NO',
   enabled         IN  VARCHAR2 := 'YES',
   commit_rows     IN  NUMBER := 1000)
 RETURN PLS_INTEGER;
```

Parameters

Table 186-18 LOAD_PLANS_FROM_AWR Function Parameters

Parameter	Description
begin_snap	Begin snapshot
end_snap	End snapshot
basic_filter	SQL predicate to filter the SQL from AWR. NULL means all plans in AWR are selected. Specifies the SQL predicate that filters the SQL from the shared SQL area defined on attributes of the SQLSET_ROW.
fixed	Default 'NO' means the loaded plans will not change the current 'fixed' property of the SQL plan baseline into which they are loaded.
enabled	Default 'YES' means the loaded plans will be considered by the optimizer
commit_rows	Number of SQL plans to load before doing a periodic commit.
dbid	The DBID that is used for imported or PDB-level AWR data.

Usage Notes

Requires the Administer SQL Management Object privilege

> **See Also:**
>
> For information on the SQLSET_ROW objects, see SQLSET_ROW Object Type.

LOAD_PLANS_FROM_CURSOR_CACHE Functions

This function loads one or more plans present in the cursor cache for a SQL statement, or a set of SQL statements. It has four overloads: using SQL statement text, using SQL handle, using SQL ID, or using attribute_name and attribute_value pair.

Syntax

```
DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (
   sql_id            IN  VARCHAR2,
   plan_hash_value   IN  NUMBER   := NULL,
   sql_text          IN  CLOB,
   fixed             IN  VARCHAR2 := 'NO',
   enabled           IN  VARCHAR2 := 'YES')
 RETURN PLS_INTEGER;

DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (
   sql_id            IN  VARCHAR2,
   plan_hash_value   IN  NUMBER   := NULL,
   sql_handle        IN  VARCHAR2,
   fixed             IN  VARCHAR2 := 'NO',
```

```
   enabled           IN  VARCHAR2 := 'YES')
 RETURN PLS_INTEGER;

DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (
   sql_id            IN  VARCHAR2,
   plan_hash_value   IN  NUMBER   := NULL,
   fixed             IN  VARCHAR2 := 'NO',
   enabled           IN  VARCHAR2 := 'YES')
 RETURN PLS_INTEGER;

DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE (
   attribute_name   IN VARCHAR2,
   attribute_value  IN VARCHAR2,
   fixed            IN VARCHAR2 := 'NO',
   enabled          IN VARCHAR2 := 'YES')
  RETURN PLS_INTEGER;
```

Parameters

Table 186-19 LOAD_PLANS_FROM_CURSOR_CACHE Function Parameters

Parameter	Description
sql_id	SQL statement identifier. Identifies a SQL statement in the cursor cache. Note: In the third overload the text of identified SQL statement is extracted from cursor cache and is used to identify the SQL plan baseline into which the plan(s) are loaded. If the SQL plan baseline doesn't exist it is created.
plan_hash_value	Plan identifier. Default NULL means capture all plans present in the cursor cache for the SQL statement identified by SQL_ID.
sql_text	SQL text to use in identifying the SQL plan baseline into which the plans are loaded. If the SQL plan baseline does not exist, it is created. The use of text is crucial when the user tunes a SQL statement by adding hints to its text and then wants to load the resulting plan(s) into the SQL plan baseline of the original SQL statement.
sql_handle	SQL handle to use in identifying the SQL plan baseline into which the plans are loaded. The sql_handle must denote an existing SQL plan baseline. The use of handle is crucial when the user tunes a SQL statement by adding hints to its text and then wants to load the resulting plan(s) into the SQL plan baseline of the original SQL statement.
fixed	Default 'NO' means the loaded plans are used as non-fixed plans. Value 'YES' means the loaded plans are used as fixed plans and the SQL plan baseline will not be evolved over time.
attribute_name	One of possible attribute names: • SQL_TEXT" • 'PARSING_SCHEMA_NAME' • 'MODULE' • 'ACTION'

Table 186-19 (Cont.) LOAD_PLANS_FROM_CURSOR_CACHE Function Parameters

Parameter	Description
attribute_value	Attribute value is used as a search pattern of LIKE predicate if attribute name is 'SQL_TEXT'. Otherwise, it is used as an equality search value. (for example, for specifying attribute_name => 'SQL_TEXT', and attribute_value => '% HR-123 %' means applying SQL_TEXT LIKE '% HR-123 %' as a selection filter. Similarly, specifying attribute_name => 'MODULE', and attribute_value => 'HR' means applying 'MODULE = 'HR' as a plan selection filter). The attribute value is upper-cased except when it is enclosed in double quotes or attribute name is 'SQL_TEXT'.
enabled	Default 'YES' means the loaded plans are enabled for use by the optimizer

Return Values

Number of plans loaded

Usage Notes

Invoking this subprogram requires the ADMINISTER SQL MANAGEMENT OBJECT privilege.

LOAD_PLANS_FROM_SQLSET Function

This function loads plans stored in a SQL tuning set (STS) into SQL plan baselines. The plans loaded from STS are not verified for performance but added as accepted plans to existing or new SQL plan baselines. This function can be used to seed SQL management base with new SQL plan baselines.

Syntax

```
DBMS_SPM.LOAD_PLANS_FROM_SQLSET (
   sqlset_name      IN  VARCHAR2,
   sqlset_owner     IN  VARCHAR2 := NULL,
   basic_filter     IN  VARCHAR2 := NULL,
   fixed            IN  VARCHAR2 := 'NO',
   enabled          IN  VARCHAR2 := 'YES'
   commit_rows      IN  NUMBER   := 1000)
RETURN PLS_INTEGER;
```

Parameters

Table 186-20 LOAD_PLANS_FROM_SQLSET Function Parameters

Parameter	Description
sqlset_name	Name of the STS from where the plans are loaded into SQL plan baselines
sqlset_owner	Owner of STS. NULL means current schema is the owner.

Table 186-20 (Cont.) LOAD_PLANS_FROM_SQLSET Function Parameters

Parameter	Description
basic_filter	A filter applied to the STS to select only qualifying plans to be loaded. The filter can take the form of any WHERE clause predicate that can specified against the view DBA_SQLSET_STATEMENTS. For example basic_filter => 'sql_text like ''select /*LOAD_STS*/ %''' or basic_filter => 'sql_id=''b62q7nc33gzwx'''.
fixed	Default 'NO' means the loaded plans are used as non-fixed plans. Value 'YES' means the loaded plans are used as fixed plans and the SQL plan baseline will not be evolved over time.
enabled	Default 'YES' means the loaded plans are enabled for use by the optimizer
commit_rows	Number of SQL plans to load before doing a periodic commit. This helps to shorten the undo log.

Return Values

The number of plans loaded

Usage Notes

- To load plans from a remote system, first load the plans into an STS on the remote system, export/import the STS from remote to local system, and then use this function.
- To load plans from Automatic Workload Repository (AWR), first load the plans stored in AWR snapshots into an STS, and then use this procedure.
- The user can also capture plans resident in the cursor cache for one or more SQL statements into an STS, and then use this procedure.

MIGRATE_STORED_OUTLINE Functions

This function migrates stored outlines for one or more SQL statements to plan baselines in the SQL management base (SMB). Users can specify which stored outline(s) to be migrated based on outline name, SQL text, or outline category, or migrate all stored outlines in the system to SQL plan baselines.

This second overload of the function migrates stored outlines for one or more SQL statements to plan baselines in the SQL management base (SMB) given one or more outline names.

Syntax

```
DBMS_SPM.MIGRATE_STORED_OUTLINE (
   attribute_name     IN  VARCHAR2,
   attribute_value    IN  CLOB,
   fixed              IN  VARCHAR2 := 'NO')
 RETURN CLOB;

DBMS_SPM.MIGRATE_STORED_OUTLINE (
   outln_list         IN  DBMS_SPM.NAME_LIST,
   fixed              IN  VARCHAR2 := 'NO')
 RETURN CLOB;
```

Parameters

Table 186-21 MIGRATE_STORED_OUTLINE Function Parameters

Parameter	Description
attribute_name	Specifies the type of parameter used in attribute_value to identify the migrated stored outlines. It is case insensitive. Possible values: • outline_name • sql_text • category • all
attribute_value	Based on attribute_name, this can be: • Name of stored outline to be migrated • SQL text of stored outlines to be migrated • Category of stored outlines to be migrated • NULL if attribute_name is all
fixed	NO (default) or YES. Specifies the "fixed" status of the plans generated during migration. By default, plans are generated as "non-fixed" plans.
outln_list	List of outline names to be migrated

Return Values

A CLOB containing a formatted report to describe the statistics during the migration, including:

- Number of stored outlines successfully migrated
- Number of stored outlines (and also the corresponding outline names) failed to be migrated and the reasons for the failure

Usage Note

- When the user specifies an outline name, the function migrates stored outlines to plan baseline based on given outline name, which uniquely identifies a single stored outline to be migrated.
- When the user specifies SQL text, the function migrates all stored outlines created for a given SQL statement. A single SQL statement can have multiple stored outlines created for it under different category names. One plan baseline plan is created for each stored outline. The new plan baselines have category names set to DEFAULT. The module name of a plan baseline is set to be the same as the stored outline.
- When the user specifies a category name, the function migrates all stored outlines with the given category name. Only one stored outline exists per category per SQL statement. One plan baseline is created for each stored outline.
- When user specifies to migrate all, the function migrates all stored outlines in the system to plan baselines. One plan baseline is created for each stored outline.

PACK_STGTAB_BASELINE Function

This function packs (exports) SQL plan baselines from SQL management base into a staging table.

Syntax

```
DBMS_SPM.PACK_STGTAB_BASELINE (
   table_name       IN VARCHAR2,
   table_owner      IN VARCHAR2 := NULL,
   sql_handle       IN VARCHAR2 := NULL,
   plan_name        IN VARCHAR2 := NULL,
   sql_text         IN CLOB     := NULL,
   creator          IN VARCHAR2 := NULL,   origin           IN VARCHAR2 := NULL,
   enabled          IN VARCHAR2 := NULL,
   accepted         IN VARCHAR2 := NULL,
   fixed            IN VARCHAR2 := NULL,
   module           IN VARCHAR2 := NULL,
   action           IN VARCHAR2 := NULL)
RETURN NUMBER;
```

Parameters

Table 186-22 PACK_STGTAB_BASELINE Function Parameters

Parameter	Description
table_name	Name of staging table into which SQL plan baselines are packed (case insensitive unless double quoted)
table_owner	Name of staging table owner.Default NULL means current schema is the table owner
sql_handle	SQL handle (case sensitive)
plan_name	Plan name (case sensitive, % wildcards accepted)
sql_text	SQL text string (case sensitive, % wildcards accepted)
creator	Creator of SQL plan baseline (case insensitive unless double quoted)
origin	Origin of SQL plan baseline, should be 'MANUAL-LOAD', 'AUTO-CAPTURE', 'MANUAL_SQLTUNE' or 'AUTO-SQLTUNE' (case insensitive)
enabled	Must be 'YES' or 'NO' (case insensitive)
accepted	Must be 'YES' or 'NO' (case insensitive)
fixed	Must be 'YES' or 'NO' (case insensitive)
module	Module (case sensitive)
action	Action (case sensitive)

Return Values

Number of SQL plan baselines packed

RESET_EVOLVE_TASK Procedure

This procedure resets an evolve task to its initial state.

All intermediate results will be removed from the task. Call this procedure on a task that is not currently executing.

Syntax

```
DBMS_SPM.RESET_EVOLVE_TASK  (
   task_name        IN  VARCHAR2);
```

Parameters

Table 186-23 RESET_EVOLVE_TASK Procedure Parameters

Parameter	Description
task_name	Identifier of task to reset

RESUME_EVOLVE_TASK Procedure

The procedure resumes a previously interrupted task.

Syntax

```
DBMS_SPM.RESUME_EVOLVE_TASK  (
   task_name        IN  VARCHAR2);
```

Parameters

Table 186-24 RESUME_EVOLVE_TASK Procedure Parameters

Parameter	Description
task_name	Identifier of task to resume

REPORT_AUTO_EVOLVE_TASK Function

The procedure displays the results of an execution of an automatic evolve task.

Syntax

```
DBMS_SPM.REPORT_AUTO_EVOLVE_TASK  (
   type            IN  VARCHAR2  := TYPE_TEXT,
   level           IN  VARCHAR2  := LEVEL_TYPICAL,
   section         IN  VARCHAR2  := SECTION_ALL,
   object_id       IN  NUMBER    := NULL,
   execution_name  IN  VARCHAR2  := NULL)
 RETURN CLOB;
```

Parameters

Table 186-25 REPORT_AUTO_EVOLVE_TASK Function Parameters

Parameter	Description
type	Type of the report. Possible values are TEXT, HTML, XML
level	Format of the report. Possible values are BASIC, TYPICAL, ALL.
section	Particular section in the report. Possible values are: SUMMARY, FINDINGS, PLANS, INFORMATION, ERRORS, ALL.
object_id	Identifier of the advisor framework object that represents a single plan. If NULL, the report is generated for all objects.
execution_name	Name to qualify and identify an execution. If NULL, the report is generated for the last task execution.

Return Values

The report

REPORT_EVOLVE_TASK Function

The procedure displays the results of an evolved task.

Syntax

```
DBMS_SPM.REPORT_EVOLVE_TASK  (
   task_name       IN  VARCHAR2,
   type            IN  VARCHAR2  := TYPE_TEXT,
   level           IN  VARCHAR2  := LEVEL_TYPICAL,
   section         IN  VARCHAR2  := SECTION_ALL,
   object_id       IN  NUMBER    := NULL,
   task_owner      IN  VARCHAR2  := NULL,
   execution_name  IN  VARCHAR2  := NULL)
 RETURN CLOB;
```

Parameters

Table 186-26 REPORT_EVOLVE_TASK Function Parameters

Parameter	Description
task_name	Identifier of task to report
type	Type of the report. Possible values are TEXT, HTML, XML
level	Format of the report. Possible values are BASIC, TYPICAL, ALL.
section	Particular section in the report. Possible values are: SUMMARY, FINDINGS, PLANS, INFORMATION, ERRORS, ALL.
object_id	Identifier of the advisor framework object that represents a single plan. If NULL, the report is generated for all objects.
task_owner	Owner of the evolve task. Defaults to the current schema owner.
execution_name	Name to qualify and identify an execution. If NULL, the report is generated for the last task execution.

Return Values

The report

SET_EVOLVE_TASK_PARAMETER Procedure

The procedure sets a parameter of an evolve task, either a `VARCHAR2` or a `NUMBER`.

Syntax

```
DBMS_SPM.SET_EVOLVE_TASK_PARAMETER  (
   task_name     IN  VARCHAR2,
   parameter     IN  VARCHAR2,
   value         IN  NUMBER);

DBMS_SPM.SET_EVOLVE_TASK_PARAMETER  (
   task_name     IN  VARCHAR2  := NULL,
   parameter     IN  VARCHAR2,
   value         IN  VARCHAR2);
```

Parameters

Table 186-27 SET_EVOLVE_TASK_PARAMETER Procedure Parameters

Parameter	Description
`task_name`	Evolve task name
`parameter`	Name of the parameter to set (see following table)
`value`	Value of the parameter (see following table)

The following table describes parameters for the `SET_EVOLVE_TASK_PARAMETER` procedure.

Table 186-28 DBMS_SPM.SET_EVOLVE_TASK_PARAMETER Parameters

Parameter	Description	Default
`alternate_plan_source`	Determines which sources to search for additional plans: • `AUTO` (the database selects the source automatically) • `AUTOMATIC_WORKLOAD_REPOSITORY` • `CURSOR_CACHE` • `SQL_TUNING_SET` You can combine multiple values with the plus sign (+).	The default depends on whether the SPM Evolve Advisor task is automated or manual: • If automated, the default is `AUTO`. • If manual, the default is `CURSOR_CACHE+AUTOMATIC_WORKLOAD_REPOSITORY`.

Table 186-28 (Cont.) DBMS_SPM.SET_EVOLVE_TASK_PARAMETER Parameters

Parameter	Description	Default
`alternate_plan_baseline`	Determines which alternative plans should be loaded: • `AUTO` lets Autonomous Database choose whether to load plans for statements with or without baselines. • `EXISTING` loads alternate plans with for statements with existing baselines. • `NEW` loads alternative plans for statements without a baseline, in which case a new baseline is created. You can combine multiple values with the plus sign (+), as in `EXISTING+NEW`.	`EXISTING`
`alternate_plan_limit`	Specifies the maximum number of plans to load in total (that is, not the limit for each SQL statement).	The default depends on whether the SPM Evolve Advisor task is automated or manual: • If automated, the default is `UNLIMITED`. • If manual, the default is `10`.
`accept_plans`	Specifies whether to accept recommended plans automatically. When `ACCEPT_PLANS` is `true`, SQL plan management automatically accepts all plans recommended by the task. When `ACCEPT_PLANS` is `false`, the task verifies the plans and generates a report of its findings, but does not evolve the plans automatically. You can use a report to identify new SQL plan baselines and accept them manually.	`true` (regardless of whether the advisor is run automatically or manually)
`time_limit`	Global time limit in seconds. This is the total time allowed for the task.	The default depends on whether the SPM Evolve Advisor task is automated or manual: • If automated, the default is `3600`. • If manual, the default is `2147483646`.

See Also:

Oracle Database Licensing Information User Manual for details on which features are supported for different editions and services

UNPACK_STGTAB_BASELINE Function

This function unpacks (imports) SQL plan baselines from a staging table into SQL management base.

Syntax

```
DBMS_SPM.UNPACK_STGTAB_BASELINE (
   table_name        IN VARCHAR2,
   table_owner       IN VARCHAR2 := NULL,
   sql_handle        IN VARCHAR2 := NULL,
   plan_name         IN VARCHAR2 := NULL,
   sql_text          IN CLOB     := NULL,
   creator           IN VARCHAR2 := NULL,   origin           IN VARCHAR2 := NULL,
   enabled           IN VARCHAR2 := NULL,
   accepted          IN VARCHAR2 := NULL,
   fixed             IN VARCHAR2 := NULL,
   module            IN VARCHAR2 := NULL,
   action            IN VARCHAR2 := NULL)
RETURN NUMBER;
```

Parameters

Table 186-29 UNPACK_STGTAB_BASELINE Function Parameters

Parameter	Description
table_name	Name of staging table from which SQL plan baselines are unpacked (case insensitive unless double quoted)
table_owner	Name of staging table owner.Default NULL means current schema is the table owner
sql_handle	SQL handle (case sensitive)
plan_name	Plan name (case sensitive,% wildcards accepted)
sql_text	SQL text string (case sensitive, % wildcards accepted)
creator	Creator of SQL plan baseline (case insensitive unless double quoted)
origin	Origin of SQL plan baseline, should be 'MANUAL-LOAD', 'AUTO-CAPTURE','MANUAL_SQLTUNE' or 'AUTO-SQLTUNE' (case insensitive)
enabled	Must be 'YES' or 'NO' (case insensitive)
accepted	Must be 'YES' or 'NO' (case insensitive)
fixed	Must be 'YES' or 'NO' (case insensitive)
module	Module (case sensitive)
action	Action (case sensitive)

Return Values

Number of plans unpacked