C

Using Supplied Packages

Using Supplied Packages

You can take advantage of a number of pre-configured packages supplied by with Oracle7.

The DBMS_OUTPUT Package Performs the Following:

Outputs values and messages from triggers, stored procedures and functions.

DBMS_OUTPUT Procedures:

.PUT	Append text from the procedure to the current line of the line output buffer.
.NEW_LINE	Place an end_of_line marker in the output buffer.
.PUT_LINE	Combine the action of PUT and NEW_LINE.
.GET_LINE	Retrieve the current line from the output buffer into the procedure.
.GET_LINES	Retrieve an array of lines from the output buffer into the procedure.
.ENABLE/DISABLE	Enable/disable calls to the DBMS_OUTPUT procedures.

Note:

- These procedures will be available as soon as your DBA has run the dbmsotpt.sql script within the sys schema.
- Within SQL*Plus or SQL*DBA, just set SERVEROUTPUT option to ON/OFF, instead of executing ENABLE/DISABLE procedures.

Practical Uses

You can output intermediary results to the screen for debugging purposes.

This package:

- Allows developers to closely follow the execution of a function or procedure by sending messages and values to the output buffer.

Using Supplied Packages

The DBMS_DDL Package Performs the Following:

Recompiles procedures, functions and packages, and analyze indexes, tables and clusters.

DBMS_DDL Procedures:

.ALTER_COMPILE	Recompile procedures, functions or packages.
.ANALYZE_OBJECT	Analyze indexes, tables or clusters, on rows or percentage of rows estimation.

Note: The use of DBMS_DDL is not allowed in triggers, in procedures called from SQL*Forms or in remote sessions.

Practical Uses

- You can recompile your modified procedures, functions and packages using DBMS_DDL.ALTER_COMPILE.
- Analyze a single object, using DBMS_DDL.ANALYZE_OBJECT. (There is a way of analyzing more than one object at a time, using DBMS_UTILITY.ANALYZE_SCHEMA.)

This package:

- Allows developers to have access to COMPILE and ANALYZE SQL statements through PL/SQL environments.

Using Supplied Packages

The DBMS_UTILITY Package Performs the Following:

Analyzes objects in a particular schema, checks whether the server is running in parallel mode, and returns the time.

DBMS_UTILITY Procedures:

.COMPILE_SCHEMA	Compile all procedures, functions and packages in the specified schema.
.ANALYZE_SCHEMA	Analyze all the tables, clusters and indexes in the specified schema, on rows or percentage of rows estimation.

DBMS_UTILITY Functions:

.IS_PARALLEL_SERVER	Return a boolean specifying if you are running in Parallel Server mode or not.
.GET_TIME	Return time in hundredths of a second, to get the difference between two dates.

Note: The COMPILE_SCHEMA procedure is not allowed in triggers, in procedures called SQL*Forms, in remote sessions.

Practical Uses

- You can recompile all procedures, functions and packages in one scheme, by issusing only one command, DBMS_UTILITY.COMPILE_SCHEMA.
- You can analyse all tables, indexes and clusters on one schema. By using this procedure in conjunction with DBMS_JOB, you can regulary gather statistics for use with the Cost Based Optimiser.

This package:

- Allows developers to COMPILE and ANALYZE all the objects of a schema at once.

The DBMS_SESSION Package Performs the Following:

Alters the user session, sets the role of the user, and deinstantiates all packages in a session.

DBMS_SESSION Procedures:

.SET_SQL_TRACE	Turn tracing on or off for a user's session.
.SET_NLS	Set the values of the Native Language Service parameters for a user's session (language, date format, numeric characters, currency...).
.CLOSE_DATABASE_LINK	Close the database link dblink, eliminating a session's connection to a remote database.
.SET_LABEL	Set the value of the session label in Trusted Oracle.
.SET_MLS_LABEL_FORMAT	Set the default session format for a user.
.SET_CLOSE_CACHED_OPEN_CURSORS	Keep control on opened and cached in memory cursors to automatically close or hold them opened after each COMMIT statement.
.RESET_PACKAGE	Free all package states in a session.
.SET_ROLE	Activate one or several granted roles for a user's session.

DBMS_SESSION Functions:

.IS_ROLE_ENABLED	Determine if the named role is activated or not.
.UNIQUE_SESSION_ID	Return the unique value assigned to a session.

Note:

- RESET_PACKAGE procedure is not allowed in recursive sessions.
- SET_ROLE, SET_NLS procedures are not allowed in triggers.
- SET_LABEL, SET_MLS_LABEL_FORMAT, SET_NLS procedures are not allowed in remote sessions.
- SET_ROLE procedure, if invoked from stored procedures, functions and packages, will not affect the security role for embedded SQL statements or procedure calls, since PL/SQL does the security check on SQL when the block is compiled. (DBMS_SQL package is however not subject to this restriction).

Practical Uses

During the execution of a procedure, you can activate and deactivate the setting of certain roles for the user session. This allows the user to temporarily gain certain privileges.

This package:

- Allows developers to have access to ALTER SESSION and SET ROLE SQL statement through PL/SQL environments.

The DBMS_SHARED_POOL Package Performs the Following:

Accesses the shared pool areas, allows PL/SQL objects and SQL cursors to be locked or kept in shared memory that they will not be aged out using the normal Least Recently Used (LRU) mechanism, and allows you to unlock them when they are no longer needed, and then they will use the LRU mechanism to age out of memory.

DBMS_SHARED_POOL Procedures:

.SIZES	Show all objects in the shared pool that are larger in kilobytes than the specified size.
.KEEP	Keep a PL/SQL object or an SQL cursor in the shared pool area so that it is not subject to aging out of the pool.
.UNKEEP	Not keep a PL/SQL object or an SQL cursor in the shared pool area.

Note: Within SQL*Plus or SQL*DBA, just set SERVEROUTPUT ON SIZE option to a size to display the results of the SIZES procedure.

Practical Uses

You can use this package to keep objects in the shared memory. This will be useful for those packages and/or procedures that you do not wish to age out of memory. (Considerations have to be made to avoid reorganising memory to create large contiguous areas, for large objects that frequently need to be reloaded into memory.)

This package:

- Prevents users' response time decrease when loading large PL/SQL objects, because it enables the developer to ensure that memory is available.
- Allows developers to have access to the shared pool area to display the size of objects and eventually reduce memory fragmentation by keeping or not keeping them.

The DBMS_TRANSACTION Package Performs the Following:

Controls logical transactions and improves the performance of short, non-distributed transactions by creating them as discrete.

DBMS_TRANSACTION Procedures:

.READ_ONLY	Set a transaction to read only.
.READ_WRITE	Set a transaction to read and write.
.USE_ROLLBACK_SEGMENT	Set a transaction to use a specific rollback segment.
.ADVISE_COMMIT	Send advice to remote database for forcing an in-doubt distributed transaction to commit.
.ADVISE_ROLLBACK	Send advice to remote database for forcing an in-doubt distributed transaction to roll back.
.ADVISE_NOTHING	Send no particular advice to the remote database.
.COMMIT	Validate a transaction.
.COMMIT_COMMENT	Specify a text to be associated with the current transaction being validated.
.COMMIT_FORCE	Manually validate an in-doubt distributed transaction.
.ROLLBACK	Roll back a transaction.
.ROLLBACK_FORCE	Manually roll back an in-doubt distributed transaction.
.SAVEPOINT	Assign a name to a savepoint in a transaction.
.ROLLBACK_SAVEPOINT	Partially roll back a transaction to a specified savepoint.

. PURGE_MIXED	Manually delete information about a given mixed outcome transaction: forced by some sites to commit and by others to roll back.
.BEGIN_DISCRETE_ TRANSACTION:	Set a transaction to work in discrete mode, so as to execute it faster. No undo information is generated and therefore no query on changed data by discrete transactions is accepted.

DBMS TRANSACTION Functions:

.LOCAL_TRANSACTION_ID	Return the unique value assigned to a transaction or starts a new transaction if none.

Note:

- All information resulting from ADVISE_xxx, COMMIT_ xxx and ROLLBACK_xxx procedures is stored in DBA_2PC_PENDING data dictionary view.
- COMMIT_FORCE and ROLLBACK_FORCE procedures require FORCE [ANY] TRANSACTION system privilege.
- BEGIN_DISCRETE_TRANSACTION procedure will work properly if the DISCRETE_TRANSACTIONS_ENABLED initialization parameter is set to TRUE.

Practical Uses

You can improve the performance of short, non-distributed transactions by using the DBMS_TRANSACTION.BEGIN_DISCRETE_TRANSACTION procedure. This procedure streamlines transaction processing so short transactions can execute more rapidly. However, it can only be used in certain circumstances.

This package:

- Allows developers to have access to SQL transactions statements from stored procedures and to monitor transaction activities from stored functions.

The DBMS_MAIL Package Performs the Following:

Sends messages from the Oracle Server directly to an Oracle*Mail identifier.

DBMS_MAIL Procedures:

.SEND	Send the text of a message to an Oracle*Mail identifier and copy, blank copy and reply–to Oracle*Mail identifiers.

Note:

- The Distributed Option is required.
- Refer to the *dbmsmail.sql* script for the other consecutive actions that have to be executed.
- This SEND procedure may be called from a trigger to automate sending message on specific actions.

Practical Uses

You can send a message to alert the inventory manager when a stock item has decreased below a certain value, by using a trigger after the update of the INVENTORY table.

This package:

- Allows any Oracle tool that can store procedures, to send messages to the Oracle*Mail, and therefore automate sending messages to the Oracle*Mail on specific actions.

The DBMS_PIPE Package Performs the Following:

Sends messages from one session to another in the same instance.

DBMS_PIPE Procedures:

.PACK_MESSAGE	Pack an item into the local message buffer, that is, to be sent by the SEND_MESSAGE function. (varchar2, number or date type item).
.PACK_MESSAGE_RAW	Pack a raw item into the local message buffer.
.PACK_MESSAGE_ROWID	Pack a rowid item into the local message buffer.

DBMS_PIPE Functions:

.SEND_MESSAGE	Send a message contained in the local message buffer to the named pipe.
.RECEIVE_MESSAGE	Retrieve a message from the named pipe and put it into the local message buffer to be unpacked by the UNPACK procedure.

DBMS_PIPE Procedures:

.UNPACK_MESSAGE	Unpack an item from the local message buffer (varchar2, number or date type item).
.UNPACK_MESSAGE_ROW	Unpack a raw item from the local message buffer.
.UNPACK_MESSAGE_ROWID	Unpack a rowid item from the local message buffer.

DBMS_PIPE Functions:

.NEXT_ITEM_TYPE	Get the type of the next item in the local message buffer (varchar2, number, date, raw, rowid).
.UNIQUE_SESSION_NAME	Get the unique name of a session.

DBMS_PIPE Procedures:

.PURGE	Empty out the named pipe, removing it from the SGA.

Practical Uses

- You can use DBMS_PIPE to send debugging information from triggers and/or stored procedures. Another session could kepp reading the pipe and the contents on a screen or to a file.
- You can set up a PIPE to communicate to a different session that can be performing an operation in an independent transaction, for example, logging an attempted security violation detected by a trigger.

This package:

- Allows developers to use the pipe service of the SGA memory to send messages between sessions. This way they communicate information to a separate session which can perform an operation in an independent transaction (in applications or during debugging or as alerters).
- Allows PL/SQL to execute system commands (like HOST in SQL*Plus).

The DBMS_ALERT Package Performs the Following:

Waits for an event in the database, when it happens or a time-out expires, and signals it.

DBMS_ALERT Procedures:

.REGISTER	Register interest of an alert for a session. A session may register an unlimited number of alerts.
.WAITANY	Wait for an alert to occur for any of the alerts for which this session is registered, until the time-out period expires.
.WAITONE	Wait for specified alert to occur until the time-out period expires. If multiple signals on this alert occurred before the waitone call the most,recent message will be considered. If the return status is 0 : alert occurred If the return status is 1 : time-out occurred
.SIGNAL	Signal an alert to all sessions that have registered interest in it, passing a message to them. Sessions that are currently waiting will be awakened, otherwise they will be notified the next time they make a wait call.
.REMOVE	Remove alert from registration list when a session is no longer interested in it.
.REMOVEALL	Remove all alerts for this session from registration list when this session is no longer interested in any alerts. If a session dies without removing all of its alerts, the alerts will eventually be cleaned up, but not immediately.
.SET_DEFAULTS	Set the time to sleep between polls, in case a polling loop is required (default is 5 sec.).

Note:

- If your database is running in parallel mode, a polling loop is required to check for alerts from another instance.
- If you use 'WAITANY' call, and a signalling session makes a signal but does not commit within one second of the signal, then a polling loop is required so that this uncommitted alert does not camouflage other alerts.
- When running with a multi-threaded server configuration, applications which register for alerts should use dedicated servers rather than shared servers so as not to tie these up for too long.
- This package requires the use of dbms pipe and dbms_lock packages.

Practical Uses

- You can alert an application that a database event has occured, for example, register interest in the data changes to the EMP table, wait for a change to occur and when it does, signal it so that a report is automatically generated.
- Wake up a graphic tool displaying a graph of data. When the data is changed, there is a refresh of the graphical display, including the new data.

This package:

- Allows an application, by means of appropriate database triggers, to cause itself to be notified whenever values of interest are changed.

The DBMS_LOCK Package Performs the Following:

Requests, converts and releases user locks, which are managed by the rdbms lock management services.

DBMS_LOCK Procedures:

.ALLOCATE_UNIQUE	Generate a unique lockid and assign a lockhandle whose name may be used by a session to refer to the same lock. This lockhandle is used to reduce the chance that a programming error can accidentally create an incorrect but valid lockid.

DBMS_LOCK Functions:

.REQUEST	Request a lock with the given mode, for the given lockid or the lockhandle name.
.CONVERT	Convert a lock, referenced by its id or handle name, from one mode to another.
.RELEASE	Release a lock, referenced by its id or handle name and previously acquired by REQUEST.

DBMS_LOCK Procedures:

.SLEEP	Suspend the session for a specified number of seconds.

Note:

- These locks will show up in the SQL*DBA lock monitor screen.
- Deadlock detection is performed on these locks.
- When running in Parallel server and Multi-threaded server configuration, a shared server will be bound to a session during the time that any locks are held, and will therefore not be shareable during this time.

Practical Uses

If you want to update the majority of a tables rows, within a procedure and you do not want any other transaction to update any of these rows, set an SSX lock on your table, so that only SELECT FOR UPDATE transactions will get a lock on this table.

This package:

- Allows applications to request an explicit lock in different modes, such as sub-shared (SS), sub-exclusive (SX), share (S), share–sub–exclusive (SSX), exclusive (X) and null (NL), like the LOCK SQL statement.

The DBMS_SQL Package Performs the Following:

Uses dynamic SQL to access the database.

DBMS_SQL Functions:

.OPEN_CURSOR	Open a new cursor and assign a cursor ID number.

DBMS_SQL Procedures:

.PARSE	Parse the DDL or DML statement : check the statement's syntax and associate it with the opened cursor. DDL_statements are immediately executed when parsed.
.BIND_VARIABLE	Bind the given value to the variable identified by its name in the parsed statement in the given cursor.
.DEFINE_COLUMN	Specify the variables that are to receive the SELECT values, such as makes an INTO clause in a static query (only for SELECT statement).

DBMS_SQL Functions:

.EXECUTE	Execute the SQL statement and returns the number of rows processed.
.FETCH_ROWS	Retrieve a row for the specified cursor. For multiple rows, call it in a loop.
.EXECUTE_AND_FETCH	Execute and retrieve a row for the specified cursor. More efficient than the EXECUTE and FETCH_ROWS for a single iteration.

DBMS_SQL Procedures:

.COLUMN_VALUE	Return the value of the cursor element for a given position in a given cursor, for NUMBER, DATE, VARCHAR2 and MLSLABEL datatypes. Is used to access the data fetched by FETCH ROWS call.
.COLUMN_VALUE_CHAR	Return the same information for CHAR.
.COLUMN_VALUE_RAW	Return the same information for RAW.
.COLUMN_VALUE_ROWID	Return the same information for ROWID.
.VARIABLE_VALUE	Return the value of the named variable for a given cursor, for NUMBER, DATE, VARCHAR2 and MLSLABEL datatypes. Is used to retrieve the values assigned to the output variables of PL/SQL procedures once they are executed.
.VARIABLE_VALUE_CHAR	Return the same information for CHAR.
.VARIABLE_VALUE_RAW	Return the same information for RAW.
.VARIABLE_VALUE_ROWID	Return the same information for ROWID.

DBMS_SQL Functions:

.IS_OPEN	Check if the specified cursor is currently opened or not.

This package:

- Allows developers not to embed SQL statements in source programs, but store them in character strings that are input to, or built by, the program at runtime, and therefore permits them to create more general purpose procedures, for example, create a procedure that operates on a table whose name is not known until runtime.
- Allows parsing of any DML or DDL statement, which solves the problem of not being able to parse DDL statements directly using PL/SQL, for example, dropping a table from within a stored procedure.

DBMS_SQL Procedures:

.CLOSE_CURSOR	Close the specified cursor.

DBMS_SQL Functions locating Errors:

.LAST_ERROR_POSITION	Return the byte offset in the SQL statement text where the error occurred (first character is 0).
.LAST_ROW_COUNT	Return the cumulative count of the number of rows fetched.
.LAST_ROW_ID	Return the ROWID of the last row fetched.
.LAST_SQL_FUNCTION_CODE	Return the SQL function code for the statement. Use this function immediately after the execution of the SQL statement, otherwise the return value is not pertinent.

Note:

- Using this package to execute DDL statements can result in a program hang.
- The operations provided by this package are performed under the current user not under the package owner SYS. Therefore, if the caller is an anonymous PL/SQL block, the operations are performed according to the privileges of the current user; if the caller is a stored procedure, the operations are performed according to the owner of the stored procedure.
- OCIs use bind by address, whereas this package uses bind by values.
- DBMS_SESSION.SET_ROLE procedure call is ineffective with embedded SQL statements in stored procedures, but not with DBMS_SQL procedures.

Practical Uses

- You can execute a procedure that takes a table name, column name and column value as parameter values.
- You can execute a procedure that takes the names of a source and destination table, and copies rows from the source table to the destination table.

Example of dbms_sql

```
create or replace procedure exec_select
(       p_table_name in varchar2,
        p_column_name in varchar2,
        p_column_value in varchar2,
        p_ename out varchar2,
        p_sal out number)
as
        mycursor            integer;
        myreturn            integer;
--
        v_ename scott.emp.ename%type;
        v_sal   scott.emp.sal%type;
        v_column_value varchar2(30);
--
        sql_statement varchar2(200) :=
              'select ename,sal from '||p_table_name||' where'
               ||p_column_name||'=:col_value';
--
begin
/*
        Assign column-value parameter to a variable as cannot
        reference a parameter in call to bind_variable
*/
        v_column_value := upper(p_column_value);
--
        mycursor := dbms_sql.open_cursor;
--
        dbms_sql.parse(mycursor,sql_statement,dbms_sql.native);
--
        dbms_sql.bind_variable(mycursor,'col_value',v_column_value);
--
        dbms_sql.define_column(mycursor,1,v_ename,30);
        dbms_sql.define_column(mycursor,2,v_sal);
--
        myreturn := dbms_sql.execute(mycursor);
        myreturn := dbms_sql.fetch_rows(mycursor);
        if myreturn = 0 then
              raise no_data_found;
        end if;
--
        dbms_sql.column_value(mycursor,1,v_ename);
        dbms_sql.column_value(mycursor,2,v_sal);
--
```

Example of dbms_sql continued

```
        myreturn := dbms_sql.fetch_rows(mycursor);
        if myreturn > 0 then
              raise too_many_rows;
        end if;
--
        p_name := v_ename;
        p_sal  := v_sal
--
        dbms_sql.close_cursor(mycursor);
exception
        when others then
              dbms_output.put('error - sqlcode is : ');
              dbms_output.put_line(sqlcode);
              dbms_output.put_line('message is : '||sqlerrm);
              p_name := null;
              p_sal  := null;
end;
/
To run it in SQL*Plus :
var v_out1 varchar2(10)
var v_out2 number
set serveroutput on
execute exec_select('&table_name','&column_name',-
'&column_value',:v_out1,:v_out2)
print v_out1
print v_out2
```

The DBMS_JOB Package Performs the Following:

Schedules periodic execution of PL/SQL code. It requires at least one SNP background process.

DBMS_JOB Procedures:

.SUBMIT	Submits a job to the job queue (name of the procedure, first date of run, periodicity).
.REMOVE	Removes specified job from the job queue.
.CHANGE	Alters a specified job.
.WHAT	Alters the job description for a specified job (name of the procedure, periodicity).
.NEXT_DATE	Alters the next execution time for a specified job.
.INTERVAL	Alters the interval between executions of a job.
.BROKEN	Disables job execution.
.RUN	Forces a job to run.

Note: Job Queues are used by the Oracle Server's automatic snapshot refresh.

Practical Uses

- You can use DBMS_JOB to submit a job to the job queue, for example, for those objects that need to have statistics gathered regularly, execute DBMS_DDL.ANALYZE_OBJECT or DBMS_DDL.ANALYZE_SCHEMA, every night, and the objects or schema will be analyized for the next days commands.
- You can submit a job every weekend, that drops your indexes and recreates them.

This package:

- Allows users to automate periodic tasks such as reporting.
- Allows DBAs to automate periodic administration tasks such as generating statistics, recreating indexes and so on.

The DBMS_APPLICATION_INFO Package Performs the Following:

Allows application tools and application developers to inform the database of the high level of actions they are currently performing.

DBMS_APPLICATION_INFO Procedures:

.SET_MODULE	Sets the name of the module currently running to a new module
.READ_MODULE	Reads the values of the module and action fields of the current session
.SET_ACTION	Sets the name of the current action within the current module
.SET_CLIENT_INFO	Sets the client info field of the session
.READ CLIENT INFO	Reads the value of the client info field of the current session

Note: The names for module and action appear in the 'module' and 'action' columns of V$SESSION and V$SQLAREA views.

Practical Uses

By registering the name of the modules, the DBA can monitor how the system is being used, and do performance analysis, and resource accounting by module.

This package:

- Allows DBAs to track resource usage by applications.
- Enables developers to relate actions that are happening in the database to the application that is causing the action.

Using Supplied Packages

Supplied Packages Summary

All packages are installed in the script *catproc.sql* or each one can be installed individually.

DBMS_ALERT	*dbmsalrt.sql*
DBMS_APPLICATION_INFO	*dbmsutil.sql*
DBMS_DDL	*dbmsutil.sql*
DBMS_LOCK	*dbmslock.sql*
DBMS_MAIL	*dbmsmail.sql*
DBMS_OUTPUT	*dbmsotpt.sql*
DBMS_PIPE	*dbmspipe.sql*
DBMS_SESSION	*dbmsutil.sql*
DBMS_SHARED_POOL	*dbmsspool.sql*
DBMS_SQL	*dbmssql.sql*
DBMS_TRANSACTION	*dbmsutil.sql*
DBMS_UTILITY	*dbmsutil.sql*