178

DBMS_SESSION

This package provides access to SQL `ALTER SESSION` and `SET ROLE` statements, and other session information, from PL/SQL. You can use `DBMS_SESSION` to set preferences and security levels.

This chapter contains the following topics:

DBMS_SESSION Security Model

This package runs with the privileges of the calling user, rather than the package owner `SYS`.

DBMS_SESSION Operational Notes

You should not attempt to turn `close_cached_open_cursors` on or off.

DBMS_SESSION Data Structures

The `DBMS_SESSION` package defines `TABLE` types.

Table Types

- INTEGER_ARRAY Table Type
- LNAME_ARRAY Table Type

DBMS_SERVICE INTEGER_ARRAY Table Type

`INTEGER_ARRAY` is a table type of `BINARY_INTEGER`.

Syntax

```
TYPE integer_array IS TABLE OF BINARY_INTEGER INDEX BY BINARY_INTEGER;
```

DBMS_SERVICE LNAME_ARRAY Table Type

`LNAME_ARRAY` is a table type of `VARCHAR2`.

Syntax

```
TYPE lname_array IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```

Summary of DBMS_SESSION Subprograms

This table lists the DBMS_SESSION subprograms in alphabetical order and briefly describes them.

Table 178-1 ***DBMS_SESSION Package Subprograms***

Subprogram	Description
CLEAR_ALL_CONTEXT Procedure	Clears all context information.
CLEAR_ALL_LOCAL_CONTEXTS Procedure	Clears all session based application context values that are set in the current session.
CLEAR_CONTEXT Procedure	Clears the context.
CLEAR_IDENTIFIER Procedure	Clears the identifier.
CLOSE_DATABASE_LINK Procedure	Closes database link.
CURRENT_IS_ROLE_ENABLED Function	Determines if the named role is currently enabled.
FREE_UNUSED_USER_MEMORY Procedure	Lets you reclaim unused memory after performing operations requiring large amounts of memory.
GET_PACKAGE_MEMORY_UTILIZATION Procedure	Describes static package memory usage.
IS_ROLE_ENABLED Function	Determines if the named role is enabled for the session.
IS_SESSION_ALIVE Function	Determines if the specified session is active.
LIST_CONTEXT Procedures	Returns a list of active namespace and context for the current session.
MODIFY_PACKAGE_STATE Procedure	Used to perform various actions (as specified by the `action_flags` parameter) on the session state of all PL/SQL program units active in the session.
RESET_PACKAGE Procedure	De-instantiates all packages in the session.
SESSION_IS_ROLE_ENABLED Function	Determines if the named role is enabled at the session level.
SESSION_TRACE_DISABLE Procedure	Resets the session-level SQL trace for the session from which it was called.
SESSION _TRACE_ENABLE Procedure	Enables session-level SQL trace for the invoking session.
SET_CONTEXT Procedure	Sets or resets the value of a context attribute.
SET_EDITION_DEFERRED Procedure	Requests a switch to the specified edition.
SET_IDENTIFIER Procedure	Sets the identifier.
SET_NLS Procedure	Sets Globalization Support (NLS).
SET_ROLE Procedure	Sets role.
SET_SQL_TRACE Procedure	Turns tracing on or off.
SLEEP Procedure	Suspends the session for a specified period of time.
SWITCH_CURRENT_CONSUMER_GROUP Procedure	Facilitates changing the current resource consumer group of a user's current session.
UNIQUE_SESSION_ID Function	Returns an identifier that is unique for all sessions currently connected to this database.

CLEAR_ALL_CONTEXT Procedure

This procedure clears application context information in the specified namespace.

Syntax

```
DBMS_SESSION.CLEAR_ALL_CONTEXT
   namespace         VARCHAR2);
```

Parameters

Table 178-2 CLEAR_ALL_CONTEXT Procedure Parameters

Parameter	Description
`namespace`	The namespace where the application context information is to be cleared. Required.

Usage Notes

- This procedure must be invoked directly or indirectly by the trusted package.
- Any changes in context value are reflected immediately and subsequent calls to access the value through `SYS_CONTEXT` return the most recent value.

CLEAR_ALL_LOCAL_CONTEXTS Procedure

This procedure clears all session-based application context values that are set in the current session and are initialized locally, externally, and globally, but cannot be accessed globally.

Syntax

```
DBMS_SESSION.CLEAR_ALL_LOCAL_CONTEXTS;
```

Usage Notes

Any public user with the `CLEAR ALL LOCAL CONTEXTS` privilege can execute this procedure. If a user without the `CLEAR ALL LOCAL CONTEXTS` privilege executes this procedure, it fails with the following error:

```
ORA-01056: User $USER_NAME does not have CLEAR ALL LOCAL CONTEXTS privilege
```

CLEAR_CONTEXT Procedure

This procedure clears application context in the specified namespace.

Syntax

```
DBMS_SESSION.CLEAR_CONTEXT
   namespace         VARCHAR2,
   client_identifier VARCHAR2,
   attribute         VARCHAR2);
```

Parameters

Table 178-3 CLEAR_CONTEXT Procedure Parameters

Parameter	Description
`namespace`	Namespace in which the application context is to be cleared. Required. For a session-local context, `namespace` must be specified. If `namespace` is defined as `Session Local Context`, then `client_identifier` is optional since it is only associated with a globally accessed context. For a globally accessed context, `namespace` must be specified. `NULL` is a valid value for `client_identifier` because a session with no identifier set can see a context that looks like the (`namespace, attribute, value, username, null`) set using `SET_CONTEXT`.
`client_identifier`	Applies to a global context and is optional for other types of contexts; 64-byte maximum
`attribute`	Specific attribute in the namespace to be cleared. Optional. the default is `NULL`. If you specify `attribute` as `NULL`, then (`namespace, attribute, value`) for that namespace are cleared from the session. If `attribute` is not specified, then all context information that has the `namespace` and `client_identifier` arguments is cleared.

Usage Notes

- This procedure must be invoked directly or indirectly by the trusted package.
- Any changes in context value are reflected immediately and subsequent calls to access the value through `SYS_CONTEXT` return the most recent value.

CLEAR_IDENTIFIER Procedure

This procedure removes the `set_client_id` in the session.

Syntax

```
DBMS_SESSION.CLEAR_IDENTIFIER;
```

Usage Notes

This procedure is executable by public.

CLOSE_DATABASE_LINK Procedure

This procedure closes an open database link. It is equivalent to the following SQL statement: `ALTER SESSION CLOSE DATABASE LINK <name>`

Syntax

```
DBMS_SESSION.CLOSE_DATABASE_LINK (
   dblink VARCHAR2);
```

Parameters

Table 178-4 CLOSE_DATABASE_LINK Procedure Parameters

Parameter	Description
`dblink`	Name of the database link to close

CURRENT_IS_ROLE_ENABLED Function

This function determines if the named role is currently enabled.

Syntax

```
DBMS_SESSION.CURRENT_IS_ROLE_ENABLED (
   rolename     VARCHAR2)
  RETURN BOOLEAN;
```

Parameters

Table 178-5 CURRENT_IS_ROLE_ENABLED Function Parameters

Parameter	Description
`rolename`	Name of the role.

Return Values

- `TRUE`-if the role is enabled.
- `FALSE`-if the role is not enabled.

FREE_UNUSED_USER_MEMORY Procedure

This procedure reclaims unused memory after performing operations requiring large amounts of memory (more than 100K).

Examples of operations that use large amounts of memory include:

- Large sorting where entire `sort_area_size` is used and `sort_area_size` is hundreds of KB.
- Compiling large PL/SQL packages, procedures, or functions.
- Storing hundreds of KB of data within PL/SQL indexed tables.

You can monitor user memory by tracking the statistics "session UGA memory" and "session PGA memory" in the `v$sesstat` or `v$statname` fixed views. Monitoring these statistics also shows how much memory this procedure has freed.

Note:

This procedure should only be used in cases where memory is at a premium. It should be used infrequently and judiciously.

Syntax

```
DBMS_SESSION.FREE_UNUSED_USER_MEMORY;
```

Return Values

The behavior of this procedure depends upon the configuration of the server operating on behalf of the client:

- **Dedicated server**: This returns unused PGA memory and session memory to the operating system. Session memory is allocated from the PGA in this configuration.
- **Shared server**: This returns unused session memory to the `shared_pool`. Session memory is allocated from the `shared_pool` in this configuration.

Usage Notes

In order to free memory using this procedure, the memory must not be in use.

After an operation allocates memory, only the same type of operation can reuse the allocated memory. For example, after memory is allocated for sort, even if the sort is complete and the memory is no longer in use, only another sort can reuse the sort-allocated memory. For both sort and compilation, after the operation is complete, the memory is no longer in use, and the user can call this procedure to free the unused memory.

An indexed table implicitly allocates memory to store values assigned to the indexed table's elements. Thus, the more elements in an indexed table, the more memory the RDBMS allocates to the indexed table. As long as there are elements within the indexed table, the memory associated with an indexed table is in use.

The scope of indexed tables determines how long their memory is in use. Indexed tables declared globally are indexed tables declared in packages or package bodies. They allocate memory from session memory. For an indexed table declared globally, the memory remains in use for the lifetime of a user's login (lifetime of a user's session), and is freed after the user disconnects from ORACLE.

Indexed tables declared locally are indexed tables declared within functions, procedures, or anonymous blocks. These indexed tables allocate memory from PGA memory. For an indexed table declared locally, the memory remains in use for as long as the user is still running the procedure, function, or anonymous block in which the indexed table is declared.After the procedure, function, or anonymous block is finished running, the memory is then available for other locally declared indexed tables to use (in other words, the memory is no longer in use).

Assigning an uninitialized, "empty" indexed table to an existing index table is a method to explicitly re-initialize the indexed table and the memory associated with the indexed table. After this operation, the memory associated with the indexed table is no longer in use, making it available to be freed by calling this procedure. This method is particularly useful on indexed tables declared globally which can grow during the lifetime of a user's session, as long as the user no longer needs the contents of the indexed table.

The memory rules associated with an indexed table's scope still apply; this method and this procedure, however, allow users to intervene and to explicitly free the memory associated with an indexed table.

Examples

The following PL/SQL illustrates the method and the use of procedure `FREE_UNUSED_USER_MEMORY`.

```
CREATE PACKAGE foobar
   type number_idx_tbl is table of number indexed by binary_integer;

   store1_table  number_idx_tbl;     --  PL/SQL indexed table
   store2_table  number_idx_tbl;     --  PL/SQL indexed table
   store3_table  number_idx_tbl;     --  PL/SQL indexed table
   ...
END;            --  end of foobar

DECLARE
   ...
   empty_table   number_idx_tbl;     --  uninitialized ("empty") version
BEGIN
   FOR i in 1..1000000 loop
     store1_table(i) := i;           --  load data
   END LOOP;
   ...
   store1_table := empty_table;      --  "truncate" the indexed table
   ...
   -
   dbms_session.free_unused_user_memory;  -- give memory back to system

   store1_table(1) := 100;           --  index tables still declared;
   store2_table(2) := 200;           --  but truncated.
   ...
END;
```

GET_PACKAGE_MEMORY_UTILIZATION Procedure

This procedure describes static package memory usage.

The output collections describe memory usage in each instantiated package. Each package is described by its owner name, package name, used memory amount, and unused allocated memory amount. The amount of unused memory is greater than zero because of memory fragmentation and also because once used free memory chunks initially go to a free list owned by the package memory heap. They are released back to the parent heap only when the FREE_UNUSED_USER_MEMORY Procedure is invoked.

Syntax

```
DBMS_SESSION.GET_PACKAGE_MEMORY_UTILIZATION (
   owner_names     OUT NOCOPY LNAME_ARRAY,
   unit_names      OUT NOCOPY LNAME_ARRAY,
   unit_types      OUT NOCOPY INTEGER_ARRAY,
   used_amounts    OUT NOCOPY INTEGER_ARRAY,
   free_amounts    OUT NOCOPY INTEGER_ARRAY);
```

Parameters

Table 178-6   GET_PACKAGE_MEMORY_UTILIZATION Function Parameters

Parameter	Description
`owner_name`	Owner of package
`unit_name`	Name of package
`unit_types`	Value of the `type#` columns of the dictionary table `obj$`
`used_amounts`	Amount of allocated memory specified in bytes
`free_amounts`	Amount of available memory specified in bytes

IS_ROLE_ENABLED Function

This function determines if the named role is enabled for this session.

> **Note:**
>
> This function is deprecated starting in Oracle Database 19c. Use `DBMS_SESSION.CURRENT_IS_ROLE_ENABLED` or `DBMS_SESSION.SESSION_IS_ROLE_ENABLED` instead.

Syntax

```
DBMS_SESSION.IS_ROLE_ENABLED (
   rolename     VARCHAR2)
  RETURN BOOLEAN;
```

Parameters

Table 178-7 IS_ROLE_ENABLED Function Parameters

Parameter	Description
`rolename`	Name of the role.\

Return Values

Table 178-8 IS_ROLE_ENABLED Function Return Values

Return	Description
`is_role_enabled`	`TRUE` or `FALSE`, depending on whether the role is enabled

IS_SESSION_ALIVE Function

This function determines if the specified session is active.

Syntax

```
DBMS_SESSION.IS_SESSION_ALIVE (
   uniqueid VARCHAR2)
  RETURN BOOLEAN;
```

Parameters

Table 178-9 IS_SESSION_ALIVE Function Parameters

Parameter	Description
`uniqueid`	Unique ID of the session: This is the same one as returned by `UNIQUE_SESSION_ID`.

Return Values

Table 178-10 IS_SESSION_ALIVE Function Return Values

Return	Description
`is_session_alive`	`TRUE` or `FALSE`, depending on whether the session is active

LIST_CONTEXT Procedures

This procedure returns a list of active namespaces and contexts for the current session.

Syntax

```
TYPE AppCtxRecTyp IS RECORD (
   namespace VARCHAR2(30),
   attribute VARCHAR2(30),
   value     VARCHAR2(256));

TYPE AppCtxTabTyp IS TABLE OF AppCtxRecTyp INDEX BY BINARY_INTEGER;

DBMS_SESSION.LIST_CONTEXT (
   list OUT AppCtxTabTyp,
   size OUT NUMBER);
```

Parameters

Table 178-11 LIST_CONTEXT Procedure Parameters

Parameter	Description
`list`	Buffer to store a list of application context set in the current session

Return Values

Table 178-12 LIST_CONTEXT Procedure Return Values

Return	Description
`list`	A list of (namespace, attribute, values) set in current session
`size`	Returns the number of entries in the buffer returned

Usage Notes

The context information in the list appears as a series of <`namespace`> <`attribute`> <`value`>. Because `list` is a table type variable, its size is dynamically adjusted to the size of returned list.

MODIFY_PACKAGE_STATE Procedure

This procedure is used to perform various actions (as specified by the `action_flags` parameter) on the session state of all PL/SQL program units active in the session. This takes effect after the PL/SQL call that made the current invocation finishes running.

The procedure uses the `DBMS_SESSION` constants listed in Table 178-14.

Syntax

```
DBMS_SESSION.MODIFY_PACKAGE_STATE(
   action_flags IN PLS_INTEGER);
```

Parameters

Table 178-13 MODIFY_PACKAGE_STATE Procedure Parameters

Parameter	Description
`action_flags`	Bit flags that determine the action taken on PL/SQL program units: `DBMS_SESSION.FREE_ALL_RESOURCES` (or `1`)—frees all memory associated with each of the previously run PL/SQL programs from the session. Clears the current values of any package globals and closes cached cursors. On subsequent use, the PL/SQL program units are reinstantiated and package globals are reinitialized. Invoking `MODIFY_PACKAGE_STATE` with the `DBMS_SESSION.FREE_ALL_RESOURCES` parameter provides functionality identical to the `DBMS_SESSION.RESET_PACKAGE()` interface. `DBMS_SESSION.REINITIALIZE` (or `2`)—reinitializes packages without actually being freed and recreated from scratch. Instead the package memory is reused. In terms of program semantics, the `DBMS_SESSION.REINITIALIZE` flag is similar to the `DBMS_SESSION.FREE_ALL_RESOURCES` flag in that both have the effect of reinitializing all packages. However, `DBMS_SESSION.REINITIALIZE` should exhibit better performance than the `DBMS_SESSION.FREE_ALL_RESOURCES` option because: • Packages are reinitialized without actually being freed and recreated from scratch. Instead the package memory gets reused. • Any open cursors are closed, semantically speaking. However, the cursor resource is not actually freed. It is simply returned to the PL/SQL cursor cache. The cursor cache is not flushed. Hence, cursors corresponding to frequently accessed static SQL in PL/SQL remains cached in the PL/SQL cursor cache and the application does not incur the overhead of opening, parsing, and closing a new cursor for those statements on subsequent use. • The session memory for PL/SQL modules without global state (such as types, stored-procedures) are not freed and recreated.

Usage Notes

See the parameter descriptions in Table 178-16 for the differences between the flags and why `DBMS_SESSION.REINITIALIZE` exhibits better performance than `DBMS_SESSION.FREE_ALL_RESOURCES`.

Table 178-14 Action_flags Constants for MODIFY_PACKAGE_STATE

Constant	Description
`FREE_ALL_RESOURCES`	`PLS_INTEGER:= 1`
`REINITIALIZE`	`PLS_INTEGER:= 2`

- Reinitialization refers to the process of resetting all package variables to their initial values and running the initialization block (if any) in the package bodies. Consider the package:

```
package P is
  n number;
  m number := P2.foo;
  d date := SYSDATE;
  cursor c is select * from emp;
  procedure bar;
end P;
/
package body P is
  v   varchar2(20) := 'hello';
  procedure bar is
  begin
    ...
  end;
  procedure init_pkg is
  begin
    ....
  end;
begin
  -- initialization block
  init_pkg;
  ...
  ...
end P;
/
```

For the package P, reinitialization involves:

- Setting `P.n` to `NULL`
- Invoking function `P2.foo` and setting `P.m` to the value returned from `P2.foo`
- Setting `P.d` to the return value of `SYSDATE` built-in
- Closing cursor `P.c` if it was previously opened
- Setting `P.v` to 'hello'
- Running the initialization block in the package body
- The reinitialization for a package is done only if the package is actually referenced subsequently. Furthermore, the packages are reinitialized in the order in which they are referenced subsequently.
- When using `FREE_ALL_RESOURCES` or `REINITIALIZE`, make sure that resetting package variable values does not affect the application.
- Because `DBMS_SESSION.REINITIALIZE` does not actually cause all the package state to be freed, in some situations, the application could use significantly more session memory than if the `FREE_ALL_RESOURCES` flag or the `RESET_PACKAGE` procedure had been used. For instance, after performing `DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.REINITIALIZE)`, if the application does not refer to many of the packages that were previously referenced, then the session memory for those packages remains until the end of the session (or until `DBMS_SESSION.RESET_PACKAGE` is called).
- Because the client-side PL/SQL code cannot reference remote package variables or constants, you must explicitly use the values of the constants. For example, `DBMS_SESSION.MODIFY_PACKAGE_STATE(DBMS_SESSION.REINITIALIZE)` does not compile on the client because it uses the constant `DBMS_SESSION.REINITIALIZE`.

  Instead, use `DBMS_SESSION.MODIFY_PACKAGE_STATE(2)` on the client, because the argument is explicitly provided.

Examples

This example illustrates the use of `DBMS_SESSION.MODIFY_PACKAGE_STATE`. Consider a package `P` with some global state (a cursor `c` and a number `cnt`). When the package is first initialized, the package variable `cnt` is `0` and the cursor `c` is `CLOSED`. Then, in the session, change the value of `cnt` to `111` and also execute an `OPEN` operation on the cursor. If you call `print_status` to display the state of the package, you see that `cnt` is `111` and that the cursor is `OPEN`. Next, call `DBMS_SESSION.MODIFY_PACKAGE_STATE`. If you print the status of the package `P` again using `print_status`, you see that `cnt` is `0` again and the cursor is `CLOSED`. If the call to `DBMS_SESSION.MODIFY_PACKAGE_STATE` had not been made, then the second `print_status` would have printed `111` and `OPEN`.

```
create or replace package P is
  cnt    number := 0;
  cursor c is select * from emp;
  procedure print_status;
end P;
/
show errors;

create or replace package body P is
  procedure print_status is
  begin
    dbms_output.put_line('P.cnt = ' || cnt);
    if c%ISOPEN then
      dbms_output.put_line('P.c is OPEN');
    else
      dbms_output.put_line('P.c is CLOSED');
    end if;
  end;
end P;
/
show errors;

SQL> set serveroutput on;
SQL> begin
  2   P.cnt := 111;
  3   open p.c;
  4   P.print_status;
  5  end;
  6  /
P.cnt = 111
P.c is OPEN

PL/SQL procedure successfully completed.

SQL> begin
  2   dbms_session.modify_package_state(dbms_session.reinitialize);
  3  end;
  4  /

PL/SQL procedure successfully completed.

SQL> set serveroutput on;
SQL>
SQL> begin
  2   P.print_status;
  3  end;
  4  /
P.cnt = 0
```

```
P.c is CLOSED

PL/SQL procedure successfully completed.
```

RESET_PACKAGE Procedure

This procedure de-instantiates all packages in this session. It frees the package state.

Note:

See "SESSION _TRACE_ENABLE Procedure" . The MODIFY_PACKAGE_STATE interface, introduced in Oracle9i, provides an equivalent of the RESET_PACKAGE capability. It is an efficient, lighter-weight variant for reinitializing the state of all PL/SQL packages in the session.

Memory used for caching the execution state is associated with all PL/SQL functions, procedures, and packages that were run in a session.

For packages, this collection of memory holds the current values of package variables and controls the cache of cursors opened by the respective PL/SQL programs. A call to `RESET_PACKAGE` frees the memory associated with each of the previously run PL/SQL programs from the session, and, consequently, clears the current values of any package globals and closes any cached cursors.

`RESET_PACKAGE` can also be used to reliably restart a failed program in a session. If a program containing package variables fails, then it is hard to determine which variables need to be reinitialized. `RESET_PACKAGE` guarantees that all package variables are reset to their initial values.

Syntax

```
DBMS_SESSION.RESET_PACKAGE;
```

Usage Notes

Because the amount of memory consumed by all executed PL/SQL can become large, you might use `RESET_PACKAGE` to trim down the session memory footprint at certain points in your database application. However, make sure that resetting package variable values does not affect the application. Also, remember that later execution of programs that have lost their cached memory and cursors will perform slower, because they need to re-create the freed memory and cursors.

`RESET_PACKAGE` does not free the memory, cursors, and package variables immediately when called.

Note:

`RESET_PACKAGE` only frees the memory, cursors, and package variables after the PL/SQL call that made the invocation finishes running.

For example, PL/SQL procedure `P1` calls PL/SQL procedure `P2`, and `P2` calls `RESET_PACKAGE`. The `RESET_PACKAGE` effects do not occur until procedure `P1` finishes execution (the PL/SQL call ends).

Examples

This SQL*Plus script runs a large program with many PL/SQL program units that may or may not use global variables, but it doesn't need them beyond this execution:

```
EXCECUTE large_plsql_program1;
```

To free up PL/SQL cached session memory:

```
EXECUTE DBMS_SESSION.RESET_PACKAGE;
```

To run another large program:

```
EXECUTE large_plsql_program2;
```

SESSION_IS_ROLE_ENABLED Function

This function determines if the named role is enabled for the login user session.

Syntax

```
DBMS_SESSION.SESSION_IS_ROLE_ENABLED (
   rolename    VARCHAR2)
  RETURN BOOLEAN;
```

Parameters

Table 178-15 SESSION_IS_ROLE_ENABLED Function Parameters

Parameter	Description
`rolename`	Name of the role.

Return Values

- `TRUE`-if the role is enabled.
- `FALSE`-if the role is not enabled.

SESSION_TRACE_DISABLE Procedure

This procedure resets the session-level SQL trace for the session from which it was called. Client ID and service/module/action traces are not affected.

Syntax

```
DBMS_SESSION.SESSION_TRACE_DISABLE;
```

SESSION _TRACE_ENABLE Procedure

This procedure enables session-level SQL trace for the invoking session. Invoking this procedure results in SQL tracing of every SQL statement issued by the session.

Syntax

```
DBMS_SESSION.SESSION_TRACE_ENABLE(
   waits     IN   BOOLEAN DEFAULT TRUE,
   binds     IN   BOOLEAN DEFAULT FALSE,
   plan_stat IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 178-16 SESSION_TRACE_ENABLE Procedure Parameters

Parameter	Description
`waits`	Specifies if wait information is to be traced
`binds`	Specifies if bind information is to be traced
`plan_stat`	Frequency at which we dump row source statistics. Value should be '`NEVER`', '`FIRST_EXECUTION`' (equivalent to `NULL`) or '`ALL_EXECUTIONS`'.

SET_CONTEXT Procedure

This procedure sets the context, of which there are four types: session local, globally initialized, externally initialized, and globally accessed.

Of the five parameters, only the first three are required; the final two parameters are optional, used only in globally accessed contexts. Further parameter information appears in the parameter table and the usage notes.

Syntax

```
DBMS_SESSION.SET_CONTEXT (
   namespace VARCHAR2,
   attribute VARCHAR2,
   value     VARCHAR2,
   username  VARCHAR2,
   client_id VARCHAR2 );
```

Parameters

Table 178-17 SET_CONTEXT Procedure Parameters

Parameter	Description
`namespace`	The namespace of the application context to be set, limited to 128 bytes. Exceeding the maximum permissible length will result in an error during the execution of the procedure.
`attribute`	The attribute of the application context to be set, limited to 128 bytes. Exceeding the maximum permissible length will result in an error during the execution of the procedure.
`value`	Value of the application context to be set, limited to 4 kilobytes.

Table 178-17 (Cont.) SET_CONTEXT Procedure Parameters

Parameter	Description
`username`	Database username attribute of the application context, limited to 128 bytes. Exceeding the maximum permissible length will result in an error during the execution of the procedure. The default value is `NULL`.
`client_id`	Application-specific client_id attribute of the application context, limited to 64 bytes. The default value is `NULL`.

Usage Notes

- The first three parameters are required for all types of context.
- The `username` parameter must be a valid SQL identifier.
- The `client_id` parameter must be a string of at most 64 bytes. It is case-sensitive and must match the argument provided for `set_identifier`.
- If the namespace parameter is a global context namespace, then the `username` parameter is matched against the current database user name in the session, and the `client_id` parameter is matched against the current `client_id` in the session. If these parameters are not set, NULL is assumed, enabling any user to see the context values.
- This procedure must be invoked directly or indirectly by the trusted package.
- The caller of SET_CONTEXT must be in the calling stack of a procedure that has been associated to the context namespace through a `CREATE CONTEXT` statement. The checking of the calling stack does not cross a DBMS boundary.
- No limit applies to the number of attributes that can be set in a namespace. An attribute retains its value during the user's session unless it is reset by the user.
- If the value of the parameter in the namespace has been set, `SET_CONTEXT` overwrites this value.
- Any changes in context value are reflected immediately and subsequent calls to access the value through `SYS_CONTEXT` return the most recent value.

See Also:

Oracle Database Security Guide for more information about

- "Setting the username and client ID"
- "Example: Creating a Global Application Context that Uses a Client Session ID"

SET_EDITION_DEFERRED Procedure

This procedure requests a switch to the specified edition. The switch takes effect at the end of the current client call.

Syntax

```
DBMS_SESSION.SET_EDITION_DEFERRED (
   edition     IN     VARCHAR2);
```

Parameters

Table 178-18 SET_EDITION_DEFERRED Procedure Parameters

Parameter	Description
`edition`	Name of the edition to which to switch. The contents of the string are processed as a SQL identifier; double quotation marks must surround the remainder of the string if special characters or lower case characters are present in the edition's actual name and, if double quotation marks are not used, the contents are set in uppercase. The caller must have `USE` privilege on the named edition.

SET_IDENTIFIER Procedure

This procedure sets the client ID in the session.

Syntax

```
DBMS_SESSION.SET_IDENTIFIER (
   client_id VARCHAR2);
```

Parameters

Table 178-19 SET_IDENTIFIER Procedure Parameters

Parameter	Description
`client_id`	Case-sensitive application-specific identifier of the current database session. The maximum number of bytes for this parameter is 64 bytes. If the input exceeds 64 bytes, then `ORA-28264` is raised.

Usage Notes

- `SET_IDENTIFIER` sets the session's client id to the given value. This value can be used to identify sessions in `v$session` by means of `v$session.client_identifier`. It can also be used to identify sessions by means of `sys_context('USERENV','CLIENT_IDENTIFIER')`.
- This procedure is executable by `PUBLIC`.

SET_NLS Procedure

This procedure sets up your Globalization Support (NLS). It is equivalent to the SQL statement: `ALTER SESSION SET <nls_parameter> = <value>`.

Syntax

```
DBMS_SESSION.SET_NLS (
   param VARCHAR2,
   value VARCHAR2);
```

Parameters

Table 178-20 SET_NLS Procedure Parameters

Parameter	Description
`param`	Globalization Support parameter. The parameter name must begin with 'NLS'.
`value`	Parameter value. If the parameter is a text literal, then it needs embedded single-quotes. For example, "`set_nls` ('`nls_date_format`','"`DD-MON-YY`"')".

SET_ROLE Procedure

This procedure enables and disables roles. It is equivalent to the `SET ROLE` SQL statement.

Syntax

```
DBMS_SESSION.SET_ROLE (
   role_cmd VARCHAR2);
```

Parameters

Table 178-21 SET_ROLE Procedure Parameters

Parameter	Description
`role_cmd`	Text is appended to "set role" and then run as SQL

Usage Notes

Note that the procedure creates a new transaction if it is not invoked from within an existing transaction.

SET_SQL_TRACE Procedure

This procedure turns tracing on or off. It is equivalent to the SQL statement `ALTER SESSION SET SQL_TRACE ....`

Syntax

```
DBMS_SESSION.SET_SQL_TRACE (
   sql_trace boolean);
```

Parameters

Table 178-22 SET_SQL_TRACE Procedure Parameters

Parameter	Description
`sql_trace`	`TRUE` turns tracing on, `FALSE` turns tracing off

SLEEP Procedure

This procedure suspends the session for a specified period of time.

Syntax

```
DBMS_SESSION.SLEEP (
   seconds   IN NUMBER);
```

Parameters

Table 178-23 SLEEP Procedure Parameters

Parameter	Description
`seconds`	Amount of time, in seconds, to suspend the session. The smallest increment can be entered in hundredths of a second; for example, 1.95 is a legal time value.

SWITCH_CURRENT_CONSUMER_GROUP Procedure

This procedure changes the current resource consumer group of a user's current session.

This lets you switch to a consumer group if you have the switch privilege for that particular group. If the caller is another procedure, then this enables the user to switch to a consumer group for which the owner of that procedure has switch privilege.

Syntax

```
DBMS_SESSION.switch_current_consumer_group (
   new_consumer_group     IN  VARCHAR2,
   old_consumer_group     OUT VARCHAR2,
   initial_group_on_error IN  BOOLEAN);
```

Parameters

Table 178-24 SWITCH_CURRENT_CONSUMER_GROUP Procedure Parameters

Parameter	Description
`new_consumer_group`	Name of consumer group to which you want to switch
`old_consumer_group`	Name of the consumer group from which you just switched out
`initial_group_on_error`	If `TRUE`, then sets the current consumer group of the caller to his/her initial consumer group in the event of an error

Return Values

This procedure outputs the old consumer group of the user in the parameter `old_consumer_group`.

> **Note:**
>
> - The `old_consumer_group` parameter returns the name of old consumer group only if it were set explicitly. That is, you might get NULL if the old consumer group was set by some mapping rules.
> - You can switch back to the old consumer group later using the value returned in `old_consumer_group`.

Exceptions

Table 178-25 SWITCH_CURRENT_CONSUMER_GROUP Procedure Exceptions

Exception	Description
`29368`	Non-existent consumer group
`1031`	Insufficient privileges
`29396`	Cannot switch to `OTHER_GROUPS` consumer group

Usage Notes

The owner of a procedure must have privileges on the group from which a user was switched (`old_consumer_group`) in order to switch them back. There is one exception: The procedure can always switch the user back to his/her initial consumer group (skipping the privilege check).

By setting `initial_group_on_error` to `TRUE`, `SWITCH_CURRENT_CONSUMER_GROUP` puts the current session into the default group, if it can't put it into the group designated by `new_consumer_group`. The error associated with the attempt to move a session into `new_consumer_group` is raised, even though the current consumer group has been changed to the initial consumer group.

Examples

```
CREATE OR REPLACE PROCEDURE high_priority_task is
   old_group varchar2(30);
   prev_group varchar2(30);
   curr_user varchar2(30);
BEGIN
  -- switch invoker to privileged consumer group. If we fail to do so, an
  -- error is thrown, but the consumer group does not change
  -- because 'initial_group_on_error' is set to FALSE

  dbms_session.switch_current_consumer_group('tkrogrp1', old_group, FALSE);
  -- set up exception handler (in the event of an error, we do not want to
  -- return to caller while leaving the session still in the privileged
  -- group)

  BEGIN
```

```
    -- perform some operations while under privileged group

  EXCEPTION
    WHEN OTHERS THEN
     -- It is possible that the procedure owner does not have privileges
     -- on old_group. 'initial_group_on_error' is set to TRUE to make sure
     -- that the user is moved out of the privileged group in such a
     -- situation

     dbms_session.switch_current_consumer_group(old_group,prev_group,TRUE);
     RAISE;
    END;

  -- we've succeeded. Now switch to old_group, or if cannot do so, switch
  -- to caller's initial consumer group

  dbms_session.switch_current_consumer_group(old_group,prev_group,TRUE);
END high_priority_task;
/
```

UNIQUE_SESSION_ID Function

This function returns an identifier that is unique for all sessions currently connected to this database. Multiple calls to this function during the same session always return the same result.

Syntax

```
DBMS_SESSION.UNIQUE_SESSION_ID
  RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(unique_session_id,WNDS,RNDS,WNPS);
```

Return Values

Table 178-26   UNIQUE_SESSION_ID Function Return Values

Return	Description
`unique_session_id`	Returns up to 24 bytes

USE_DEFAULT_EDITION_ALWAYS procedure

This procedure turns a mode 'on' or 'off' that disassociates the session from an edition at the end of each and every client call.

Following each call and before the next top level call, no edition will be in use by the session. The session will use the reigning database default edition on its next operation.

Syntax

```
DBMS_SESSION.USE_DEFAULT_EDITION_ALWAYS(
                        mode_on IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 178-27 USE_DEFAULT_EDITION_ALWAYS Procedure Parameters

Parameter	Description
`mode_on`	`TRUE` to turn on this mode, `FALSE` to turn off this mode.

Usage Notes

- A choice of session edition that might otherwise occur due to use of `ALTER SESSION SET EDITION`, `SET_EDITION_DEFERRED`, `USE_DEFAULT_EDITION_DEFERRED`, or the edition-choosing aspect of `ALTER SESSION SET CONTAINER` will be overridden by this mode.
- Turn this mode off before invoking the above mechanisms to avoid the overriding effect.

USE_DEFAULT_EDITION_DEFERRED procedure

This procedure disassociates the session from an edition.

The reset takes effect at the end of the current client call. Following this call, and before the next top level call, no edition will be in use by the session. The session will use the reigning database default edition on its next operation.

Syntax

```
DBMS_SESSION.USE_DEFAULT_EDITION_DEFERRED
```