Oracle8*i*

Supplied Packages Reference

Release 8.1.5

February 1999

Part No. A68001-01

ORACLE

Supplied Packages Reference, Release 8.1.5

Part No. A68001-01

Primary Author: Michele Cyran

Contributors: D. Alpern, G. Arora, N. Bhatt, S. Chandrasekar, T. Chang, G. Claborn, R. Decker, A. Downing, J. Draaijer, J. Finnerty, R. Frank, A. Ganesh, J. Gosselin, R. Govindarajan, B. Goyal, S. Harris, B. Himatsingka, C. Iyer, H. Jakobsson, A. Jasuja, M. Jungerman, P. Justus, A. Kalra, M. Kamath, S. Kotsovolos, V. Krishnamurthy, J. Krishnaswamy, J. Kundu, B. Lee, J. Liu, P. Locke, A. Logan, N. Mallavarupu, J. Mallory, R. Mani, S. Mavris, A. Mozes, J. Muller, C. Murray, K. Muthukkaruppan, S. Muthulingam, R. Park, G. Pongracz, T. Portfolio, L. Price, S. Puranik, M. Ramacher, S. Ramakrishnan, D. Raphaely, S. Ray, A. Rhee, S. Samu, U. Sangam, A. Saxena, J. Sharma, E. Soylemez, A. Srinivasan, J. Srinivasan, I. Stocks, R. Sujithan, A. Swaminathan, K. Tarkhanov, A. Tsukerman A. To, S. Urman, S. Vivian, D. Voss, W. Wang, R. Ward, S. Wertheimer, R. Wessman, J. Wijaya, D. Wong, L. Wu, R. Yaseen

Graphic Designer: Valerie Moore

Contents

12 DBMS_DISTRIBUTED_TRUST_ADMIN

13 DBMS_HS

14 DBMS_HS_PASSTHROUGH

22 DBMS_OFFLINE_SNAPSHOT

23 DBMS_OLAP

24 DBMS_ORACLE_TRACE_AGENT

25 DBMS_ORACLE_TRACE_USER

26 DBMS_OUTPUT

29 DBMS_PROFILER

30 DBMS_RANDOM

31 DBMS_RECTIFIER_DIFF

32 DBMS_REFRESH

33 DBMS_REPAIR

34 DBMS_REPCAT

42 DBMS_ROWID

43 DBMS_SESSION

44 DBMS_SHARED_POOL

45 DBMS_SNAPSHOT

46 DBMS_SPACE

47 DBMS_SPACE_ADMIN

Send Us Your Comments

Supplied Packages Reference, Release 8.1.5

Part No. A68001-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev@us.oracle.com
- Fax: (650) 506-7228 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation Manager
  500 Oracle Parkway
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, and telephone number below.

Preface

This reference manual describes the Oracle packages that are shipped with the Oracle8*i*, Release 8.1.5 program. Information in this manual applies to versions of the Oracle Server that run on all platforms, and it does not include system-specific information.

This Preface includes the following sections:

Introduction to Packages

A package is a schema object that groups logically related PL/SQL types, items, and subprograms. Packages usually have two parts: a specification and a body, although sometimes the body is unnecessary. The specification is the interface to your applications; it declares the types, variables, constants, exceptions, cursors, and subprograms available for use. The body fully defines cursors and subprograms, and so implements the specification.

Unlike subprograms, packages cannot be called, parameterized, or nested. Still, the format of a package is similar to that of a subprogram:

```
CREATE PACKAGE name AS  -- specification (visible part)
   -- public type and item declarations
   -- subprogram specifications
END [name];

CREATE PACKAGE BODY name AS  -- body (hidden part)
   -- private type and item declarations
   -- subprogram bodies
[BEGIN
   -- initialization statements]
END [name];
```

The specification holds public declarations, which are visible to your application. The body holds implementation details and private declarations, which are hidden from your application.

You can debug, enhance, or replace a package body without changing the interface (package specification) to the package body.

To create packages and store them permanently in an Oracle database, use the CREATE PACKAGE and CREATE PACKAGE BODY statements, which you can execute interactively from SQL*Plus or Enterprise Manager.

Only the declarations in the package specification are visible and accessible to applications. Implementation details in the package body are hidden and inaccessible. So, you can change the body (implementation) without having to recompile calling programs.

The advantages of using packages include modularity, easier application design, information hiding, added functionality, and better performance.

Audience

This manual is intended for anyone who is using or is interested in Oracle packages. It will also be valuable to systems analysts, project managers, and others interested in the development and tuning of database applications.

This manual assumes that you have a working knowledge of application programming and that you are familiar with the use of Structured Query Language (SQL) to access information in relational database systems.

Certain sections of this manual also assume a knowledge of the basic concepts of object oriented programming.

Related Documents

For more information, see the following manuals in the Oracle8*i* documentation set:

- *Oracle8i Application Developer's Guide - Fundamentals*
- *PL/SQL User's Guide and Reference*

Conventions

The following conventions are used in this manual:

Convention	Meaning
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
UPPERCASE TEXT	Uppercase text is used to call attention to package names, command keywords, database object names, filenames, and so on.
Italicized Text	Italicized words within text are book titles or emphasized words.

`Code Examples`	Commands or statements of SQL, Oracle Enterprise Manager line mode (Server Manager), and SQL*Plus appear in a monospaced font.
	For example:
	`INSERT INTO emp (empno, ename) VALUES (1000, 'SMITH');`
	`ALTER TABLESPACE users ADD DATAFILE 'users2.ora' SIZE 50K;`

< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
$	The dollar sign represents the DIGITAL CommandLanguage prompt in OpenVMS and the Bourne shell prompt in Digital UNIX

Your Comments Are Welcome

We value and appreciate your comments as an Oracle user. As we write, revise, and evaluate our documentation, your opinions are the most important feedback we receive.

You can send comments and suggestions about this manual to the following e-mail address:

infodev@us.oracle.com

If you prefer, you can send letters or faxes containing your comments to the following address:

Server Technologies Documentation Manager

Oracle Corporation

500 Oracle Parkway

Redwood Shores, CA 94065

Fax: (650) 506-7228

1

Oracle Supplied Packages

Oracle provides many packages with the Oracle server, either to extend the functionality of the database or to give PL/SQL access to SQL features. You may take advantage of the functionality provided by these packages when creating your application, or you may simply want to use these packages for ideas in creating your own stored procedures.

> **Note:** Oracle provides packages with various products, such as Oracle Developer and the Oracle Application Server. This manual, however, only covers the packages that Oracle provides with the database server.

This chapter contains the following sections:

- Package Overview
- List of Oracle Supplied Packages
- Subprograms in Supplemental Packages

> **See Also:** For detailed information on how to create your own packages, see *Oracle8i Application Developer's Guide - Fundamentals*.

Package Overview

A *package* is an encapsulated collection of related program objects stored together in the database. Program objects are procedures, functions, variables, constants, cursors, and exceptions.

Packages have many advantages over stand-alone procedures and functions. For example, they:

- Let you organize your application development more efficiently.
- Let you grant privileges more efficiently.
- Let you modify package objects without recompiling dependent schema objects.
- Enable Oracle to read multiple package objects into memory at once.
- Let you *overload* procedures or functions. Overloading means creating multiple procedures with the same name in the same package, each taking arguments of different number or datatype.
- Can contain global variables and cursors that are available to all procedures and functions in the package.

Using Oracle Supplied Packages

Most Oracle supplied packaged are automatically installed when the database is created and the CATPROC.SQL script is run. For example, to create the DBMS_ALERT package, the DBMSALRT.SQL and PRVTALRT.PLB scripts must be run when connected as the user SYS. These scripts, however, are run automatically by the CATPROC.SQL script.

Certain packages are not installed automatically. Special installation instructions for these packages are documented in the individual chapters.

To call a PL/SQL function from SQL, you must either own the function or have EXECUTE privileges on the function. To select from a view defined with a PL/SQL function, you must have SELECT privileges on the view. No separate EXECUTE privileges are needed to select from the view. Instructions on special requirements for packages are documented in the individual chapters.

Creating New Packages

There are two distinct steps to creating a new package:

1. Create the package *specification* with the CREATE PACKAGE statement.

   You can declare program objects in the package specification. Such objects are called *public* objects. Public objects can be referenced outside the package, as well as by other objects in the package.

   Note: It is often more convenient to add the OR REPLACE clause in the CREATE PACKAGE statement

2. Create the package *body* with the CREATE PACKAGE BODY statement.

   You can declare and define program objects in the package body:

   - You must define public objects declared in the package specification.
   - You can declare and define additional package objects. Such objects are called *private* objects. Private objects are declared in the package body rather than in the package specification, so they can be referenced only by other objects in the package: They cannot be referenced outside the package.

     See Also: For more information on creating new packages, see *PL/SQL User's Guide and Reference* and *Oracle8i Application Developer's Guide - Fundamentals*. For more information on storing and executing packages, see *Oracle8i Concepts*.

Separation of Specification and Body

The specification of a package *declares* the public types, variables, constants, and subprograms that are visible outside the immediate scope of the package. The body of a package *defines* the objects declared in the specification, as well as private objects that are not visible to applications outside the package.

Oracle stores the specification and body of a package separately in the database. Other schema objects that call or reference public program objects depend only on the package specification, not on the package body. This distinction allows you to change the definition of a program object in the package body without causing Oracle to invalidate other schema objects that call or reference the program object. Oracle invalidates dependent schema objects only if you change the declaration of the program object in the package specification.

Example The following example shows a package specification for a package named EMPLOYEE_MANAGEMENT. The package contains one stored function and two stored procedures.

```
CREATE PACKAGE employee_management AS
   FUNCTION hire_emp (name VARCHAR2, job VARCHAR2,
      mgr NUMBER, hiredate DATE, sal NUMBER, comm NUMBER,
      deptno NUMBER) RETURN NUMBER;
   PROCEDURE fire_emp (emp_id NUMBER);
   PROCEDURE sal_raise (emp_id NUMBER, sal_incr NUMBER);
END employee_management;
```

The body for this package defines the function and the procedures:

```
CREATE PACKAGE BODY employee_management AS
   FUNCTION hire_emp (name VARCHAR2, job VARCHAR2,
      mgr NUMBER, hiredate DATE, sal NUMBER, comm NUMBER,
      deptno NUMBER) RETURN NUMBER IS
```

The function accepts all arguments for the fields in the employee table except for the employee number. A value for this field is supplied by a sequence. The function returns the sequence number generated by the call to this function.

```
       new_empno    NUMBER(10);

   BEGIN
      SELECT emp_sequence.NEXTVAL INTO new_empno FROM dual;
      INSERT INTO emp VALUES (new_empno, name, job, mgr,
         hiredate, sal, comm, deptno);
      RETURN (new_empno);
   END hire_emp;

   PROCEDURE fire_emp(emp_id IN NUMBER) AS
```

The procedure deletes the employee with an employee number that corresponds to the argument emp_id. If no employee is found, then an exception is raised.

```
   BEGIN
      DELETE FROM emp WHERE empno = emp_id;
      IF SQL%NOTFOUND THEN
      raise_application_error(-20011, 'Invalid Employee
         Number: ' || TO_CHAR(emp_id));
   END IF;
END fire_emp;

PROCEDURE sal_raise (emp_id IN NUMBER, sal_incr IN NUMBER) AS
```

The procedure accepts two arguments. `Emp_id` is a number that corresponds to an employee number. `Sal_incr` is the amount by which to increase the employee's salary.

```
   BEGIN

   -- If employee exists, then update salary with increase.

      UPDATE emp
         SET sal = sal + sal_incr
         WHERE empno = emp_id;
      IF SQL%NOTFOUND THEN
         raise_application_error(-20011, 'Invalid Employee
            Number: ' || TO_CHAR(emp_id));
      END IF;
   END sal_raise;
END employee_management;
```

Note: If you want to try this example, then first create the sequence number `emp_sequence`. You can do this using the following SQL*Plus statement:

```
SQL> EXECUTE CREATE SEQUENCE emp_sequence
   > START WITH 8000 INCREMENT BY 10;
```

Referencing Package Contents

To reference the types, items, and subprograms declared in a package specification, use the dot notation. For example:

```
package_name.type_name
package_name.item_name
package_name.subprogram_name
```

List of Oracle Supplied Packages

This section lists each of the Oracle supplied server packages and indicates where they are described in more detail. These packages run as the invoking user, rather than the package owner. Unless otherwise noted, the packages are callable through public synonyms of the same name.

Caution:

- **The procedures and functions provided in these packages and their external interfaces are reserved by Oracle and are subject to change in future releases.**
- **You must not modify Oracle supplied packages. Doing so could cause internal errors and security violations in the database.**

Table 1–1 List of Oracle Supplied Packages

Package Name	Description	Documentation
Calendar (see Note #2 below)	Provides calendar maintenance functions.	*Oracle8i Time Series User's Guide*
DBMS_ALERT	Provides support for the asynchronous notification of database events.	Chapter 2
DBMS_APPLICATION_INFO	Lets you register an application name with the database for auditing or performance tracking purposes.	Chapter 3
DBMS_AQ	Lets you add a message (of a predefined object type) onto a queue or to dequeue a message.	Chapter 4
DBMS_AQADM	Lets you perform administrative functions on a queue or queue table for messages of a predefined object type.	Chapter 5
DBMS_DDL	Provides access to some SQL DDL statements from stored procedures, and provides special administration operations not available as DDLs.	Chapter 6
DBMS_DEBUG	A PL/SQL API to the PL/SQL debugger layer, Probe, in the Oracle server.	Chapter 7

Table 1–1 List of Oracle Supplied Packages

Package Name	Description	Documentation
DBMS_DEFER	Provides the user interface to a replicated transactional deferred remote procedure call facility. Requires the Distributed Option.	Chapter 8
DBMS_DEFER_QUERY	Permits querying the deferred remote procedure calls (RPC) queue data that is not exposed through views. Requires the Distributed Option.	Chapter 9
DMBS_DEFER_SYS	Provides the system administrator interface to a replicated transactional deferred remote procedure call facility. Requires the Distributed Option.	Chapter 10
DBMS_DESCRIBE	Describes the arguments of a stored procedure with full name translation and security checking.	Chapter 11
DBMS_DISTRIBUTED_TRUST_ADMIN	Maintains the Trusted Database List, which is used to determine if a privileged database link from a particular server can be accepted.	Chapter 12
DBMS_HS	Lets you create and modify objects in the Heterogeneous Services dictionary.	Chapter 13
DBMS_HS_PASSTHROUGH	Lets you use Heterogeneous Services to send pass-through SQL statements to non-Oracle systems.	Chapter 14
DBMS_IOT	Creates a table into which references to the chained rows for an Index Organized Table can be placed using the ANALYZE command.	Chapter 15
DBMS_JOB	Lets you schedule administrative procedures that you want performed at periodic intervals; it is also the interface for the job queue.	Chapter 16
DBMS_LOB	Provides general purpose routines for operations on Oracle Large Object (LOBs) datatypes - BLOB, CLOB (read-write), and BFILEs (read-only).	Chapter 17
DBMS_LOCK	Lets you request, convert and release locks through Oracle Lock Management services.	Chapter 18
DBMS_LOGMNR	Provides functions to initialize and run the log reader.	Chapter 19
DBMS_LOGMNR_D	Queries the dictionary tables of the current database, and creates a text based file containing their contents.	Chapter 20

Table 1–1 List of Oracle Supplied Packages

Package Name	Description	Documentation
DBMS_OFFLINE_OG	Provides public APIs for offline instantiation of master groups.	Chapter 21
DBMS_OFFLINE_SNAPSHOT	Provides public APIs for offline instantiation of snapshots.	Chapter 22
DBMS_OLAP	Provides procedures for summaries, dimensions, and query rewrites.	Chapter 23
DBMS_ORACLE_TRACE_AGENT	Provides client callable interfaces to the Oracle TRACE instrumentation within the Oracle7 Server.	Chapter 24
DBMS_ORACLE_TRACE_USER	Provides public access to the Oracle release 7 Server Oracle TRACE instrumentation for the calling user.	Chapter 25
DBMS_OUTPUT	Accumulates information in a buffer so that it can be retrieved out later.	Chapter 26
DBMS_PCLXUTIL	Provides intra-partition parallelism for creating partition-wise local indexes.	Chapter 27
DBMS_PIPE	Provides a DBMS pipe service which enables messages to be sent between sessions.	Chapter 28
DBMS_PROFILER	Provides a Probe Profiler API to profile existing PL/SQL applications and identify performance bottlenecks.	Chapter 29
DBMS_RANDOM	Provides a built-in random number generator.	Chapter 30
DBMS_RECTIFIER_DIFF	Provides APIs used to detect and resolve data inconsistencies between two replicated sites.	Chapter 31
DBMS_REFRESH	Lets you create groups of snapshots that can be refreshed together to a transactionally consistent point in time. Requires the Distributed Option.	Chapter 32
DBMS_REPAIR	Provides data corruption repair procedures.	Chapter 33
DBMS_REPCAT	Provides routines to administer and update the replication catalog and environment. Requires the Replication Option.	Chapter 34
DBMS_REPCAT_ADMIN	Lets you create users with the privileges needed by the symmetric replication facility. Requires the Replication Option.	Chapter 35

Table 1–1 List of Oracle Supplied Packages

Package Name	Description	Documentation
DBMS_REPCAT_INSTATIATE	Instantiates deployment templates. Requires the Replication Option.	Chapter 36
DBMS_REPCAT_RGT	Controls the maintenance and definition of refresh group templates. Requires the Replication Option.	Chapter 37
DBMS_REPUTIL	Provides routines to generate shadow tables, triggers, and packages for table replication.	Chapter 38
DBMS_RESOURCE_MANAGER	Maintains plans, consumer groups, and plan directives; it also provides semantics so that you may group together changes to the plan schema.	Chapter 39
DBMS_RESOURCE_MANAGER_PRIVS	Maintains privileges associated with resource consumer groups.	Chapter 40
DBMS_RLS	Provides row level security administrative interface.	Chapter 41
DBMS_ROWID	Provides procedures to create ROWIDs and to interpret their contents.	Chapter 42
DBMS_SESSION	Provides access to SQL ALTER SESSION statements, and other session information, from stored procedures.	Chapter 43
DBMS_SHARED_POOL	Lets you keep objects in shared memory, so that they will not be aged out with the normal LRU mechanism.	Chapter 44
DBMS_SNAPSHOT (synonym DBMS_MVIEW)	Lets you refresh snapshots that are not part of the same refresh group and purge logs. Requires the Distributed Option.	Chapter 45
DBMS_SPACE	Provides segment space information not available through standard SQL.	Chapter 46
DBMS_SPACE_ADMIN	Provides tablespace and segment space administration not available through the standard SQL.	Chapter 47
DBMS_SQL	Lets you use dynamic SQL to access the database.	Chapter 48
DBMS_STANDARD	Provides language facilities that help your application interact with Oracle.	(see Note #1 below)
DBMS_STATS	Provides a mechanism for users to view and modify optimizer statistics gathered for database objects.	Chapter 49

Table 1–1 List of Oracle Supplied Packages

Package Name	Description	Documentation
DBMS_TRACE	Provides routines to start and stop PL/SQL tracing.	Chapter 50
DBMS_TRANSACTION	Provides access to SQL transaction statements from stored procedures and monitors transaction activities.	Chapter 51
DBMS_TTS	Checks if the transportable set is self-contained.	Chapter 52
DBMS_UTILITY	Provides various utility routines.	Chapter 53
DEBUG_EXTPROC	Lets you debug external procedures on platforms with debuggers that can attach to a running process.	Chapter 54
OUTLN_PKG	Provides the interface for procedures and functions associated with management of stored outlines.	Chapter 55
PLITBLM	Handles index-table operations.	(see Note #1 below)
SDO_ADMIN (see Note #3 below)	Provides functions implementing spatial index creation and maintenance for spatial objects.	*Oracle8i Spatial User's Guide and Reference*
SDO_GEOM (see Note #3 below)	Provides functions implementing geometric operations on spatial objects.	*Oracle8i Spatial User's Guide and Reference*
SDO_MIGRATE (see Note #3 below)	Provides functions for migrating spatial data from release 7.3.3 and 7.3.4 to 8.1.x.	*Oracle8i Spatial User's Guide and Reference*
SDO_TUNE (see Note #3 below)	Provides functions for selecting parameters that determine the behavior of the spatial indexing scheme used in the Spatial Cartridge.	*Oracle8i Spatial User's Guide and Reference*
STANDARD	Declares types, exceptions, and subprograms which are available automatically to every PL/SQL program.	(see Note #1 below)
TimeSeries (see Note #2 below)	Provides functions that perform operations, such as extraction, retrieval, arithmetic, and aggregation, on time series data.	*Oracle8i Time Series User's Guide*
TimeScale (see Note #2 below)	Provides scaleup and scaledown functions.	*Oracle8i Time Series User's Guide*
TSTools (see Note #2 below)	Provides administrative tools procedures.	*Oracle8i Time Series User's Guide*

Table 1–1 List of Oracle Supplied Packages

Package Name	Description	Documentation
UTL_COLL	Enables PL/SQL programs to use collection locators to query and update.	Chapter 56
UTL_FILE	Enables your PL/SQL programs to read and write operating system (OS) text files and provides a restricted version of standard OS stream file I/O.	Chapter 57
UTL_HTTP	Enables HTTP callouts from PL/SQL and SQL to access data on the Internet or to call Oracle Web Server Cartridges.	Chapter 58
UTL_PG	Provides functions for converting COBOL numeric data into Oracle numbers and Oracle numbers into COBOL numeric data.	*Oracle Procedural Gateway for APPC User's Guide*
UTL_RAW	Provides SQL functions for RAW datatypes that concat, substr, etc. to and from RAWS.	Chapter 59
UTL_REF	Enables a PL/SQL program to access an object by providing a reference to the object.	Chapter 60
Vir_Pkg (see Note #2 below)	Provides analytical and conversion functions for Visual Information Retrieval.	*Oracle8i Visual Information Retrieval User's Guide and Reference*

Note #1:

The DBMS_STANDARD, STANDARD, and PLITBLM packages contain subprograms to help implement basic language features. Oracle does not recommend that the subprograms be directly called. For this reason, these three supplied packages are not documented in this book.

Note #2:

Time-Series, Image, Visual Information Retrieval, Audio, and Server-Managed Video Cartridge packages are installed in user ORDSYS without public synonyms.

Note #3:

Spatial Cartridge packages are installed in user MDSYS with public synonyms.

Subprograms in Supplemental Packages

The packages listed in the remainder of this chapter are primarily documented in other Oracle books. This section lists the subprograms provided with each of these packages. Please refer to the above table for the cross-reference to the full documentation.

Calendar

Table 1–2 Calendar Package Subprograms

Subprograms	Description
CombineCals	Combines two calendars.
Day	Creates a calendar with a frequency of day.
DeleteExceptions	Deletes from the specified calendar all exceptions that either match a specified date (delExcDate) or are included in a table of dates (delExcTab), and returns the resulting calendar.
DisplayValCal	Displays the results returned by the ValidateCal function.
EqualCals	Checks if two calendars are equal.
GenDateRangeTab	Returns a table of date ranges that represent all of the valid intervals in the input calendar (or from the startDate through endDate parameters).
GetIntervalEnd	Returns the end of the interval that includes the input timestamp.
GetIntervalStart	Returns the start of the interval that includes the input timestamp.
GetOffset	Returns the number of timestamps that the second date is offset from the first.
Hour	Creates a calendar with a frequency of hour.
InsertExceptions	Inserts into the specified calendar all exceptions that either match a specified date (newExcDate) or are included in a table of dates (newExcTab), and returns the resulting calendar.
IntersectCals	Returns the intersection of two calendars.
InvalidTimeStamps Between	Returns a table (ORDTDateTab) containing the invalid timestamps within that range according to the specified calendar.
IsValidCal	Returns 1 if the calendar is valid and 0 if the calendar is not valid.
IsValidDate	Checks whether an input date is valid or invalid according to the specified calendar.

Table 1–2 Calendar Package Subprograms

Subprograms	Description
Minute	Creates a calendar with a frequency of minute.
Month	Creates a calendar with a frequency of month.
NumInvalidTimeStamps Between	Returns the number of invalid timestamps within that range according to the specified calendar.
NumOffExceptions	Returns the number of off-exceptions within that range according to the specified calendar.
NumOnExceptions	Returns the number of on-exceptions within that range according to the specified calendar.
NumTimeStampsBetween	Returns the number of valid timestamps within that range according to the specified calendar.
OffsetDate	Returns the timestamp corresponding to the offset input.
Quarter	Creates a calendar with a frequency of quarter.
Second	Creates a calendar with a frequency of second.
Semi_annual	Creates a calendar with a frequency of semi-annual.
Semi_monthly	Creates a calendar with a frequency of semi-monthly.
SetPrecision	Returns a timestamp that reflects the level of precision implied by the frequency of the specified calendar.
Ten_day	Creates a calendar with a frequency of 10-day.
TimeStampsBetween	Returns a table (ORDTDateTab) containing the valid timestamps within that range according to the specified calendar.
UnionCals	Returns a calendar that is the union of two input calendars.
ValidateCal	Validates a calendar and, if necessary, repairs the calendar and outputs information related to the problems and repairs.
Week	Creates a calendar with a frequency of week.
Year	Creates a calendar with a frequency of year.

SDO_ADMIN

Table 1–3 SDO_ADMIN Package Subprograms

Subprogram	Description
POPULATE_INDEX	Creates a spatial index on a table containing spatial data.
UPDATE_INDEX	Updates the spatial index for a given spatial object instance.

SDO_GEOM

Table 1–4 SDO_GEOM Package Subprograms

Subprogram	Description
RELATE	Determines the topological relationship between two spatial objects.
VALIDATE_GEOMETRY	Evaluates geometric integrity constraints for a spatial object.
WITHIN_DISTANCE	Determines if two spatial objects are within a given Euclidean distance of each other.

SDO_MIGRATE

Table 1–5 SDO_MIGRATE Package Subprograms

Subprogram	Description
TO_81x	Migrates spatial data from a 7.3.3 or 7.3.4 database to an 8.1.x database.

SDO_TUNE

Table 1–6 SDO_TUNE Package Subprograms

Subprogram	Description
ESTIMATE_TILING_LEVEL	Suggests a value for the parameter that determines the resolution of the spatial index.

TimeScale

Table 1–7 TimeScale Package Subprograms

Subprogram	Description
Scaledown Interpolate	Returns a time series in which data values are interpolated between values in the input time series.
ScaledownRepeat	Returns a time series in which data values in the input time series are repeated.
ScaledownSplit	Returns a time series in which data values reflect the division of the data value in the input time series by the number of associated timestamps in the resulting time series.
ScaleupAvg	Returns a time series reflecting the average value of each scaled group of values.
ScaleupAvgX	Returns a time series reflecting the average value of each scaled group of values *plus* the immediately preceding source period.
ScaleupCount	Returns a time series reflecting the count of non-null timestamps in each scaled group of values.
ScaleupFirst	Returns a time series reflecting the first non-null value of each scaled group of values.
ScaleupGMean	Returns a time series reflecting the geometric mean of each scaled group of values.
ScaleupLast	Returns a time series reflecting the last non-null value of each scaled group of values.
ScaleupMax	Returns a time series reflecting the maximum value of each scaled group of values.
ScaleupMin	Returns a time series reflecting the minimum value of each scaled group of values.
ScaleupSum	Returns a time series reflecting the sum of each scaled group of values.
ScaleupSumAnnual	Returns a time series reflecting the sum, expressed as an annual rate, of each scaled group of values.

TimeSeries

Table 1–8 TimeSeries Package Subprograms

Subprogram	Description
Cavg	Returns an ORDTNumSeries with each element containing the cumulative average up to and including the corresponding element in the input ORDTNumSeries.
Cmax	Returns an ORDTNumSeries with each element containing the cumulative maximum up to and including the corresponding element in the input ORDTNumSeries.
Cmin	Returns an ORDTNumSeries with each element containing the cumulative minimum up to and including the corresponding element in the input ORDTNumSeries.
Cprod	Returns an ORDTNumSeries with each element containing the cumulative product of multiplication up to and including the corresponding element in the input ORDTNumSeries.
Csum	Returns an ORDTNumSeries with each element containing the cumulative sum up to and including the corresponding element in the input ORDTNumSeries.
DeriveExceptions	Derives calendar exceptions from a time series, a calendar and a table of dates, or two time series.
Display	Displays various information using DBMS_OUTPUT routines.
DisplayValTS	Displays the results returned by the ValidateTS function.
ExtractCal	Returns a calendar that is the same as the calendar on which the time series is based.
ExtractDate	Returns the date.
ExtractTable	Returns the time series table (ORDTNumTab or ORDTVarchar2Tab) associated with the time series.
ExtractValue	Returns the value stored in a given element in a time series.
Fill	Returns a time series in which values for missing dates are inserted.
First	Returns the first element in a given time series.
FirstN	Returns the first NumValues elements in the given time series.
GetDatedElement	Returns the time series element for a given date.
GetNthElement	Returns the Nth element (element whose position corresponds to target_index) in the specified time series, or within the date range if one is specified.

Table 1–8 TimeSeries Package Subprograms

Subprogram	Description
GetSeries	Returns a time series instance (ORDTNumSeries or ORDTVarchar2Series).
IsValidTS	Returns 1 if the time series is valid and 0 if the time series is invalid.
Lag	Returns a time series that lags or (for negative numeric values) leads the input time series by the appropriate number of timestamps.
Last	Returns the last element in a given time series.
LastN	Returns the last NumValues elements in the time series.
Lead	Returns a time series that leads or (for negative numeric values) lags the input time series by the appropriate number of timestamps.
Mavg	Returns a moving average for the given time series, or for the date range if one is specified.
Msum	Returns a moving sum for the time series, or for the date range if one is specified.
TrimSeries	Returns an ORDT series of the same type with all data outside of the given date range removed.
TSAdd	Given two input time series or a time series and a constant, and optionally starting and ending dates, returns a time series that reflects the addition of the first two parameters.
TSAvg	Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the average of all non-null time series entries.
TSCount	Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the count of all non-null time series entries.
TSDivide	Given two input time series or a time series and a constant, and optionally starting and ending dates, returns a time series that reflects the division of the first parameter by the second parameter.
TSMax	Given an input ORDTNumSeries and optionally starting and ending dates, returns a number corresponding to the highest (maximum) of all non-null time series entries.
TSMaxN	Returns an ORDTNumTab with the specified number (NumValues) of the top (highest) values.

Table 1–8 TimeSeries Package Subprograms

Subprogram	Description
TSMedian	Returns a number corresponding to the median of all non-null time series entries.
TSMin	Returns a number corresponding to the lowest (minimum) of all non-null time series entries.
TSMinN	Returns an ORDTNumTab with the specified number (NumValues) of the bottom (lowest) values.
TSMultiply	Given two input time series or a time series and a constant, and optionally starting and ending dates, returns a time series that reflects the multiplication of the first parameter by the second parameter.
TSProd	Returns a number corresponding to the product (result of multiplication) of all non-null time series entries.
TSStdDev	Returns a number corresponding to the standard deviation of all non-null time series entries.
TSSubtract	Given two input time series or a time series and a constant, and optionally starting and ending dates, returns a time series that reflects the subtraction of the second parameter from the first parameter.
TSSum	Returns a number corresponding to the sum of all non-null time series entries.
TSVariance	Returns a number corresponding to the variance of all non-null time series entries.
ValidateTS	Checks whether a time series is valid, and if the time series is not valid, outputs a diagnostic message and tables with timestamps that are causing the time series to be invalid.

TSTools

Table 1–9 TSTools Package Subprograms

Subprogram	Description
Add_Existing_Column	Adds a column attribute from an existing flat table to a time series.
Add_Integer_Column	Adds an integer column attribute to an ongoing flat time series creation specification.
Add_Number_Column	Adds an integer column attribute to an ongoing flat time series creation specification.
Add_Varchar2_Column	Adds a VARCHAR2 column attribute to an ongoing flat time series creation specification.
Begin_Create_TS_Schema	Initiates the context for creating the schema objects for a time series.
Cancel_Create_TS_Schema	Cancels the creation of a time series schema.
Close_Log	Closes the log file that had been opened by the Open_Log procedure.
Display_Attributes	Displays information about the time series schema being created.
Drop_TS_Schema	Deletes the time series schema definition and all views associated with it.
Drop_TS_Schema_All	Deletes the time series schema definition and all tables, views, indexes, constraints, and triggers associated with it.
End_Create_TS_Schema	Closes the context established by the Begin_Create_TS_Schema procedure and creates all appropriate schema objects.
Get_Flat_Attributes	Retrieves the attributes of a flat time series.
Get_Object_Attributes	Retrieves the attributes of an object-model time series.
Get_Status	Checks whether s time series creation sequence is in progress.
Open_Log	Opens a log file that will contain the data definition language (DDL) statements generated by the Time Series administrative tools procedures.
Set_Flat_Attributes	Sets the attributes of a flat time series.
Set_Object_Attributes	Sets the attributes of an object-model time series.

Table 1–9 TSTools Package Subprograms

Subprogram	Description
Trace_Off	Disables debugging for Time Series cartridge administrative tools procedures.
Trace_On	Enables debugging for Time Series cartridge administrative tools procedures.

UTL_PG

Table 1–10 UTL_PG Package Subprograms

Subprogram	Description
MAKE_NUMBER_TO_RAW_FORMAT	Makes a number_to_raw format conversion specification used to convert an Oracle number of declared precision and scale to a RAW byte-string in the remote host internal format.
MAKE_RAW_TO_NUMBER_FORMAT	Makes a raw_to_number format conversion specification used to convert a RAW byte-string from the remote host internal format into an Oracle number of comparable precision and scale.
NUMBER_TO_RAW	Converts an Oracle number of declared precision and scale to a RAW byte-string in the remote host internal format.
NUMBER_TO_RAW_FORMAT	Converts, according to the number_to_raw conversion format n2rfmt, an Oracle number numval of declared precision and scale to a RAW byte-string in the remote host internal format.
RAW_TO_NUMBER	Converts a RAW byte-string from the remote host internal format into an Oracle number.
RAW_TO_NUMBER_FORMAT	Converts, according to the raw_to_number conversion format r2nfmt, a RAW byte-string rawval in the remote host internal format to an Oracle number.
WMSG	Extracts a warning message specified by wmsgitem from wmsgblk.
WMSGCNT	Tests a wmsgblk to determine how many warnings, if any, are present.

Vir_Pkg

Table 1–11 Vir_Pkg Package Subprograms

Subprogram	Description
`Analyze`	Analyzes an image `BLOB` or `BFILE`, derives information relating to the visual attributes, and creates the image signature.
`Convert`	Converts the image signature to a format usable by the host machine, or converts the signature from the format used for Viisage face-recognition to a signature usable by the `Score` and `Similar` operators.
`Score`	Compares the signature of two images and returns a number representing the weighted sum of the distances for the visual attributes.
`Similar`	Determines whether or not two images match.

2

DBMS_ALERT

The DBMS_ALERT package provides support for the asynchronous notification of database events (alerts). By appropriate use of this package and database triggers, an application can cause itself to be notified whenever values of interest in the database are changed.

For example, suppose a graphics tool is displaying a graph of some data from a database table. The graphics tool can, after reading and graphing the data, wait on a database alert (WAITONE) covering the data just read. The tool automatically wakes up when the data is changed by any other user. All that is required is that a trigger be placed on the database table, which then performs a signal (SIGNAL) whenever the trigger is fired.

Alerts are transaction-based. This means that the waiting session does not get alerted until the transaction signalling the alert commits.There can be any number of concurrent signallers of a given alert, and there can be any number of concurrent waiters on a given alert.

A waiting application is blocked in the database and cannot do any other work.

Note: Because database alerters issue commits, they cannot be used with Oracle Forms. For more information on restrictions on calling stored procedures while Oracle Forms is active, refer to your Oracle Forms documentation.

Security

Security on this package can be controlled by granting EXECUTE on this package to selected users or roles. You might want to write a cover package on top of this one that restricts the alert names used. EXECUTE privilege on this cover package can then be granted rather than on this package.

Constants

```
maxwait constant integer :=  86400000; -- 1000 days
```

The maximum time to wait for an alert (this is essentially forever).

Errors

DBMS_ALERT raises the application error -20000 on error conditions. This table shows the messages and the procedures that can raise them.

Table 2–1 DBMS_ALERT Error Messages

Error Message	Procedure
ORU-10001 lock request error, status: N	SIGNAL
ORU-10015 error: N waiting for pipe status	WAITANY
ORU-10016 error: N sending on pipe 'X'	SIGNAL
ORU-10017 error: N receiving on pipe 'X'	SIGNAL
ORU-10019 error: N on lock request	WAIT
ORU-10020 error: N on lock request	WAITANY
ORU-10021 lock request error; status: N	REGISTER
ORU-10022 lock request error, status: N	SIGNAL
ORU-10023 lock request error; status N	WAITONE
ORU-10024 there are no alerts registered	WAITANY
ORU-10025 lock request error; status N	REGISTER
ORU-10037 attempting to wait on uncommitted signal from same session	WAITONE

Using Alerts

The application can register for multiple events and can then wait for any of them to occur using the WAITANY procedure.

An application can also supply an optional timeout parameter to the WAITONE or WAITANY procedures. A timeout of 0 returns immediately if there is no pending alert.

The signalling session can optionally pass a message that is received by the waiting session.

Alerts can be signalled more often than the corresponding application wait calls. In such cases, the older alerts are discarded. The application always gets the latest alert (based on transaction commit times).

If the application does not require transaction-based alerts, then the DBMS_PIPE package may provide a useful alternative.

See Also: Chapter 28, "DBMS_PIPE"

If the transaction is rolled back after the call to SIGNAL, then no alert occurs.

It is possible to receive an alert, read the data, and find that no data has changed. This is because the data changed after the *prior* alert, but before the data was read for that *prior* alert.

Checking for Alerts

Usually, Oracle is event-driven; this means that there are no polling loops. There are two cases where polling loops can occur:

- Shared mode. If your database is running in shared mode, a polling loop is required to check for alerts from another instance. The polling loop defaults to one second and can be set by the SET_DEFAULTS procedure.
- WAITANY procedure. If you use the WAITANY procedure, and if a signalling session does 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. The polling loop begins at a one second interval and exponentially backs off to 30-second intervals.

Summary of Subprograms

Table 2–2 DBMS_ALERT Package Subprograms

Subprogram	Description
REGISTER procedure on page 2-4	Receives messages from an alert.
REMOVE procedure on page 2-5	Disables notification from an alert.
REMOVEALL procedure on page 2-5	Removes all alerts for this session from the registration list.
SET_DEFAULTS procedure on page 2-6	Sets the polling interval.
SIGNAL procedure on page 2-6	Signals an alert (send message to registered sessions).
WAITANY procedure on page 2-7	Waits timeout seconds to receive alert message from an alert registered for session.
WAITONE procedure on page 2-8	Waits timeout seconds to receive message from named alert.

REGISTER procedure

This procedure lets a session register interest in an alert. The name of the alert is the IN parameter. A session can register interest in an unlimited number of alerts. Alerts should be deregistered when the session no longer has any interest, by calling REMOVE.

Syntax

```
DBMS_ALERT.REGISTER (
   name  IN  VARCHAR2);
```

Parameters

Table 2–3 REGISTER Procedure Parameters

Parameter	Description
name	Name of the alert in which this session is interested.

Caution: Alert names beginning with 'ORA$' are reserved for use for products provided by Oracle Corporation. Names must be 30 bytes or less. The name is case-insensitive.

REMOVE procedure

This procedure enables a session that is no longer interested in an alert to remove that alert from its registration list. Removing an alert reduces the amount of work done by signalers of the alert.

Removing alerts is important because it reduces the amount of work done by signalers of the alert. If a session dies without removing the alert, that alert is eventually (but not immediately) cleaned up.

Syntax

```
DBMS_ALERT.REMOVE (
   name  IN  VARCHAR2);
```

Parameters

Table 2–4 REMOVE Procedure Parameters

Parameter	Description
name	Name of the alert (case-insensitive) to be removed from registration list.

REMOVEALL procedure

This procedure removes all alerts for this session from the registration list. You should do this when the session is no longer interested in any alerts.

This procedure is called automatically upon first reference to this package during a session. Therefore, no alerts from prior sessions which may have terminated abnormally can affect this session.

This procedure always performs a commit.

Syntax

```
DBMS_ALERT.REMOVEALL;
```

Parameters

None.

SET_DEFAULTS procedure

In case a polling loop is required, use the SET_DEFAULTS procedure to set the polling interval.

Syntax

```
DBMS_ALERT.SET_DEFAULTS (
   polling_interval  IN  NUMBER);
```

Parameters

Table 2–5 SET_DEFAULTS Procedure Parameters

Parameter	Description
polling_interval	Time, in seconds, to sleep between polls. The default interval is five seconds.

SIGNAL procedure

This procedure signals an alert. The effect of the SIGNAL call only occurs when the transaction in which it is made commits. If the transaction rolls back, then SIGNAL has no effect.

All sessions that have registered interest in this alert are notified. If the interested sessions are currently waiting, then they are awakened. If the interested sessions are not currently waiting, then they are notified the next time they do a wait call.

Multiple sessions can concurrently perform signals on the same alert. Each session, as it signals the alert, blocks all other concurrent sessions until it commits. This has the effect of serializing the transactions.

Syntax

```
DBMS_ALERT.SIGNAL (
   name     IN  VARCHAR2,
   message  IN  VARCHAR2);
```

Parameters

Table 2–6 SIGNAL Procedure Parameters

Parameter	Description
name	Name of the alert to signal.
message	Message, of 1800 bytes or less, to associate with this alert. This message is passed to the waiting session. The waiting session might be able to avoid reading the database after the alert occurs by using the information in the message.

WAITANY procedure

Call WAITANY to wait for an alert to occur for any of the alerts for which the current session is registered. The same session that waits for the alert may also first signal the alert. In this case remember to commit after the signal and before the wait; otherwise, DBMS_LOCK.REQUEST (which is called by DBMS_ALERT) returns status 4.

Syntax

```
DBMS_ALERT.WAITANY (
   name      OUT  VARCHAR2,
   message   OUT  VARCHAR2,
   status    OUT  INTEGER,
   timeout   IN   NUMBER DEFAULT MAXWAIT);
```

Parameters

Table 2–7 WAITANY Procedure Parameters

Parameter	Description
name	Returns the name of the alert that occurred.
message	Returns the message associated with the alert. This is the message provided by the SIGNAL call. If multiple signals on this alert occurred before WAITANY, then the message corresponds to the most recent SIGNAL call. Messages from prior SIGNAL calls are discarded.
status	Values returned: 0 - alert occurred 1 - time-out occurred
timeout	Maximum time to wait for an alert. If no alert occurs before timeout seconds, then this returns a status of 1.

Errors

```
-20000, ORU-10024: there are no alerts registered.
```

Cause: You must register an alert before waiting.

WAITONE procedure

This procedure waits for a specific alert to occur. A session that is the first to signal an alert can also wait for the alert in a subsequent transaction. In this case, remember to commit after the signal and before the wait; otherwise, DBMS_LOCK.REQUEST (which is called by DBMS_ALERT) returns status 4.

Syntax

```
DBMS_ALERT.WAITONE (
   name      IN   VARCHAR2,
   message   OUT  VARCHAR2,
   status    OUT  INTEGER,
   timeout   IN   NUMBER DEFAULT MAXWAIT);
```

Parameters

Table 2–8 WAITONE Procedure Parameters

Parameter	Description
name	Name of the alert to wait for.
message	Returns the message associated with the alert. This is the message provided by the SIGNAL call. If multiple signals on this alert occurred before WAITONE, then the message corresponds to the most recent SIGNAL call. Messages from prior SIGNAL calls are discarded.
status	Values returned: 0 - alert occurred 1 - time-out occurred
timeout	Maximum time to wait for an alert. If the named alert does not occurs before timeout seconds, this returns a status of 1.

Example

Suppose you want to graph average salaries by department, for all employees. Your application needs to know whenever EMP is changed. Your application would look similar to this code:

```
DBMS_ALERT.REGISTER('emp_table_alert');
    readagain:
   /* ... read the emp table and graph it */
      DBMS_ALERT.WAITONE('emp_table_alert', :message, :status);
      if status = 0 then goto readagain; else
      /* ... error condition */
```

The EMP table would have a trigger similar to this:

```
CREATE TRIGGER emptrig AFTER INSERT OR UPDATE OR DELETE ON emp
    BEGIN
      DBMS_ALERT.SIGNAL('emp_table_alert', 'message_text');
   END;
```

When the application is no longer interested in the alert, it makes this request:

```
DBMS_ALERT.REMOVE('emp_table_alert');
```

This reduces the amount of work required by the alert signaller. If a session exits (or dies) while registered alerts exist, then they are eventually cleaned up by future users of this package.

The above example guarantees that the application always sees the latest data, although it may not see every intermediate value.

3

DBMS_APPLICATION_INFO

Application developers can use the DBMS_APPLICATION_INFO package with Oracle Trace and the SQL trace facility to record names of executing modules or transactions in the database for later use when tracking the performance of various modules.

Registering the application allows system administrators and performance tuning specialists to track performance by module. System administrators can also use this information to track resource use by module. When an application registers with the database, its name and actions are recorded in the V$SESSION and V$SQLAREA views.

Your applications should set the name of the module and name of the action automatically each time a user enters that module. The module name could be the name of a form in an Oracle Forms application, or the name of the code segment in an Oracle Precompilers application. The action name should usually be the name or description of the current transaction within a module.

If you want to gather you own statistics based on module, then you can implement a wrapper around this package by writing a version of this package in another schema that first gathers statistics and then calls the SYS version of the package. The public synonym for DBMS_APPLICATION_INFO can then be changed to point to the DBA's version of the package.

Note: The public synonym for DBMS_APPLICATION_INFO is not dropped before creation, in order to allow users to redirect the public synonym to point to their own package.

Privileges

Before using this package, you must run the DBMSUTL.SQL script to create the DBMS_APPLICATION_INFO package.

Summary of Subprograms

Table 3–1 DBMS_APPLICATION_INFO Package Subprograms

Subprogram	Description
SET_MODULE procedure on page 3-2	Sets the name of the module that is currently running to a new module.
SET_ACTION procedure on page 3-3	Sets the name of the current action within the current module.
READ_MODULE procedure on page 3-4	Reads the values of the module and action fields of the current session.
SET_CLIENT_INFO procedure on page 3-5	Sets the client info field of the session.
READ_CLIENT_INFO procedure on page 3-6	Reads the value of the client_info field of the current session.
SET_SESSION_LONGOPS procedure on page 3-6	Sets a row in the V$SESSION_LONGOP table.

SET_MODULE procedure

This procedure sets the name of the current application or module. The module name should be the name of the procedure (if using stored procedures), or the name of the application. The action name should describe the action performed.

Syntax

```
DBMS_APPLICATION_INFO.SET_MODULE (
   module_name IN VARCHAR2,
   action_name IN VARCHAR2);
```

Parameters

Table 3–2 SET_MODULE Procedure Parameters

Parameter	Description
module_name	Name of module that is currently running. When the current module terminates, call this procedure with the name of the new module if there is one, or NULL if there is not. Names longer than 48 bytes are truncated.
action_name	Name of current action within the current module. If you do not want to specify an action, this value should be NULL. Names longer than 32 bytes are truncated.

Example

```
CREATE PROCEDURE add_employee(
   name VARCHAR2(20),
   salary NUMBER(7,2),
   manager NUMBER,
   title VARCHAR2(9),
   commission NUMBER(7,2),
   department NUMBER(2)) AS
BEGIN

   DBMS_APPLICATION_INFO.SET_MODULE(
      module_name => 'add_employee',
      action_name => 'insert into emp');
   INSERT INTO emp
      (ename, empno, sal, mgr, job, hiredate, comm, deptno)
      VALUES (name, next.emp_seq, manager, title, SYSDATE,
              commission, department);
   DBMS_APPLICATION_INFO.SET_MODULE('','');

END;
```

SET_ACTION procedure

This procedure sets the name of the current action within the current module. The action name should be descriptive text about the current action being performed. You should probably set the action name before the start of every transaction.

Syntax

```
DBMS_APPLICATION_INFO.SET_ACTION (
   action_name IN VARCHAR2);
```

Parameters

Table 3–3 SET_ACTION Procedure Parameters

Parameter	Description
action_name	The name of the current action within the current module. When the current action terminates, call this procedure with the name of the next action if there is one, or NULL if there is not. Names longer than 32 bytes are truncated.

Usage Notes

Set the transaction name to NULL after the transaction completes, so that subsequent transactions are logged correctly. If you do not set the transaction name to NULL, then subsequent transactions may be logged with the previous transaction's name.

Example

The following is an example of a transaction that uses the registration procedure:

```
CREATE OR REPLACE PROCEDURE bal_tran (amt IN NUMBER(7,2)) AS
BEGIN

-- balance transfer transaction

   DBMS_APPLICATION_INFO.SET_ACTION(
      action_name => 'transfer from chk to sav');
   UPDATE chk SET bal = bal + :amt
      WHERE acct# = :acct;
   UPDATE sav SET bal = bal - :amt
      WHERE acct# = :acct;
   COMMIT;
   DBMS_APPLICATION_INFO.SET_ACTION('');

END;
```

READ_MODULE procedure

This procedure reads the values of the module and action fields of the current session.

Syntax

```
DBMS_APPLICATION_INFO.READ_MODULE (
   module_name OUT VARCHAR2,
   action_name OUT VARCHAR2);
```

Parameters

Table 3–4 READ_MODULE Procedure Parameters

Parameter	Description
module_name	Last value that the module name was set to by calling SET_MODULE.
action_name	Last value that the action name was set to by calling SET_ACTION or SET_MODULE.

Usage Notes

Module and action names for a registered application can be retrieved by querying V$SQLAREA or by calling the READ_MODULE procedure. Client information can be retrieved by querying the V$SESSION view, or by calling the READ_CLIENT_INFO procedure.

Example

The following sample query illustrates the use of the MODULE and ACTION column of the V$SQLAREA.

```
SELECT sql_text, disk_reads, module, action
FROM v$sqlarea
WHERE module = 'add_employee';

SQL_TEXT DISK_READS MODULE ACTION
------------------- ---------- ------------------ ---------------
INSERT INTO emp 1 add_employee insert into emp
(ename, empno, sal, mgr, job, hiredate, comm, deptno)
VALUES
(name, next.emp_seq, manager, title, SYSDATE, commission, department)

1 row selected.
```

SET_CLIENT_INFO procedure

This procedure supplies additional information about the client application.

Syntax

```
DBMS_APPLICATION_INFO.SET_CLIENT_INFO (
   client_info IN VARCHAR2);
```

Parameters

Table 3–5 SET_CLIENT_INFO Procedure Parameters

Parameter	Description
client_info	Supplies any additional information about the client application. This information is stored in the V$SESSIONS view. Information exceeding 64 bytes is truncated.

READ_CLIENT_INFO procedure

This procedure reads the value of the client_info field of the current session.

Syntax

```
DBMS_APPLICATION_INFO.READ_CLIENT_INFO (
   client_info OUT VARCHAR2);
```

Parameters

Table 3–6 READ_CLIENT_INFO Procedure Parameters

Parameter	Description
client_info	Last client information value supplied to the SET_CLIENT_INFO procedure.

SET_SESSION_LONGOPS procedure

This procedure sets a row in the V$SESSION_LONGOP table. This is a table which is customarily used to indicate the on-going progress of a long running operation. Some Oracle functions, such as Parallel Query and Server Managed Recovery, use rows in this table to indicate the status of, for example, a database backup.

Applications may use this function to advertise information about application-specific long running tasks.

Syntax

```
DBMS_APPLICATION_INFO.SET_SESSION_LONGOPS (
   rindex      IN OUT PLS_INTEGER,
   slno        IN OUT PLS_INTEGER,
   op_name     IN     VARCHAR2    DEFAULT NULL,
   target      IN     PLS_INTEGER DEFAULT 0,
   context     IN     PLS_INTEGER DEFAULT 0,
   sofar       IN     NUMBER      DEFAULT 0,
   totalwork   IN     NUMBER      DEFAULT 0,
   target_desc IN     VARCHAR2    DEFAULT 'unknown target',
   units       IN     VARCHAR2    DEFAULT NULL)

set_session_longops_nohint constant pls_integer := -1;
```

Pragmas

```
pragma TIMESTAMP('1998-03-12:12:00:00');
```

Parameters

Table 3–7 SET_SESSION_LONGOPS Procedure Parameters

Parameter	Description
rindex	A token which represents the v$session_longops row to update. Set this to set_session_longops_nohint to start a new row. Use the returned value from the prior call to reuse a row.
slno	Saves information across calls to set_session_longops: It is for internal use and should not be modified by the caller.
op_name	Specifies the name of the long running task. It appears as the OPNAME column of v$session_longops. The maximum length is 64 bytes.
target	Specifies the object that is being worked on during the long running operation. For example, it could be a table ID that is being sorted. It appears as the TARGET column of v$session_longops.
context	Any number the client wants to store. It appears in the CONTEXT column of v$session_longops.
sofar	Any number the client wants to store. It appears in the SOFAR column of v$session_longops. This is typically the amount of work which has been done so far.

Table 3–7 SET_SESSION_LONGOPS Procedure Parameters

Parameter	Description
totalwork	Any number the client wants to store. It appears in the TOTALWORK column of v$session_longops. This is typically an estimate of the total amount of work needed to be done in this long running operation.
target_desc	Specifies the description of the object being manipulated in this long operation. This provides a caption for the target parameter. This value appears in the TARGET_DESC field of v$session_longops. The maximum length is 32 bytes.
units	Specifies the units in which sofar and totalwork are being represented. It appears as the UNITS field of v$session_longops. The maximum length is 32 bytes.

4

DBMS_AQ

The DBMS_AQ package provides an interface to Oracle's Advanced Queuing.

See Also: *Oracle8i Application Developer's Guide - Advanced Queuing* contains detailed information about DBMS_AQ.

Java Classes

Java interfaces are available for DBMS_AQ and DBMS_AQADM. The java interfaces are provided in the $ORACLE_HOME/rdbms/jlib/aqapi.jar. In this release, these Java API are available only for queues with RAW type payloads. Users are required to have EXECUTE privileges on the DBMS_AQIN package to use these interfaces.

Enumerated Constants

When using enumerated constants such as BROWSE, LOCKED, or REMOVE, the PL/SQL constants must be specified with the scope of the packages defining it. All types associated with the operational interfaces have to be prepended with DBMS_AQ. For example:

```
DBMS_AQ.BROWSE
```

Table 4–1 Enumerated Constants

Parameter	Options
visibility	IMMEDIATE, ON_COMMIT
dequeue mode	BROWSE, LOCKED, REMOVE, REMOVE_NODATA
navigation	FIRST_MESSAGE, NEXT_MESSAGE, NEXT_TRANSACTION
state	WAITING, READY, PROCESSED, EXPIRED
sequence_deviation	BEFORE, TOP
wait	FOREVER, NO_WAIT
delay	NO_DELAY
expiration	NEVER

Data Structures

The following data structures are used in both DBMS_AQ and DBMS_AQADM:

- Object Name
- Type Name
- Agent
- Enqueue Options Type
- Dequeue Options Type
- Message Properties Type

- AQ Recipient List Type
- AQ Agent List Type
- AQ Subscriber List Type

Object Name

This names database objects. This naming convention applies to queues, queue tables, agent names, and object types.

Syntax

```
object_name := VARCHAR2;
object_name := [<schema_name>.]<name>;
```

Usage Names for objects are specified by an optional schema name and a name. If the schema name is not specified, then the current schema is assumed. The name must follow object name guidelines in the *Oracle8i SQL Reference* with regard to reserved characters. The schema name, agent name, and the object type name can each be up to 30 bytes long. However, queue names and queue table names have a maximum of 24 bytes.

Type Name

This defines queue types.

Syntax

```
type_name := VARCHAR2;
type_name := <object_type> | "RAW";
```

Usage

Table 4–2 Type Name

Parameter	Description
<object_types>	Maximum number of attributes in the object type is limited to 900. See Also: *Oracle8i Concepts*

Table 4–2 Type Name

Parameter	Description
"RAW"	To store payload of type RAW, AQ creates a queue table with a LOB column as the payload repository. The theoretical maximum size of the message payload is the maximum amount of data that can be stored in a LOB column. However, the maximum size of the payload is determined by which programmatic environment you use to access AQ. For PL/SQL, Java and precompilers the limit is 32K; for the OCI the limit is 4G. Because the PL/SQL enqueue and dequeue interfaces accept RAW buffers as the payload parameters you will be limited to 32K bytes. In OCI, the maximum size of your RAW data will be limited to the maximum amount of contiguous memory (as an OCIRaw is simply an array of bytes) that the OCI Object Cache can allocate. Typically, this will be at least 32K bytes and much larger in many cases. Because LOB columns are used for storing RAW payload, the AQ administrator can choose the LOB tablespace and configure the LOB storage by constructing a LOB storage string in the storage_clause parameter during queue table creation time.

Agent

This identifies a producer or a consumer of a message.

Syntax

```
TYPE sys.aq$_agent IS OBJECT (
   name      VARCHAR2(30),
   address   VARCHAR2(1024),
   protocol  NUMBER);
```

Usage

Table 4–3 Agent

Parameter	Description
name	Name of a producer or consumer of a message. The name must follow object name guidelines in the *Oracle8i SQL Reference* with regard to reserved characters.
address	Protocol-specific address of the recipient. If the protocol is 0 (default), then the address is of the form [schema.]queue[@dblink].

Table 4–3 Agent

Parameter	Description
protocol	Protocol to interpret the address and propagate the message. The default is 0.

Enqueue Options Type

This specifies the options available for the enqueue operation.

Syntax

```
TYPE enqueue_options_t IS RECORD (
   visibility            BINARY_INTEGER DEFAULT ON_COMMIT,
   relative_msgid        RAW(16)        DEFAULT NULL,
   sequence_deviation    BINARY_INTEGER DEFAULT NULL);
```

Usage

Table 4–4 Enqueue Options Type

Parameter	Description
visibility	Specifies the transactional behavior of the enqueue request. ON_COMMIT: The enqueue is part of the current transaction. The operation is complete when the transaction commits. This is the default case. IMMEDIATE: The enqueue is not part of the current transaction. The operation constitutes a transaction on its own. This is the only value allowed when enqueuing to a non-persistent queue.
relative_msgid	Specifies the message identifier of the message which is referenced in the sequence deviation operation. This field is valid if, and only if, BEFORE is specified in sequence_deviation. This parameter is ignored if sequence deviation is not specified.
sequence_deviation	Specifies whether the message being enqueued should be dequeued before other message(s) already in the queue. BEFORE: The message is enqueued ahead of the message specified by *relative_msgid.* TOP: The message is enqueued ahead of any other messages. NULL: Default

Dequeue Options Type

This specifies the options available for the dequeue operation.

Syntax

```
TYPE dequeue_options_t IS RECORD (
   consumer_name    VARCHAR2(30)   DEFAULT NULL,
   dequeue_mode     BINARY_INTEGER DEFAULT REMOVE,
   navigation       BINARY_INTEGER DEFAULT NEXT_MESSAGE,
   visibility       BINARY_INTEGER DEFAULT ON_COMMIT,
   wait             BINARY_INTEGER DEFAULT FOREVER,
   msgid            RAW(16)        DEFAULT NULL,
   correlation      VARCHAR2(128)  DEFAULT NULL);
```

Usage

Table 4–5 Dequeue Options Type

Parameter	Description
consumer_name	Name of the consumer. Only those messages matching the consumer name are accessed. If a queue is not set up for multiple consumers, then this field should be set to NULL.
dequeue_mode	Specifies the locking behavior associated with the dequeue. BROWSE: Read the message without acquiring any lock on the message. This is equivalent to a select statement. LOCKED: Read and obtain a write lock on the message. The lock lasts for the duration of the transaction. This is equivalent to a select for update statement. REMOVE: Read the message and update or delete it. This is the default. The message can be retained in the queue table based on the retention properties. REMOVE_NODATA: Mark the message as updated or deleted. The message can be retained in the queue table based on the retention properties.

Table 4–5 Dequeue Options Type

Parameter	Description
navigation	Specifies the position of the message that will be retrieved. First, the position is determined. Second, the search criterion is applied. Finally, the message is retrieved. NEXT_MESSAGE: Retrieve the next message which is available and matches the search criteria. If the previous message belongs to a message group, then AQ retrieves the next available message which matches the search criteria and belongs to the message group. This is the default. NEXT_TRANSACTION: Skip the remainder of the current transaction group (if any) and retrieve the first message of the next transaction group. This option can only be used if message grouping is enabled for the current queue. FIRST_MESSAGE: Retrieves the first message which is available and matches the search criteria. This resets the position to the beginning of the queue.
visibility	Specifies whether the new message is dequeued as part of the current transaction.The visibility parameter is ignored when using the BROWSE mode. ON_COMMIT: The dequeue will be part of the current transaction. This is the default case. IMMEDIATE: The dequeued message is not part of the current transaction. It constitutes a transaction on its own.
wait	Specifies the wait time if there is currently no message available which matches the search criteria. FOREVER: wait forever. This is the default. NO_WAIT: do not wait number: wait time in seconds
msgid	Specifies the message identifier of the message to be dequeued.
correlation	Specifies the correlation identifier of the message to be dequeued. Special pattern matching characters, such as the percent sign (%) and the underscore (_) can be used. If more than one message satisfies the pattern, then the order of dequeuing is undetermined.

Message Properties Type

This describes the information that is used by AQ to manage individual messages. These are set at enqueue time, and their values are returned at dequeue time.

Syntax

```
TYPE message_properties_t IS RECORD (
   priority               BINARY_INTEGER DEFAULT 1,
   delay                  BINARY_INTEGER DEFAULT NO_DELAY,
   expiration             BINARY_INTEGER DEFAULT NEVER,
   correlation            VARCHAR2(128)  DEFAULT NULL,
   attempts               BINARY_INTEGER,
   recipient_list         aq$_recipient_list_t,
   exception_queue        VARCHAR2(51)   DEFAULT NULL,
   enqueue_time           DATE,
   state                  BINARY_INTEGER,
   sender_id              aq$_agent      DEFAULT NULL,
   original_msgid         RAW(16)        DEFAULT NULL);

TYPE aq$_recipient_list_t IS TABLE OF sys.aq$_agent
   INDEX BY BINARY_INTEGER;
```

Usage

Table 4–6 Message Properties Type

Parameter	Description
priority	Specifies / returns the priority of the message. A smaller number indicates higher priority. The priority can be any number, including negative numbers.
delay	Specifies / returns the delay of the enqueued message. The delay represents the number of seconds after which a message is available for dequeuing. Dequeuing by msgid overrides the delay specification. A message enqueued with delay set will be in the WAITING state, when the delay expires the messages goes to the READY state. DELAY processing requires the queue monitor to be started. Note that delay is set by the producer who enqueues the message. NO_DELAY: the message is available for immediate dequeuing. number: the number of seconds to delay the message.

Table 4–6 Message Properties Type

Parameter	Description
expiration	Specifies / returns the expiration of the message. It determines, in seconds, the duration the message is available for dequeuing. This parameter is an offset from the delay. Expiration processing requires the queue monitor to be running. NEVER: message does not expire. number: number of seconds message remains in READY state. If the message is not dequeued before it expires, then it is moved to the exception queue in the EXPIRED state.
correlation	Returns the identification supplied by the producer for a message at enqueuing.
attempts	Returns the number of attempts that have been made to dequeue this message. This parameter cannot be set at enqueue time.
recipient_list	For type definition, see the "Agent" on page 4-4. This parameter is only valid for queues which allow multiple consumers. The default recipients are the queue subscribers. This parameter is not returned to a consumer at dequeue time.
exception_queue	Specifies / returns the name of the queue to which the message is moved if it cannot be processed successfully. Messages are moved in two cases: The number of unsuccessful dequeue attempts has exceeded *max_retries* or the message has expired. All messages in the exception queue are in the EXPIRED state. The default is the exception queue associated with the queue table. If the exception queue specified does not exist at the time of the move, then the message is moved to the default exception queue associated with the queue table, and a warning is logged in the alert file. If the default exception queue is used, then the parameter returns a NULL value at dequeue time.
enqueue_time	Returns the time the message was enqueued. This value is determined by the system and cannot be set by the user. This parameter can not be set at enqueue time.

Table 4–6 Message Properties Type

Parameter	Description
state	Returns the state of the message at the time of the dequeue. This parameter can not be set at enqueue time. 0: The message is ready to be processed. 1: The message delay has not yet been reached. 2: The message has been processed and is retained. 3: The message has been moved to the exception queue.
sender_id	Specifies / returns the application-specified sender identification. DEFAULT: NULL
original_msgid	This parameter is used by Oracle AQ for propagating messages. DEFAULT: NULL

AQ Recipient List Type

This identifies the list of agents that will receive the message. This structure is used only when the queue is enabled for multiple dequeues.

Syntax

```
TYPE aq$_recipient_list_t IS TABLE OF sys.aq$_agent
   INDEX BY BINARY_INTEGER;
```

AQ Agent List Type

This identifies the list of agents for DBMS_AQ.LISTEN to listen for.

Syntax

```
TYPE aq$_agent_list_t IS TABLE of sys.aq$_agent
   INDEX BY BINARY INTEGER;
```

AQ Subscriber List Type

This identifies the list of subscribers that subscribe to this queue.

Syntax

```
TYPE aq$_subscriber_list_t IS TABLE OF sys.aq$_agent
   INDEX BY BINARY_INTEGER;
```

Summary of Subprograms

Table 4–7 DBMS_AQ Package Subprograms

Subprograms	Description
ENQUEUE procedure on page 4-11	Adds a message to the specified queue.
DEQUEUE procedure on page 4-13	Dequeues a message from the specified queue.
LISTEN procedure on page 4-15	Listen on one or more queues on behalf of a list of agents.

ENQUEUE procedure

This procedure adds a message to the specified queue.

Syntax

```
DBMS_AQ.ENQUEUE (
   queue_name          IN      VARCHAR2,
   enqueue_options     IN      enqueue_options_t,
   message_properties  IN      message_properties_t,
   payload             IN      "<type_name>",
   msgid               OUT     RAW);
```

Parameters

Table 4–8 ENQUEUE Procedure Parameters

Parameter	Description
queue_name	Specifies the name of the queue to which this message should be enqueued. The queue cannot be an exception queue.
enqueue_options	See "Enqueue Options Type" on page 4-5.
message_properties	See "Message Properties Type" on page 4-7.

Table 4–8 ENQUEUE Procedure Parameters

Parameter	Description
payload	Not interpreted by Oracle AQ. The payload must be specified according to the specification in the associated queue table. NULL is an acceptable parameter. For the definition of <type_name> please refer to "Type Name" on page 4-3.
msgid	System generated identification of the message. This is a globally unique identifier that can be used to identify the message at dequeue time.

Usage Notes

Using Sequence Deviation The sequence_deviation parameter in enqueue_options can be used to change the order of processing between two messages. The identity of the other message, if any, is specified by the enqueue_options parameter relative_msgid. The relationship is identified by the sequence_deviation parameter.

Specifying sequence_deviation for a message introduces some restrictions for the delay and priority values that can be specified for this message. The delay of this message must be less than or equal to the delay of the message before which this message is to be enqueued. The priority of this message must be greater than or equal to the priority of the message before which this message is to be enqueued.

Sending a Message when there are No Recipients If a message is enqueued to a multi-consumer queue with no recipient, and if the queue has no subscribers (or rule-based subscribers that match this message), then the Oracle error ORA 24033 is raised. This is a warning that the message will be discarded because there are no recipients or subscribers to whom it can be delivered.

DEQUEUE procedure

This procedure dequeues a message from the specified queue.

Syntax

```
DBMS_AQ.DEQUEUE (
   queue_name           IN      VARCHAR2,
   dequeue_options      IN      dequeue_options_t,
   message_properties   OUT     message_properties_t,
   payload              OUT     "<type_name>",
   msgid                OUT     RAW);
```

Parameters

Table 4–9 DEQUEUE Procedure Parameters

Parameter	Description
queue_name	Specifies the name of the queue.
dequeue_options	See "Dequeue Options Type" on page 4-6.
message_properties	See "Message Properties Type" on page 4-7.
payload	Not interpreted by Oracle AQ. The payload must be specified according to the specification in the associated queue table. For the definition of <type_name> please refer to "Type Name" on page 4-3.
msgid	System generated identification of the message.

Usage Notes

Search criteria and dequeue order for messages The search criteria for messages to be dequeued is determined by the `consumer_name`, `msgid` and `correlation` parameters in `dequeue_options`. `Msgid` uniquely identifies the message to be dequeued. Correlation identifiers are application-defined identifiers that are not interpreted by AQ.

Only messages in the `READY` state are dequeued unless `msgid` is specified.

The dequeue order is determined by the values specified at the time the queue table is created unless overridden by the `msgid` and correlation ID in `dequeue_options`.

The database consistent read mechanism is applicable for queue operations. For example, a BROWSE call may not see a message that is enqueued after the beginning of the browsing transaction.

Navigating through a queue The default NAVIGATION parameter during dequeue is NEXT_MESSAGE. This means that subsequent dequeues will retrieve the messages from the queue based on the snapshot obtained in the first dequeue. In particular, a message that is enqueued after the first dequeue command will be processed only after processing all the remaining messages in the queue. This is usually sufficient when all the messages have already been enqueued into the queue, or when the queue does not have a priority-based ordering. However, applications must use the FIRST_MESSAGE navigation option when the first message in the queue needs to be processed by every dequeue command. This usually becomes necessary when a higher priority message arrives in the queue while messages already-enqueued are being processed.

Note: It may also be more efficient to use the FIRST_MESSAGE navigation option when there are messages being concurrently enqueued. If the FIRST_MESSAGE option is not specified, then AQ continually generates the snapshot as of the first dequeue command, leading to poor performance. If the FIRST_MESSAGE option is specified, then AQ uses a new snapshot for every dequeue command.

Dequeue by Message Grouping Messages enqueued in the same transaction into a queue that has been enabled for message grouping will form a group. If only one message is enqueued in the transaction, then this will effectively form a group of one message. There is no upper limit to the number of messages that can be grouped in a single transaction.

In queues that have not been enabled for message grouping, a dequeue in LOCKED or REMOVE mode locks only a single message. By contrast, a dequeue operation that seeks to dequeue a message that is part of a group will lock the entire group. This is useful when all the messages in a group need to be processed as an atomic unit.

When all the messages in a group have been dequeued, the dequeue returns an error indicating that all messages in the group have been processed. The application can then use the NEXT_TRANSACTION to start dequeuing messages from the next available group. In the event that no groups are available, the dequeue will time-out after the specified WAIT period.

LISTEN procedure

This procedure listens on one or more queues on behalf of a list of agents. The 'address' field of the agent indicates the queue the agent wants to monitor. Only local queues are supported as addresses. Protocol is reserved for future use.

If agent-address is a multi-consumer queue, then agent-name is mandatory. For single-consumer queues, agent-name must not be specified.

This is a blocking call that returns when there is a message ready for consumption for an agent in the list. If there are no messages found when the wait time expires, then an error is raised.

Syntax

```
DBMS_AQ.LISTEN (
   agent_list IN    aq$_agent_list_t,
   wait       IN    BINARY_INTEGER DEFAULT DBMS_AQ.FOREVER,
   agent      OUT   sys.aq$_agent);

TYPE aq$_agent_list_t IS TABLE of aq$_agent INDEXED BY BINARY_INTEGER;
```

Parameters

Table 4–10 LISTEN Procedure Parameters

Parameter	Description
agent_list	List of agents for which to 'listen'.
wait	Time-out for the listen call (in seconds). By default, the call will block forever.
agent	Agent with a message available for consumption.

Usage Notes

This procedure takes a list of agents as an argument. You specify the queue to be monitored in the address field of each agent listed. You also must specify the name of the agent when monitoring multiconsumer queues. For single-consumer queues, an agent name must not be specified. Only local queues are supported as addresses. Protocol is reserved for future use.

This is a blocking call that returns when there is a message ready for consumption for an agent in the list. If there are messages for more than one agent, then only the

first agent listed is returned. If there are no messages found when the wait time expires, then an error is raised.

A successful return from the listen call is only an indication that there is a message for one of the listed agents in one the specified queues. The interested agent must still dequeue the relevant message.

Note that you cannot call listen on non-persistent queues.

5

DBMS_AQADM

The DBMS_AQADM package provides procedures to manage Advanced Queuing configuration and administration information.

> **See Also:** *Oracle8i Application Developer's Guide - Advanced Queuing* contains detailed information about DBMS_AQADM.

Enumerated Constants

When using enumerated constants, such as INFINITE,TRANSACTIONAL, or NORMAL_QUEUE, the symbol must be specified with the scope of the packages defining it. All types associated with the administrative interfaces must be prepended with DBMS_AQADM. For example:

```
DBMS_AQADM.NORMAL_QUEUE
```

Table 5–1 Enumerated Types in the Administrative Interface

Parameter	Options
retention	0,1,2...INFINITE
message_grouping	TRANSACTIONAL, NONE
queue_type	NORMAL_QUEUE, EXCEPTION_QUEUE, NON_PERSISTENT_QUEUE

See Also: For more information on the Java classes and data structures used in both DBMS_AQ and DBMS_AQADM, see Chapter 4, "DBMS_AQ"

Summary of Subprograms

Table 5–2 DBMS_AQADM Package Subprograms

Subprogram	Description
CREATE_QUEUE_TABLE procedure on page 5-4	Creates a queue table for messages of a pre-defined type.
ALTER_QUEUE_TABLE procedure on page 5-7	Alters an existing queue table.
DROP_QUEUE_TABLE procedure on page 5-8	Drops an existing queue table.
CREATE_QUEUE procedure on page 5-9	Creates a queue in the specified queue table.
CREATE_NP_QUEUE procedure on page 5-11	Creates a non-persistent RAW queue.
ALTER_QUEUE procedure on page 5-12	Alters existing properties of a queue.

Table 5–2 DBMS_AQADM Package Subprograms

Subprogram	Description
DROP_QUEUE procedure on page 5-13	Drops an existing queue.
START_QUEUE procedure on page 5-14	Enables the specified queue for enqueuing and / or dequeueing.
STOP_QUEUE procedure on page 5-15	Disables enqueuing and / or dequeuing on the specified queue.
GRANT_SYSTEM_PRIVILEGE procedure on page 5-15	Grants AQ system privileges to users and roles.
REVOKE_SYSTEM_PRIVILEGE procedure on page 5-16	Revokes AQ system privileges from users and roles.
GRANT_QUEUE_PRIVILEGE procedure on page 5-17	Grants privileges on a queue to users and roles.
REVOKE_QUEUE_PRIVILEGE procedure on page 5-18	Revokes privileges on a queue from users and roles.
ADD_SUBSCRIBER procedure on page 5-18	Adds a default subscriber to a queue.
ALTER_SUBSCRIBER procedure on page 5-19	Alters existing properties of a subscriber to a specified queue.
REMOVE_SUBSCRIBER procedure on page 5-20	Removes a default subscriber from a queue.
SCHEDULE_PROPAGATION procedure on page 5-21	Schedules propagation of messages from a queue to a destination identified by a specific dblink.
UNSCHEDULE_PROPAGATION procedure on page 5-22	Unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific dblink.
VERIFY_QUEUE_TYPES procedure on page 5-23	Verifies that the source and destination queues have identical types.
ALTER_PROPAGATION_SCHEDULE procedure on page 5-24	Alters parameters for a propagation schedule.
ENABLE_PROPAGATION_SCHEDULE procedure on page 5-25	Enables a previously disabled propagation schedule.
DISABLE_PROPAGATION_SCHEDULE Procedure on page 5-26	Disables a propagation schedule.

CREATE_QUEUE_TABLE procedure

This procedure creates a queue table for messages of a pre-defined type. The sort keys for dequeue ordering, if any, must be defined at table creation time. The following objects are created at this time:

- A default exception queue associated with the queue table, called aq$_<queue_table_name>_e.
- A read-only view, which is used by AQ applications for querying queue data, called aq$<queue_table_name>.
- An index or an index organized table (IOT) in the case of multiple consumer queues for the queue monitor operations, called aq$_<queue_table_name>_t.
- An index or an index organized table in the case of multiple consumer queues for dequeue operations, called aq$_<queue_table_name>_i.

For Oracle8*i*-compatible queue tables, the following two index organized tables are created:

- A table called aq$_<queue_table_name>_s. This table stores information about the subscribers.
- A table called aq$_<queue_table_name>_r. This table stores information about rules on subscriptions.
- An index organized table called aq$_<queue_table_name>_h. This table stores the dequeue history data.

Syntax

```
DBMS_AQADM.CREATE_QUEUE_TABLE (
   queue_table        IN      VARCHAR2,
   queue_payload_type IN      VARCHAR2,
   storage_clause     IN      VARCHAR2       DEFAULT NULL,
   sort_list          IN      VARCHAR2       DEFAULT NULL,
   multiple_consumers IN      BOOLEAN        DEFAULT FALSE,
   message_grouping   IN      BINARY_INTEGER DEFAULT NONE,
   comment            IN      VARCHAR2       DEFAULT NULL,
   auto_commit        IN      BOOLEAN        DEFAULT TRUE,
   primary_instance   IN      BINARY_INTEGER DEFAULT 0,
   secondary_instance IN      BINARY_INTEGER DEFAULT 0,
   compatible         IN      VARCHAR2       DEFAULT NULL);
```

Parameters

Table 5–3 CREATE_QUEUE_TABLE Procedure Parameters

Parameter	Description
queue_table	Name of a queue table to be created.
queue_payload_type	Type of the user data stored. See "Type Name" on page 4-3 for valid values for this parameter.
storage_clause	Storage parameter. The storage parameter is included in the CREATE TABLE statement when the queue table is created. The storage parameter can be made up of any combinations of the following parameters: PCTFREE, PCTUSED, INITRANS, MAXTRANS, TABLEPSACE, LOB, and a table storage clause. If a tablespace is not specified here, then the queue table and all its related objects are created in the default user tablespace. If a tablespace is specified here, then the queue table and all its related objects are created in the tablespace specified in the storage clause. See *Oracle8i SQL Reference* for the usage of these parameters.
sort_list	The columns to be used as the sort key in ascending order. Sort_list has the following format: '<sort_column_1>,<sort_column_2>' The allowed column names are priority and enq_time. If both columns are specified, then <sort_column_1> defines the most significant order. After a queue table is created with a specific ordering mechanism, all queues in the queue table inherit the same defaults. The order of a queue table cannot be altered after the queue table has been created. If no sort list is specified, then all the queues in this queue table are sorted by the enqueue time in ascending order. This order is equivalent to FIFO order. Even with the default ordering defined, a dequeuer is allowed to choose a message to dequeue by specifying its msgid or correlation. Msgid, correlation, and sequence_deviation take precedence over the default dequeueing order, if they are specified.

Table 5–3 CREATE_QUEUE_TABLE Procedure Parameters

Parameter	Description
multiple_consumers	FALSE: Queues created in the table can only have one consumer per message. This is the default. TRUE: Queues created in the table can have multiple consumers per message.
message_grouping	Message grouping behavior for queues created in the table. NONE: Each message is treated individually. TRANSACTIONAL: Messages enqueued as part of one transaction are considered part of the same group and can be dequeued as a group of related messages.
comment	User-specified description of the queue table. This user comment is added to the queue catalog.
auto_commit	TRUE: causes the current transaction, if any, to commit before the CREATE_QUEUE_TABLE operation is carried out. The CREATE_QUEUE_TABLE operation becomes persistent when the call returns. This is the default. FALSE: The operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.
primary_instance	The primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table are done in this instance. The default value for primary instance is 0, which means queue monitor scheduling and propagation will be done in any available instance.
secondary_instance	The queue table fails over to the secondary instance if the primary instance is not available. The default value is 0, which means that the queue-table will fail over to any available instance.
compatible	The lowest database version with which the queue is compatible. Currently the possible values are either '8.0' or '8.1'. The default is '8.0'.

Usage Notes

CLOB, BLOB, and BFILE are valid attributes for AQ object type payloads. However, only CLOB and BLOB can be propagated using AQ propagation in Oracle8*i* release 8.1.5. See the *Oracle8i Application Developer's Guide - Advanced Queuing* for more information.

You can specify and modify the primary_instance and secondary_instance only in 8.1-compatible mode.

You cannot specify a secondary instance unless there is a primary instance.

ALTER_QUEUE_TABLE procedure

This procedure alters the existing properties of a queue table.

Syntax

```
DBMS_AQADM.ALTER_QUEUE_TABLE (
   queue_table          IN   VARCHAR2,
   comment              IN   VARCHAR2       DEFAULT NULL,
   primary_instance     IN   BINARY_INTEGER DEFAULT NULL,
   secondary_instance   IN   BINARY_INTEGER DEFAULT NULL);
```

Parameters

Table 5–4 ALTER_QUEUE_TABLE Procedure Parameters

Parameter	Description
queue_table	Name of a queue table to be created.
comment	Modifies the user-specified description of the queue table. This user comment is added to the queue catalog. The default value is NULL which means that the value will not be changed.
primary_instance	This is the primary owner of the queue table. Queue monitor scheduling and propagation for the queues in the queue table will be done in this instance. The default value is NULL, which means that the current value will not be changed.
secondary_instance	The queue table fails over to the secondary instance if the primary instance is not available. The default value is NULL, which means that the current value will not be changed.

DROP_QUEUE_TABLE procedure

This procedure drops an existing queue table. All the queues in a queue table must be stopped and dropped before the queue table can be dropped. You must do this explicitly unless the `force` option is used, in which case this done automatically.

Syntax

```
DBMS_AQADM.DROP_QUEUE_TABLE (
   queue_table       IN    VARCHAR2,
   force             IN    BOOLEAN DEFAULT FALSE,
   auto_commit       IN    BOOLEAN DEFAULT TRUE);
```

Parameters

Table 5–5 DROP_QUEUE_TABLE Procedure Parameters

Parameter	Description
queue_table	Name of a queue table to be dropped.
force	FALSE: The operation does not succeed if there are any queues in the table. This is the default. TRUE: All queues in the table are stopped and dropped automatically.
auto_commit	TRUE: Causes the current transaction, if any, to commit before the DROP_QUEUE_TABLE operation is carried out. The DROP_QUEUE_TABLE operation becomes persistent when the call returns. This is the default. FALSE: The operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.

CREATE_QUEUE procedure

This procedure creates a queue in the specified queue table.

Syntax

```
DBMS_AQADM.CREATE_QUEUE (
   queue_name          IN      VARCHAR2,
   queue_table         IN      VARCHAR2,
   queue_type          IN      BINARY_INTEGER DEFAULT NORMAL_QUEUE,
   max_retries         IN      NUMBER         DEFAULT NULL,
   retry_delay         IN      NUMBER         DEFAULT 0,
   retention_time      IN      NUMBER         DEFAULT 0,
   dependency_tracking IN      BOOLEAN        DEFAULT FALSE,
   comment             IN      VARCHAR2       DEFAULT NULL,
   auto_commit         IN      BOOLEAN        DEFAULT TRUE);
```

Parameters

Table 5–6 CREATE_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue that is to be created. The name must be unique within a schema and must follow object name guidelines in the *Oracle8i SQL Reference* with regard to reserved characters.
queue_table	Name of the queue table that will contain the queue.
queue_type	Specifies whether the queue being created is an exception queue or a normal queue. NORMAL_QUEUE: The queue is a normal queue. This is the default. EXCEPTION_QUEUE: It is an exception queue. Only the dequeue operation is allowed on the exception queue.
max_retries	Limits the number of times a dequeue with the REMOVE mode can be attempted on a message. The count is incremented when the application issues a rollback after executing the dequeue. The message is moved to the exception queue when it is reaches its max_retries. Note that max_retries is supported for all single consumer queues and 8.1-compatible multiconsumer queues but not for 8.0-compatible multiconsumer queues.

Table 5–6 CREATE_QUEUE Procedure Parameters

Parameter	Description
retry_delay	Delay time, in seconds, before this message is scheduled for processing again after an application rollback. The default is 0, which means the message can be retried as soon as possible. This parameter has no effect if max_retries is set to 0. Note that rety_delay is supported for single consumer queues and 8.1-compatible multiconsumer queues but not for 8.0-compatible multiconsumer queues.
retention_time	Number of seconds for which a message is retained in the queue table after being dequeued from the queue. INFINITE: Message is retained forever. NUMBER: Number of seconds for which to retain the messages. The default is 0; i.e. no retention.
dependency_ tracking	Reserved for future use. FALSE: This is the default. TRUE: Not permitted in this release.
comment	User-specified description of the queue. This user comment is added to the queue catalog.
auto_commit	TRUE: Causes the current transaction, if any, to commit before the CREATE_QUEUE operation is carried out. The CREATE_QUEUE operation becomes persistent when the call returns. This is the default. FALSE: The operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.

Usage Notes

All queue names must be unique within a schema. After a queue is created with CREATE_QUEUE, it can be enabled by calling START_QUEUE. By default, the queue is created with both enqueue and dequeue disabled.

CREATE_NP_QUEUE procedure

Create a non-persistent RAW queue.

Syntax

```
DBMS_AQADM.CREATE_NP_QUEUE (
   queue_name              IN       VARCHAR2,
   multiple_consumers      IN       BOOLEAN  DEFAULT FALSE,
   comment                 IN       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 5–7 CREATE_NP_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the non-persistent queue that is to be created. The name must be unique within a schema and must follow object name guidelines in the *Oracle8i SQL Reference* with regard to reserved characters.
multiple_consumers	FALSE: Queues created in the table can only have one consumer per message. This is the default. TRUE: Queues created in the table can have multiple consumers per message. Note that this parameter is distinguished at the queue level, because a non-persistent queue does not inherit this characteristic from any user-created queue table.
comment	User-specified description of the queue. This user comment is added to the queue catalog.

Usage Notes

The queue may be either single-consumer or multiconsumer queue. All queue names must be unique within a schema. The queues are created in a 8.1-compatible system-created queue table (AQ$_MEM_SC or AQ$_MEM_MC) in the same schema as that specified by the queue name.

If the queue name does not specify a schema name, then the queue is created in the login user's schema. After a queue is created with CREATE_NP_QUEUE, it can be enabled by calling START_QUEUE. By default, the queue is created with both enqueue and dequeue disabled.

You cannot dequeue from a non-persistent queue. The only way to retrieve a message from a non-persistent queue is by using the OCI notification mechanism.

You cannot invoke the `listen` call on a non-persistent queue.

ALTER_QUEUE procedure

This procedure alters existing properties of a queue. The parameters `max_retries`, `retention_time`, and `retry_delay` are not supported for non-persistent queues.

Syntax

```
DBMS_AQADM.ALTER_QUEUE (
   queue_name        IN    VARCHAR2,
   max_retries       IN    NUMBER   DEFAULT NULL,
   retry_delay       IN    NUMBER   DEFAULT NULL,
   retention_time    IN    NUMBER   DEFAULT NULL,
   auto_commit       IN    BOOLEAN  DEFAULT TRUE,
   comment           IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 5–8 ALTER_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue that is to be altered.
max_retries	Limits the number of times a dequeue with REMOVE mode can be attempted on a message. The count is incremented when the application issues a rollback after executing the dequeue. If the time at which one of the retries has passed the expiration time, then no further retries are attempted. Default is NULL, which means that the value will not be altered. Note that `max_retries` is supported for all single consumer queues and 8.1-compatible multiconsumer queues but not for 8.0-compatible multiconsumer queues.
retry_delay	Delay time in seconds before this message is scheduled for processing again after an application rollback. The default is NULL, which means that the value will not be altered. Note that `retry_delay` is supported for single consumer queues and 8.1-compatible multiconsumer queues but not for 8.0-compatible multiconsumer queues.

Table 5–8 ALTER_QUEUE Procedure Parameters

Parameter	Description
retention_time	Retention time in seconds for which a message is retained in the queue table after being dequeued. The default is NULL, which means that the value will not be altered.
auto_commit	TRUE: Causes the current transaction, if any, to commit before the ALTER_QUEUE operation is carried out. The ALTER_QUEUE operation become persistent when the call returns. This is the default. FALSE: The operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.
comment	User-specified description of the queue. This user comment is added to the queue catalog. The default value is NULL, which means that the value will not be changed.

DROP_QUEUE procedure

This procedure drops an existing queue. DROP_QUEUE is not allowed unless STOP_QUEUE has been called to disable the queue for both enqueuing and dequeuing. All the queue data is deleted as part of the drop operation.

Syntax

```
DBMS_AQADM.DROP_QUEUE (
   queue_name         IN    VARCHAR2,
   auto_commit        IN    BOOLEAN DEFAULT TRUE);
```

Parameters

Table 5–9 DROP_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue that is to be dropped.

Table 5–9 DROP_QUEUE Procedure Parameters

auto_commit	TRUE: Causes the current transaction, if any, to commit before the DROP_QUEUE operation is carried out. The DROP_QUEUE operation becomes persistent when the call returns. This is the default. FALSE: The operation is part of the current transaction and becomes persistent only when the caller enters a commit. Caution: This parameter has been deprecated.

START_QUEUE procedure

This procedure enables the specified queue for enqueuing and/or dequeueing.

After creating a queue the administrator must use START_QUEUE to enable the queue. The default is to enable it for both ENQUEUE and DEQUEUE. Only dequeue operations are allowed on an exception queue. This operation takes effect when the call completes and does not have any transactional characteristics.

Syntax

```
DBMS_AQADM.START_QUEUE (
   queue_name      IN     VARCHAR2,
   enqueue         IN     BOOLEAN DEFAULT TRUE,
   dequeue         IN     BOOLEAN DEFAULT TRUE);
```

Parameters

Table 5–10 START_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue to be enabled.
enqueue	Specifies whether ENQUEUE should be enabled on this queue. TRUE: Enable ENQUEUE. This is the default. FALSE: Do not alter the current setting.
dequeue	Specifies whether DEQUEUE should be enabled on this queue. TRUE: Enable DEQUEUE. This is the default. FALSE: Do not alter the current setting.

STOP_QUEUE procedure

This procedure disables enqueuing and/or dequeuing on the specified queue.

By default, this call disables both ENQUEUEs or DEQUEUEs. A queue cannot be stopped if there are outstanding transactions against the queue. This operation takes effect when the call completes and does not have any transactional characteristics.

Syntax

```
DBMS_AQADM.STOP_QUEUE (
   queue_name      IN   VARCHAR2,
   enqueue         IN   BOOLEAN DEFAULT TRUE,
   dequeue         IN   BOOLEAN DEFAULT TRUE,
   wait            IN   BOOLEAN DEFAULT TRUE);
```

Parameters

Table 5–11 STOP_QUEUE Procedure Parameters

Parameter	Description
queue_name	Name of the queue to be disabled.
enqueue	Specifies whether ENQUEUE should be disabled on this queue. TRUE: Disable ENQUEUE. This is the default. FALSE: Do not alter the current setting.
dequeue	Specifies whether DEQUEUE should be disabled on this queue. TRUE: Disable DEQUEUE. This is the default. FALSE: Do not alter the current setting.
wait	Specifies whether to wait for the completion of outstanding transactions. TRUE: Wait if there are any outstanding transactions. In this state no new transactions are allowed to enqueue to or dequeue from this queue. FALSE: Return immediately either with a success or an error.

GRANT_SYSTEM_PRIVILEGE procedure

This procedure grants AQ system privileges to users and roles. The privileges are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. Initially, only SYS and SYSTEM can use this procedure successfully.

Syntax

```
DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE (
   privilege         IN    VARCHAR2,
   grantee           IN    VARCHAR2,
   admin_option      IN    BOOLEAN := FALSE);
```

Parameters

Table 5–12 GRANT_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The AQ system privilege to grant. The options are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. The operations allowed for each system privilege are specified as follows: ENQUEUE_ANY: users granted with this privilege are allowed to enqueue messages to any queues in the database. DEQUEUE_ANY: users granted with this privilege are allowed to dequeue messages from any queues in the database. MANAGE_ANY: users granted with this privilege are allowed to run DBMS_AQADM calls on any schemas in the database.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.
admin_option	Specifies if the system privilege is granted with the ADMIN option or not. If the privilege is granted with the ADMIN option, then the grantee is allowed to use this procedure to grant the system privilege to other users or roles. The default is FALSE.

REVOKE_SYSTEM_PRIVILEGE procedure

This procedure revokes AQ system privileges from users and roles. The privileges are ENQUEUE_ANY, DEQUEUE_ANY and MANAGE_ANY. The ADMIN option for a system privilege cannot be selectively revoked.

Syntax

```
DBMS_AQADM.REVOKE_SYSTEM_PRIVILEGE (
   privilege         IN   VARCHAR2,
   grantee           IN   VARCHAR2);
```

Parameters

Table 5–13 REVOKE_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The AQ system privilege to revoke. The options are ENQUEUE_ANY, DEQUEUE_ANY, and MANAGE_ANY. The ADMIN option for a system privilege cannot be selectively revoked.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.

GRANT_QUEUE_PRIVILEGE procedure

This procedure grants privileges on a queue to users and roles. The privileges are ENQUEUE or DEQUEUE. Initially, only the queue table owner can use this procedure to grant privileges on the queues.

Syntax

```
DBMS_AQADM.GRANT_QUEUE_PRIVILEGE (
   privilege        IN    VARCHAR2,
   queue_name       IN    VARCHAR2,
   grantee          IN    VARCHAR2,
   grant_option     IN    BOOLEAN := FALSE);
```

Parameters

Table 5–14 GRANT_QUEUE_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The AQ queue privilege to grant. The options are ENQUEUE, DEQUEUE, and ALL. ALL means both ENQUEUE and DEQUEUE.
queue_name	Name of the queue.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role.
grant_option	Specifies if the access privilege is granted with the GRANT option or not. If the privilege is granted with the GRANT option, then the grantee is allowed to use this procedure to grant the access privilege to other users or roles, regardless of the ownership of the queue table. The default is FALSE.

REVOKE_QUEUE_PRIVILEGE procedure

This procedure revokes privileges on a queue from users and roles. The privileges are ENQUEUE or DEQUEUE. To revoke a privilege, the revoker must be the original grantor of the privilege. The privileges propagated through the GRANT option are revoked if the grantor's privileges are revoked.

Syntax

```
DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE (
   privilege          IN      VARCHAR2,
   queue_name         IN      VARCHAR2,
   grantee            IN      VARCHAR2);
```

Parameters

Table 5–15 REVOKE_QUEUE_PRIVILEGE Procedure Parameters

Parameter	Description
privilege	The AQ queue privilege to revoke. The options are ENQUEUE, DEQUEUE, and ALL. ALL means both ENQUEUE and DEQUEUE.
queue_name	Name of the queue.
grantee	Grantee(s). The grantee(s) can be a user, a role, or the PUBLIC role. If the privilege has been propagated by the grantee through the GRANT option, then the propagated privilege is also revoked.

ADD_SUBSCRIBER procedure

This procedure adds a default subscriber to a queue.

Syntax

```
DBMS_AQADM.ADD_SUBSCRIBER (
   queue_name      IN    VARCHAR2,
   subscriber      IN    sys.aq$_agent,
   rule            IN    VARCHAR2 DEFAULT NULL);
```

Parameters

Table 5–16 ADD_SUBSCRIBER Procedure Parameters

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent on whose behalf the subscription is being defined.
rule	A conditional expression based on the message properties, the message data properties and PL/SQL functions. A rule is specified as a Boolean expression using syntax similar to the WHERE clause of a SQL query. This Boolean expression can include conditions on message properties, user data properties (object payloads only), and PL/SQL or SQL functions (as specified in the where clause of a SQL query). Currently supported message properties are priority and corrid. To specify rules on a message payload (object payload), use attributes of the object type in clauses. You must prefix each attribute with tab.user_data as a qualifier to indicate the specific column of the queue table that stores the payload. The rule parameter cannot exceed 4000 characters.

Usage Notes

A program can enqueue messages to a specific list of recipients or to the default list of subscribers. This operation only succeeds on queues that allow multiple consumers. This operation takes effect immediately, and the containing transaction is committed. Enqueue requests that are executed after the completion of this call will reflect the new behavior.

Any string within the rule must be quoted as shown below;

```
rule   => 'PRIORITY <= 3 AND CORRID =  ''FROM JAPAN'''
```

Note that these are all single quotation marks.

ALTER_SUBSCRIBER procedure

This procedure alters existing properties of a subscriber to a specified queue. Only the rule can be altered.

Syntax

```
DBMS_AQADM.ALTER_SUBSCRIBER (
   queue_name      IN    VARCHAR2,
   subscriber      IN    sys.aq$_agent,
   rule            IN    VARCHAR2);
```

Parameters

Table 5–17 ALTER_SUBSCRIBER Procedure Parameters

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent on whose behalf the subscription is being altered. See "Agent" on page 4-4.
rule	A conditional expression based on the message properties, the message data properties and PL/SQL functions. Note: The rule parameter cannot exceed 4000 characters. To eliminate the rule, set the rule parameter to NULL.

REMOVE_SUBSCRIBER procedure

This procedure removes a default subscriber from a queue. This operation takes effect immediately, and the containing transaction is committed. All references to the subscriber in existing messages are removed as part of the operation.

Syntax

```
DBMS_AQADM.REMOVE_SUBSCRIBER (
   queue_name          IN          VARCHAR2,
   subscriber          IN          sys.aq$_agent);
```

Parameters

Table 5–18 REMOVE_SUBSCRIBER Procedure Parameters

Parameter	Description
queue_name	Name of the queue.
subscriber	Agent who is being removed. See "Agent" on page 4-4.

SCHEDULE_PROPAGATION procedure

This procedure schedules propagation of messages from a queue to a destination identified by a specific dblink.

Messages may also be propagated to other queues in the same database by specifying a NULL destination. If a message has multiple recipients at the same destination in either the same or different queues, then the message is propagated to all of them at the same time.

Syntax

```
DBMS_AQADM.SCHEDULE_PROPAGATION (
   queue_name      IN    VARCHAR2,
   destination     IN    VARCHAR2 DEFAULT NULL,
   start_time      IN    DATE     DEFAULT SYSDATE,
   duration        IN    NUMBER   DEFAULT NULL,
   next_time       IN    VARCHAR2 DEFAULT NULL,
   latency         IN    NUMBER   DEFAULT 60);
```

Parameters

Table 5–19 SCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the administrative user.
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
start_time	Initial start time for the propagation window for messages from the source queue to the destination.
duration	Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.

Table 5–19 SCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
next_time	Date function to compute the start of the next propagation window from the end of the current window. If this value is NULL, then propagation is stopped at the end of the current window. For example, to start the window at the same time every day, next_time should be specified as 'SYSDATE + 1 - duration / 86400'.
latency	Maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. For example: If the latency is 60 seconds, then during the propagation window; if there are no messages to be propagated, then messages from that queue for the destination are not propagated for at least 60 more seconds. It is at least 60 seconds before the queue is checked again for messages to be propagated for the specified destination. If the latency is 600, then the queue is not checked for 10 minutes, and if the latency is 0, then a job queue process will be waiting for messages to be enqueued for the destination. As soon as a message is enqueued, it is propagated.

UNSCHEDULE_PROPAGATION procedure

This procedure unschedules previously scheduled propagation of messages from a queue to a destination identified by a specific dblink.

Syntax

```
DBMS_AQADM.UNSCHEDULE_PROPAGATION (
   queue_name      IN   VARCHAR2,
   destination     IN   VARCHAR2 DEFAULT NULL);
```

Parameters

Table 5–20 UNSCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the administrative user.

Table 5–20 UNSCHEDULE_PROPAGATION Procedure Parameters

Parameter	Description
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.

VERIFY_QUEUE_TYPES procedure

This procedure verifies that the source and destination queues have identical types. The result of the verification is stored in the table sys.aq$_message_types, overwriting all previous output of this command.

Syntax

```
DBMS_AQADM.VERIFY_QUEUE_TYPES (
   src_queue_name     IN    VARCHAR2,
   dest_queue_name    IN    VARCHAR2,
   destination        IN    VARCHAR2 DEFAULT NULL,
   rc                 OUT   BINARY_INTEGER);
```

Parameters

Table 5–21 VERIFY_QUEUE_TYPES Procedure Parameters

Parameter	Description
src_queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
dest_queue_name	Name of the destination queue where messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.

Table 5–21 VERIFY_QUEUE_TYPES Procedure Parameters

Parameter	Description
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.
rc	Return code for the result of the procedure. If there is no error, and if the source and destination queue types match, then the result is 1. If they do not match, then the result is 0. If an Oracle error is encountered, then it is returned in rc.

ALTER_PROPAGATION_SCHEDULE procedure

This procedure alters parameters for a propagation schedule.

Syntax

```
DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE (
   queue_name       IN     VARCHAR2,
   destination      IN     VARCHAR2 DEFAULT NULL,
   duration         IN     NUMBER   DEFAULT NULL,
   next_time        IN     VARCHAR2 DEFAULT NULL,
   latency          IN     NUMBER   DEFAULT 60);
```

Parameters

Table 5–22 ALTER_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.

Table 5–22 ALTER_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
duration	Duration of the propagation window in seconds. A NULL value means the propagation window is forever or until the propagation is unscheduled.
next_time	Date function to compute the start of the next propagation window from the end of the current window. If this value is NULL, then propagation is stopped at the end of the current window. For example, to start the window at the same time every day, next_time should be specified as 'SYSDATE + 1 - duration/86400'.
latency	Maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. The default value is 60. Caution: if latency is not specified for this call, then latency will over-write any existing value with the default value. For example, if the latency is 60 seconds, then during the propagation window; if there are no messages to be propagated, then messages from that queue for the destination will not be propagated for at least 60 more seconds. It will be at least 60 seconds before the queue will be checked again for messages to be propagated for the specified destination. If the latency is 600, then the queue will not be checked for 10 minutes and if the latency is 0, then a job queue process will be waiting for messages to be enqueued for the destination and as soon as a message is enqueued it will be propagated.

ENABLE_PROPAGATION_SCHEDULE procedure

This procedure enables a previously disabled propagation schedule.

Syntax

```
DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE (
   queue_name       IN     VARCHAR2,
   destination      IN     VARCHAR2 DEFAULT NULL);
```

Parameters

Table 5–23 ENABLE_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.

DISABLE_PROPAGATION_SCHEDULE Procedure

This procedure disables a propagation schedule.

Syntax

```
DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE (
   queue_name       IN     VARCHAR2,
   destination      IN     VARCHAR2 DEFAULT NULL);
```

Parameters

Table 5–24 DISABLE_PROPAGATION_SCHEDULE Procedure Parameters

Parameter	Description
queue_name	Name of the source queue whose messages are to be propagated, including the schema name. If the schema name is not specified, then it defaults to the schema name of the user.
destination	Destination dblink. Messages in the source queue for recipients at this destination are propagated. If it is NULL, then the destination is the local database and messages are propagated to other queues in the local database. The length of this field is currently limited to 128 bytes, and if the name is not fully qualified, then the default domain name is used.

MIGRATE_QUEUE_TABLE procedure

This procedure upgrades an 8.0-compatible queue table to an 8.1-compatible queue table, or downgrades an 8.1-compatible queue table to an 8.0-compatible queue table.

Syntax

```
DBMS_AQADM.MIGRATE_QUEUE_TABLE (
   queue_table    IN   VARCHAR2,
   compatible     IN   VARCHAR2);
```

Parameters

Table 5–25 MIGRATE_QUEUE_TABLE Procedure Parameters

Parameter	Description
queue_table	Specifies name of the queue table to be migrated.
compatible	Set this to '8.1' to upgrade an 8.0-compatible queue table, or set this to '8.0' to downgrade an 8.1-compatible queue table.

6

DBMS_DDL

This package provides access to some SQL Data Definition Language (DDL) statements from stored procedures. It also provides special administration operations that are not available as DDLs.

The `ALTER_COMPILE` and `ANALYZE_OBJECT` procedures commit the current transaction, perform the operation, and then commit again.

Requirements

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

Summary of Subprograms

Table 6–1 DBMS_DDL Package Subprograms

Subprogram	Description
ALTER_COMPILE procedure on page 6-2	Compiles the PL/SQL object.
ANALYZE_OBJECT procedure on page 6-3	Provides statistics for the database object.

ALTER_COMPILE procedure

This procedure is equivalent to the following SQL statement:

```
ALTER PROCEDURE|FUNCTION|PACKAGE [<schema>.] <name> COMPILE [BODY]
```

If the named object is this package, or any packages upon which it depends (currently, STANDARD or DBMS_STANDARD), then the procedure simply returns, and these packages are successfully compiled.

Syntax

```
DBMS_DDL.ALTER_COMPILE (
   type   VARCHAR2,
   schema VARCHAR2,
   name   VARCHAR2);
```

Parameters

Table 6–2 ALTER_COMPILE Procedure Parameters

Parameter	Description
type	Must be either PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY or TRIGGER.
schema	Schema name. If NULL, then use current schema (case-sensitive).

Table 6–2 ALTER_COMPILE Procedure Parameters

Parameter	Description
name	Name of the object (case-sensitive).

Exceptions

Table 6–3 ALTER_COMPILE Procedure Exceptions

Exception	Description
ORA-20000:	Insufficient privileges or object does not exist.
ORA-20001:	Remote object, cannot compile.
ORA-20002:	Bad value for object type Should be either PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, or TRIGGER.

ANALYZE_OBJECT procedure

This procedure provides statistics for the given table, index, or cluster. It is equivalent to the following SQL statement:

```
ANALYZE TABLE|CLUSTER|INDEX [<schema>.]<name> [<method>] STATISTICS [SAMPLE <n>
[ROWS|PERCENT]]
```

Syntax

```
DBMS_DDL.ANALYZE_OBJECT (
   type             VARCHAR2,
   schema           VARCHAR2,
   name             VARCHAR2,
   method           VARCHAR2,
   estimate_rows    NUMBER   DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT NULL,
   method_opt       VARCHAR2 DEFAULT NULL,
   partname         VARCHAR2 DEFAULT NULL);
```

Parameters

Table 6–4 ANALYZE_OBJECT Procedure Parameters

Parameter	Description
type	One of TABLE, CLUSTER or INDEX. If none of these, the procedure just returns.
schema	Schema of object to analyze. NULL means current schema, case-sensitive.
name	Name of object to analyze, case-sensitive.
method	One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE, then either estimate_rows or estimate_percent must be non-zero.
estimate_rows	Number of rows to estimate.
estimate_percent	Percentage of rows to estimate. If estimate_rows is specified, then ignore this parameter.
method_opt	Method options of the following format. [FOR TABLE] [FOR ALL [INDEXED] COLUMNS] [SIZE n] [FOR ALL INDEXES]
partname	Specific partition to be analyzed.

Exceptions

Table 6–5 ANALYZE_OBJECT Procedure Exceptions

Exception	Description
ORA-20000:	Insufficient privileges or object does not exist.
ORA-20001:	Bad value for object type. Should be either TABLE, INDEX or CLUSTER.
ORA-20002:	METHOD must be one of COMPUTE, ESTIMATE or DELETE.

7

DBMS_DEBUG

DBMS_DEBUG is a PL/SQL API to the PL/SQL debugger layer, *Probe*, in the Oracle server.

This API is primarily intended to implement server-side debuggers, and it provides a way to debug server-side PL/SQL program units.

> **Note:** The term *program unit* refers to a PL/SQL program of any kind (procedure, function, package, package body, trigger, anonymous block, object type, or object type body).

Using DBMS_DEBUG

In order to debug server-side code, it is necessary to have two database sessions: one session to run the code in debug-mode (the target session), and a second session to supervise the target session (the debug session).

The target session becomes available for debugging by making initializing calls with DBMS_DEBUG. This marks the session, so that the PL/SQL interpreter runs in debug-mode and generates debug events. As debug events are generated, they are posted from the session. In most cases, debug events require return notification: the interpreter pauses awaiting a reply.

Meanwhile, the debug session must also initialize itself using DBMS_DEBUG: This tells it what target session to supervise. The debug session may then call entrypoints in DBMS_DEBUG to read events which were posted from the target session and to communicate with the target session.

> **See Also:** Figure 7–1 and Figure 7–2 illustrate the flow of operations in the session to be debugged and in the debugging session.

DBMS_DEBUG does not provide any interface to the PL/SQL compiler; however, it does depend on debug information optionally generated by the compiler. Without debug information, it is not possible to look at or modify the values of parameters or variables. There are two ways to ensure that debug information is generated: through a session switch, or through individual recompilation.

To set the session switch, enter the following statement:

```
ALTER SESSION SET PLSQL_DEBUG = true;
```

This instructs the compiler to generate debug information for the remainder of the session. It does not recompile any existing PL/SQL.

To generate debug information for existing PL/SQL code, use one of the following statements (the second recompiles a package or type body):

```
ALTER [PROCEDURE | FUNCTION | PACKAGE | TRIGGER | TYPE] <name> COMPILE DEBUG;
ALTER [PACKAGE | TYPE] <name> COMPILE DEBUG BODY;
```

Figure 7–1 Target Session

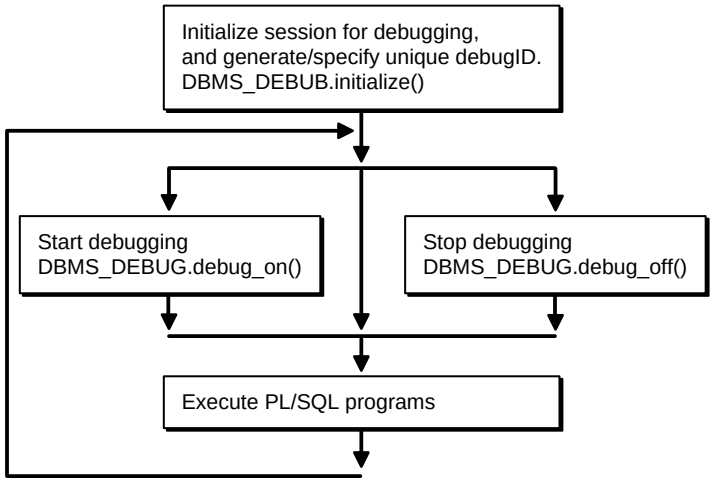


Figure 7–2 Debug Session

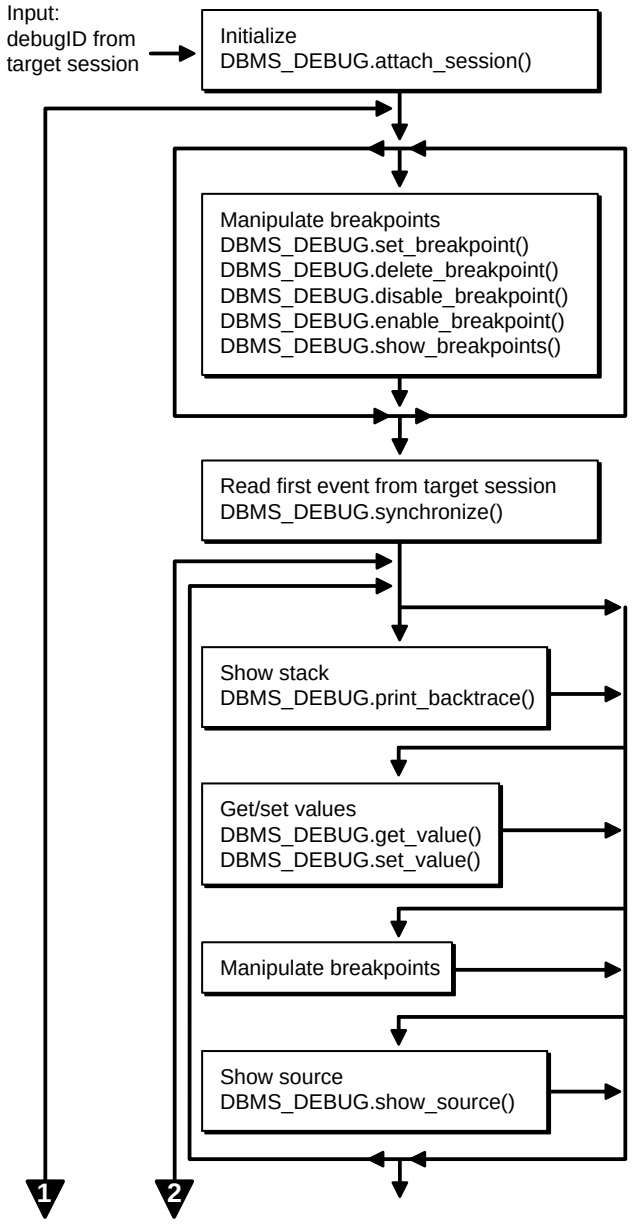
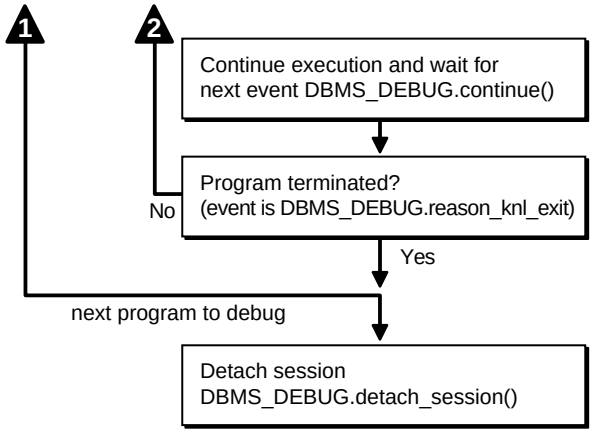


Figure 7–2 Debug Session (Cont.)



Control of the Interpreter

The interpreter pauses execution at the following times:

1. At startup of the interpreter, so that any deferred breakpoints may be installed prior to execution.
2. At any line containing an enabled breakpoint.
3. At any line where an *interesting* event occurs. The set of interesting events is specified by the flags passed to `DBMS_DEBUG.CONTINUE` in the `breakflags` parameter.

Usage Notes

Session Termination

There is no event for session termination. Therefore, it is the responsibility of the debug session to check and make sure that the target session has not ended. A call to `DBMS_DEBUG.SYNCHRONIZE` after the target session has ended causes the debug session to hang until it times out.

Deferred Operations

The diagram suggests that it is possible to set breakpoints prior to having a target session. This is true. In this case, Probe caches the breakpoint request and transmits it to the target session at first synchronization. However, if a breakpoint request is deferred in this fashion, then:

- `SET_BREAKPOINT` does not set the breakpoint number (it can be obtained later from `SHOW_BREAKPOINTS` if necessary).
- `SET_BREAKPOINT` does not validate the breakpoint request. If the requested source line does not exist, then an error silently occurs at synchronization, and no breakpoint is set.

Diagnostic Output

To debug Probe, there are *diagnostics* parameters to some of the calls in `DBMS_DEBUG`. These parameters specify whether to place diagnostic output in the RDBMS tracefile. If output to the RDBMS tracefile is disabled, then these parameters have no effect.

Types

PROGRAM_INFO This type specifies a program location. It is a line number in a program unit.This is used for stack backtraces and for setting and examining breakpoints.The read-only fields are currently ignored by Probe for breakpoint operations. They are set by Probe only for stack backtraces.

`EntrypointName`	Null, unless this is a nested procedure or function.
`LibunitType`	Disambiguate among objects that share the same namespace (for example, procedure and package specifications). See the Libunit Types on page 7-9 for more information.

```
TYPE program_info IS RECORD
(
    -- The following fields are used when setting a breakpoint
    Namespace        BINARY_INTEGER,  -- See 'NAMESPACES' section below.
    Name             VARCHAR2(30),    -- name of the program unit
    Owner            VARCHAR2(30),    -- owner of the program unit
    Dblink           VARCHAR2(30),    -- database link, if remote
    Line#            BINARY_INTEGER,
    -- Read-only fields (set by Probe when doing a stack backtrace)
    LibunitType      BINARY_INTEGER,
    EntrypointName   VARCHAR2(30)
);
```

RUNTIME_INFO This type gives context information about the running program.

```
TYPE runtime_info IS RECORD
(
    Line#            BINARY_INTEGER,  -- (duplicate of program.line#)
    Terminated       BINARY_INTEGER,  -- has the program terminated?
    Breakpoint       BINARY_INTEGER,  -- breakpoint number
    StackDepth       BINARY_INTEGER,  -- number of frames on the stack
    InterpreterDepth BINARY_INTEGER,  -- <reserved field>
    Reason           BINARY_INTEGER,  -- reason for suspension
    Program          program_info     -- source location
);
```

BREAKPOINT_INFO This type gives information about a breakpoint, such as its current status and the program unit in which it was placed.

```
TYPE breakpoint_info IS RECORD
(
   -- These fields are duplicates of 'program_info':
   Name        VARCHAR2(30),
   Owner       VARCHAR2(30),
   DbLink      VARCHAR2(30),
   Line#       BINARY_INTEGER,
   LibunitType BINARY_INTEGER,
   Status      BINARY_INTEGER   -- see breakpoint_status_* below
);
```

INDEX_TABLE This type is used by GET_INDEXES to return the available indexes for an indexed table.

```
TYPE index_table IS table of BINARY_INTEGER INDEX BY BINARY_INTEGER;
```

BACKTRACE_TABLE This type is used by PRINT_BACKTRACE.

```
TYPE backtrace_table IS TABLE OF program_info INDEX BY BINARY_INTEGER;
```

BREAKPOINT_TABLE This type is used by SHOW_BREAKPOINTS.

```
TYPE breakpoint_table IS TABLE OF breakpoint_info INDEX BY BINARY_INTEGER;
```

VC2_TABLE This type is used by SHOW_SOURCE.

```
TYPE vc2_table IS TABLE OF VARCHAR2(90) INDEX BY BINARY_INTEGER;
```

Constants

A breakpoint status may have these values:

breakpoint_status_unused	Breakpoint is not in use.

Otherwise, the status is a mask of the following values:

breakpoint_status_active	A line breakpoint.
breakpoints_status_disabled	Breakpoint is currently disabled.
breakpoint_status_remote	A 'shadow' breakpoint (a local representation of a remote breakpoint).

NAMESPACES Program units on the server reside in different namespaces.When setting a breakpoint, it is necessary to specify the desired namespace.

1. Namespace_cursor contains cursors (anonymous blocks).
2. Namespace_pgkspec_or_toplevel contains:
   - Package specifications.
   - Procedures and functions that are not nested inside other packages, procedures, or functions.
   - Object types.
3. Namespace_pkg_body contains package bodies and type bodies.

4. `Namespace_trigger` contains triggers.

Libunit Types These values are used to disambiguate among objects in a given namespace. These constants are used in `PROGRAM_INFO` when Probe is giving a stack backtrace.

`LibunitType_cursor`

`LibunitType_procedure`

`LibunitType_function`

`LibunitType_package`

`LibunitType_package_body`

`LibunitType_trigger`

`LibunitType_Unknown`

Breakflags These are values to use for the `breakflags` parameter to `CONTINUE`, in order to tell Probe what events are of interest to the client. These flags may be combined.

`break_next_line`	Break at next source line (step over calls).
`break_any_call`	Break at next source line (step into calls).
`break_any_return`	Break after returning from current entrypoint (skip over any entrypoints called from the current routine).
`break_return`	Break the next time an entrypoint gets ready to return. (This includes entrypoints called from the current one. If interpreter is running `Proc1`, which calls `Proc2`, then `break_return` stops at the end of `Proc2`.)
`break_exception`	Break when an exception is raised.
`break_handler`	Break when an exception handler is executed.
`abort_execution`	Stop execution and force an 'exit' event as soon as `DBMS_DEBUG.CONTINUE` is called.

Information Flags These are flags which may be passed as the `info_requested` parameter to `SYNCHRONIZE`, `CONTINUE`, and `GET_RUNTIME_INFO`.

`info_getStackDepth`	Get the current depth of the stack.
`info_getBreakpoint`	Get the breakpoint number.
`info_getLineinfo`	Get program unit information.

Reasons for Suspension After `CONTINUE` is run, the program either runs to completion or breaks on some line.

`reason_none`	
`reason_interpreter_starting`	Interpreter is starting.
`reason_breakpoint`	Hit a breakpoint.
`reason_enter`	Procedure entry.
`reason_return`	Procedure is about to return.
`reason_finish`	Procedure is finished.
`reason_line`	Reached a new line.
`reason_interrupt`	An interrupt occurred.
`reason_exception`	An exception was raised.
`reason_exit`	Interpreter is exiting (old form).
`reason_knl_exit`	Kernel is exiting.
`reason_handler`	Start exception-handler.
`reason_timeout`	A timeout occurred.
`reason_instantiate`	Instantiation block.
`reason_abort`	Interpreter is aborting.

Error Codes

These values are returned by the various functions called in the debug session (SYNCHRONIZE, CONTINUE, SET_BREAKPOINT, and so on). If PL/SQL exceptions worked across client/server and server/server boundaries, then these would all be exceptions rather than error codes.

success	Normal termination.

Statuses returned by GET_VALUE and SET_VALUE:

error_bogus_frame	No such entrypoint on the stack.
error_no_debug_info	Program was compiled without debug symbols.
error_no_such_object	No such variable or parameter.
error_unknown_type	Debug information is unreadable.
error_indexed_table	Returned by GET_VALUE if the object is a table, but no index was provided.
error_illegal_index	No such element exists in the collection.
error_nullcollection	Table is atomically null.
error_nullvalue	Value is null.

Statuses returned by SET_VALUE:

error_illegal_value	Constraint violation.
error_illegal_null	Constraint violation.
error_value_malformed	Unable to decipher the given value.
error_other	Some other error.
error_name_incomplete	Name did not resolve to a scalar.

Statuses returned by the breakpoint functions:

error_no_such_breakpt	No such breakpoint.
error_idle_breakpt	Cannot enable or disable an unused breakpoint.
error_bad_handle	Unable to set breakpoint in given program (non-existent or security violation).

General error codes (returned by many of the DBMS_DEBUG subprograms):

error_unimplemented	Functionality is not yet implemented.
error_deferred	No program running; operation deferred.
error_exception	An exception was raised in the DBMS_DEBUG or Probe packages on the server.
error_communication	Some error other than a timeout occurred.
error_timeout	Timout occurred.

Exceptions

illegal_init	DEBUG_ON was called prior to INITIALIZE.

The following exceptions are raised by procedure SELF_CHECK:

pipe_creation_failure	Could not create a pipe.
pipe_send_failure	Could not write data to the pipe.
pipe_receive_failure	Could not read data from the pipe.
pipe_datatype_mismatch	Datatype in the pipe was wrong.
pipe_data_error	Data got garbled in the pipe.

Variables

default_timeout	The timeout value (used by both sessions).The smallest possible timeout is 1 second. If this value is set to 0, then a large value (3600) is used.

Summary of Subprograms

Table 7–1 DBMS_DEBUG Package Subprograms

Subprogram	Description
PROBE_VERSION procedure on page 7-14	Returns the version number of DBMS_DEBUG on the server.
SELF_CHECK procedure on page 7-15	Performs an internal consistency check.
SET_TIMEOUT function on page 7-16	Sets the timeout value.
INITIALIZE function on page 7-16	Sets debugID in target session.
DEBUG_ON procedure on page 7-17	Turns debug-mode on.
DEBUG_OFF procedure on page 18	Turns debug-mode off.
ATTACH_SESSION procedure on page 7-19	Notifies the debug session about the target debugID.
SYNCHRONIZE function on page 7-19	Waits for program to start running.
SHOW_SOURCE procedure on page 7-20	Fetches program source.
PRINT_BACKTRACE procedure on page 7-22	Prints a stack backtrace.
CONTINUE function on page 7-23	Continues execution of the target program.
SET_BREAKPOINT function on page 7-24	Sets a breakpoint in a program unit.
DELETE_BREAKPOINT function on page 7-25	Deletes a breakpoint.
DISABLE_BREAKPOINT function on page 7-26	Disables a breakpoint.
ENABLE_BREAKPOINT function on page 7-27	Activates an existing breakpoint.

Table 7–1 DBMS_DEBUG Package Subprograms

Subprogram	Description
SHOW_BREAKPOINTS procedure on page 7-27	Returns a listing of the current breakpoints.
GET_VALUE function on page 7-28	Gets a value from the currently-running program.
SET_VALUE function on page 7-31	Sets a value in the currently-running program.
DETACH_SESSION procedure on page 7-33	Stops debugging the target program.
GET_RUNTIME_INFO function on page 7-33	Returns information about the current program.
GET_INDEXES function on page 7-34	Returns the set of indexes for an indexed table.
EXECUTE procedure on page 7-35	Executes SQL or PL/SQL in the target session.

Common Section

These following subprograms may be called in either the target or the debug session:

- PROBE_VERSION procedure
- SELF_CHECK procedure
- SET_TIMEOUT function

PROBE_VERSION procedure

This procedure returns the version number of DBMS_DEBUG on the server.

Syntax

```
DBMS_DEBUG.PROBE_VERSION (
   major out BINARY_INTEGER,
   minor out BINARY_INTEGER);
```

Parameters

Table 7–2 PROBE_VERSION Procedure Parameters

Parameter	Description
major	Major version number.
minor	Minor version number: increments as functionality is added.

SELF_CHECK procedure

This procedure performs an internal consistency check. SELF_CHECK also runs a communications test to ensure that the Probe processes are able to communicate.

If SELF_CHECK does not return successfully, then an incorrect version of DBMS_DEBUG was probably installed on this server. The solution is to install the correct version (pbload.sql loads DBMS_DEBUG and the other relevant packages).

Syntax

```
DBMS_DEBUG.SELF_CHECK (
   timeout IN binary_integer := 60);
```

Parameters

Table 7–3 SELF_CHECK Procedure Parameters

Parameter	Description
timeout	The timeout to use for the communication test. Default is 60 seconds.

Exceptions

Table 7–4 SELF_CHECK Procedure Exceptions

Exception	Description
OER-6516	Probe version is inconsistent.
pipe_creation_failure	Could not create a pipe.
pipe_send_failure	Could not write data to the pipe.
pipe_receive_failure	Could not read data from the pipe.
pipe_datatype_mismatch	Datatype in the pipe was wrong.

Table 7–4 SELF_CHECK Procedure Exceptions

Exception	Description
pipe_data_error	Data got garbled in the pipe.

All of these exceptions are fatal. They indicate a serious problem with Probe that prevents it from working correctly.

SET_TIMEOUT function

This function sets the timeout value and returns the new timeout value.

Syntax

```
DBMS_DEBUG.SET_TIMEOUT (
   timeout BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–5 SET_TIMEOUT Function Parameters

Parameter	Description
timeout	The timeout to use for communication between the target and debug sessions.

TARGET SESSION Section

The following subprograms are run in the target session (the session that is to be debugged):

- INITIALIZE function
- DEBUG_ON procedure
- DEBUG_OFF procedure

INITIALIZE function

This function initializes the target session for debugging.

Syntax

```
DBMS_DEBUG.INITIALIZE (
   debug_session_id  IN VARCHAR2       := NULL,
   diagnostics       IN BINARY_INTEGER := 0)
  RETURN VARCHAR2;
```

Parameters

Table 7–6 INITIALIZE Function Parameters

Parameter	Description
debug_session_id	Name of session ID. If NULL, then a unique ID is generated.
diagnostics	Indicates whether to dump diagnostic output to the tracefile. 0 = (default) no diagnostics 1 = print diagnostics

Returns

The newly-registered debug session ID (debugID)

DEBUG_ON procedure

This procedure marks the target session so that all PL/SQL is run in debug mode. This must be done before any debugging can take place.

Syntax

```
DBMS_DEBUG.DEBUG_ON (
   no_client_side_plsql_engine BOOLEAN := TRUE,
   immediate                   BOOLEAN := FALSE);
```

Parameters

Table 7–7 DEBUG_ON Procedure Parameters

Parameter	Description
no_client_side_plsql_engine	Should be left to its default value unless the debugging session is taking place from a client-side PL/SQL engine.
immediate	If this is TRUE, then the interpreter immediately switches itself into debug-mode, instead of continuing in regular mode for the duration of the call.

Caution: There must be a debug session waiting if immediate is TRUE.

DEBUG_OFF procedure

This procedure notifies the target session that debugging should no longer take place in that session. It is not necessary to call this function before ending the session.

Syntax

```
DBMS_DEBUG.DEBUG_OFF;
```

Parameters

None.

Usage Notes

The server does not handle this entrypoint specially. Therefore, it attempts to debug this entrypoint.

Debug Session Section

The following subprograms should be run in the debug session only:

- ATTACH_SESSION procedure
- SYNCHRONIZE function
- SHOW_SOURCE procedure
- PRINT_BACKTRACE procedure
- CONTINUE function
- SET_BREAKPOINT function
- DELETE_BREAKPOINT function
- DISABLE_BREAKPOINT function
- ENABLE_BREAKPOINT function
- SHOW_BREAKPOINTS procedure
- GET_VALUE function

- SET_VALUE function
- DETACH_SESSION procedure
- GET_RUNTIME_INFO function
- GET_INDEXES function
- EXECUTE procedure

ATTACH_SESSION procedure

This procedure notifies the debug session about the target program.

Syntax

```
DBMS_DEBUG.ATTACH_SESSION (
   debug_session_id  IN VARCHAR2,
   diagnostics       IN BINARY_INTEGER := 0);
```

Parameters

Table 7–8 ATTACH_SESSION Procedure Parameters

Parameter	Description
debug_session_id	Debug ID from a call to INITIALIZE in target session.
diagnostics	Generate diagnostic output if non-zero.

SYNCHRONIZE function

This function waits until the target program signals an event. If info_requested is not NULL, then it calls GET_RUNTIME_INFO.

Syntax

```
DBMS_DEBUG.SYNCHRONIZE (
   run_info       OUT  runtime_info,
   info_requested IN   BINARY_INTEGER := NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–9 SYNCHRONIZE Function Parameters

Parameter	Description
run_info	Structure in which to write information about the program. By default, this includes information about what program is running and at which line execution has paused.
info_requested	Optional bit-field in which to request information other than the default (which is info_getStackDepth + info_getLineInfo). 0 means that no information is requested at all. See "Information Flags" on page 7-10.

Returns

Table 7–10 SYNCHRONIZE Function Returns

Return	Description
success	
error_timeout	Timed out before the program started execution.
error_communication	Other communication error.

SHOW_SOURCE procedure

The best way to get the source code (for a program that is being run) is to use SQL. For example:

```
DECLARE
   info DBMS_DEBUG.runtime_info;
BEGIN
   -- call DBMS_DEBUG.SYNCHRONIZE, CONTINUE,
   -- or GET_RUNTIME_INFO to fill in 'info'
   SELECT text INTO <buffer> FROM all_source
   WHERE owner = info.Program.Owner
     AND name  = info.Program.Name
     AND line  = info.Line#;
END;
```

However, this does not work for non-persistent programs (for example, anonymous blocks and trigger invocation blocks). For non-persistent programs, call SHOW_

SOURCE. There are two flavors: one returns an indexed table of source lines, and the other returns a packed (and formatted) buffer.

There are two overloaded SHOW_SOURCE procedures.

Syntax

```
DBMS_DEBUG.SHOW_SOURCE (
   first_line  IN   BINARY_INTEGER,
   last_line   IN   BINARY_INTEGER,
   source      OUT  vc2_table);
```

Parameters

Table 7–11 SHOW_SOURCE Procedure Parameters

Parameter	Description
first_line	Line number of first line to fetch. (PL/SQL programs always start at line 1 and have no holes.)
last_line	Line number of last line to fetch. No lines are fetched past the end of the program.
source	The resulting table, which may be indexed by line#.

Returns

An indexed table of source-lines. The source lines are stored starting at first_line. If any error occurs, then the table is empty.

Usage Notes

This second overloading of SHOW_SOURCE returns the source in a formatted buffer, complete with line-numbers. It is faster than the indexed table version, but it does not guarantee to fetch all the source.

If the source does not fit in bufferlength (buflen), then additional pieces can be retrieved using the GET_MORE_SOURCE procedure (pieces returns the number of additional pieces that need to be retrieved).

Syntax

```
DBMS_DEBUG.SHOW_SOURCE (
   first_line   IN     BINARY_INTEGER,
   last_line    IN     BINARY_INTEGER,
   window       IN     BINARY_INTEGER,
   print_arrow  IN     BINARY_INTEGER,
   buffer       IN OUT VARCHAR2,
   buflen       IN     BINARY_INTEGER,
   pieces       OUT    BINARY_INTEGER);
```

Parameters

Table 7–12 SHOW_SOURCE Procedure Parameters

Parameter	Description
first_line	Smallest line-number to print.
last_line	Largest line-number to print.
window	'Window' of lines (the number of lines around the current source line).
print_arrow	Non-zero means to print an arrow before the current line.
buffer	Buffer in which to place the source listing.
buflen	Length of buffer.
pieces	Set to non-zero if not all the source could be placed into the given buffer.

PRINT_BACKTRACE procedure

This procedure prints a backtrace listing of the current execution stack. This should only be called if a program is currently running.

There are two overloaded PRINT_BACKTRACE procedures.

Syntax

```
DBMS_DEBUG.PRINT_BACKTRACE (
  listing IN OUT VARCHAR2);
```

Parameters

Table 7–13 PRINT_BACKTRACE Procedure Parameters

Parameter	Description
listing	A formatted character buffer with embedded newlines.

Syntax

```
DBMS_DEBUG.PRINT_BACKTRACE (
   backtrace OUT backtrace_table);
```

Parameters

Table 7–14 PRINT_BACKTRACE Procedure Parameters

Parameter	Description
backtrace	1-based indexed table of backtrace entries. The currently-running procedure is the last entry in the table (that is, the frame numbering is the same as that used by GET_VALUE). Entry 1 is the oldest procedure on the stack.

CONTINUE function

This function passes the given breakflags (a mask of the events that are of interest) to Probe in the target process. It tells Probe to continue execution of the target process, and it waits until the target process runs to completion or signals an event.

If info_requested is not NULL, then calls GET_RUNTIME_INFO.

Syntax

```
DBMS_DEBUG.CONTINUE (
   run_info       IN OUT runtime_info,
   breakflags     IN     BINARY_INTEGER,
   info_requested IN     BINARY_INTEGER := NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–15 CONTINUE Function Parameters

Parameter	Description
run_info	Information about the state of the program.
breakflags	Mask of events that are of interest. See "Breakflags" on page 7-9.
info_requested	Which information should be returned in run_info when the program stops. See "Information Flags" on page 7-10.

Returns

Table 7–16 CONTINUE Function Returns

Return	Description
success	
error_timeout	Timed out before the program started running.
error_communication	Other communication error.

SET_BREAKPOINT function

This function sets a breakpoint in a program unit, which persists for the current session. Execution pauses if the target program reaches the breakpoint.

Syntax

```
DBMS_DEBUG.SET_BREAKPOINT (
   program     IN  program_info,
   line#       IN  BINARY_INTEGER,
   breakpoint# OUT BINARY_INTEGER,
   fuzzy       IN  BINARY_INTEGER := 0,
   iterations  IN  BINARY_INTEGER := 0)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–17 SET_BREAKPOINT Function Parameters

Parameter	Description
program	Information about the program unit in which the breakpoint is to be set. (In version 2.1 and later, the namespace, name, owner, and dblink may be set to NULL, in which case the breakpoint is placed in the currently-running program unit.)
line#	Line at which the breakpoint is to be set.
breakpoint#	On successful completion, contains the unique breakpoint number by which to refer to the breakpoint.
fuzzy	Only applicable if there is no executable code at the specified line: 0 means return error_illegal_line. 1 means search forward for an adjacent line at which to place the breakpoint. -1 means search backwards for an adjacent line at which to place the breakpoint.
iterations	Number of times to wait before signalling this breakpoint.

Note: The fuzzy and iterations parameters are not yet implemented.

Returns

Table 7–18 SET_BREAKPOINT Function Returns

Return	Description
success	
error_illegal_line	Cannot set a breakpoint at that line.
error_bad_handle	No such program unit exists.

DELETE_BREAKPOINT function

This function deletes a breakpoint.

Syntax

```
DBMS_DEBUG.DELETE_BREAKPOINT (
   breakpoint IN BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–19 DELETE_BREAKPOINT Function Parameters

Parameter	Description
breakpoint	Breakpoint number from a previous call to SET_BREAKPOINT.

Returns

Table 7–20 DELETE_BREAKPOINT Function Returns

Return	Description
success	
error_no_such_breakpt	No such breakpoint exists.
error_idle_breakpt	Cannot delete an unused breakpoint.
error_stale_breakpt	The program unit was redefined since the breakpoint was set.

DISABLE_BREAKPOINT function

This function makes an existing breakpoint inactive, but it leaves it in place.

Syntax

```
DBMS_DEBUG.DISABLE_BREAKPOINT (
   breakpoint IN BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–21 DISABLE_BREAKPOINT Function Parameters

Parameter	Description
breakpoint	Breakpoint number from a previous call to SET_BREAKPOINT.

Returns

Table 7–22 DISABLE_BREAKPOINT Function Returns

Returns	Description
success	
error_no_such_breakpt	No such breakpoint exists.
error_idle_breakpt	Cannot disable an unused breakpoint.

ENABLE_BREAKPOINT function

This function is the reverse of disabling. This enables a previously disabled breakpoint.

Syntax

```
DBMS_DEBUG.ENABLE_BREAKPOINT (
   breakpoint IN BINARY_INTEGER)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–23 ENABLE_BREAKPOINT Function Parameters

Parameter	Description
breakpoint	Breakpoint number from a previous call to SET_BREAKPOINT.

Returns

Table 7–24 ENABLE_BREAKPOINT Function Returns

Return	Description
success	
error_no_such_breakpt	No such breakpoint exists.
error_idle_breakpt	Cannot enable an unused breakpoint.

SHOW_BREAKPOINTS procedure

This procedure returns a listing of the current breakpoints. There are two overloaded SHOW_BREAKPOINTS procedures.

Syntax

```
DBMS_DEBUG.SHOW_BREAKPOINTS (
   listing    IN OUT VARCHAR2);
```

Parameters

Table 7–25 SHOW_BREAKPOINTS Procedure Parameters

Parameter	Description
listing	A formatted buffer (including newlines) of the breakpoints.

Syntax

```
DBMS_DEBUG.SHOW_BREAKPOINTS (
   listing  OUT breakpoint_table);
```

Parameters

Table 7–26 SHOW_BREAKPOINTS Procedure Parameters

Parameter	Description
listing	Indexed table of breakpoint entries. The breakpoint number is indicated by the index into the table. Breakpoint numbers start at 1 and are reused when deleted.

GET_VALUE function

This function gets a value from the currently-running program. There are two overloaded GET_VALUE functions.

Syntax

```
DBMS_DEBUG.GET_VALUE (
   variable_name  IN  VARCHAR2,
   frame#         IN  BINARY_INTEGER,
   scalar_value   OUT VARCHAR2,
   format         IN  VARCHAR2 := NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–27 GET_VALUE Function Parameters

Parameter	Description
variable_name	Name of the variable or parameter.
frame#	Frame in which it lives; 0 means the current procedure.
scalar_value	Value.
format	Optional date format to use, if meaningful.

Returns

Table 7–28 GET_VALUE Function Returns

Return	Description
success	
error_bogus_frame	Frame does not exist.
error_no_debug_info	Entrypoint has no debug information.
error_no_such_object	variable_name does not exist in frame#.
error_unknown_type	The type information in the debug information is illegible.
error_nullvalue	Value is NULL.
error_indexed_table	The object is a table, but no index was provided.

This form of GET_VALUE is for fetching package variables. Instead of a frame#, it takes a handle, which describes the package containing the variable.

Syntax

```
DBMS_DEBUG.GET_VALUE (
   variable_name  IN  VARCHAR2,
   handle         IN  program_info,
   scalar_value   OUT VARCHAR2,
   format         IN  VARCHAR2 := NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–29 GET_VALUE Function Parameters

Parameter	Description
variable_name	Name of the variable or parameter.
handle	Description of the package containing the variable.
scalar_value	Value.
format	Optional date format to use, if meaningful.

Returns

Table 7–30 GET_VALUE Function Returns

Return	Description
error_no_such_object	Either: - Package does not exist. - Package is not instantiated. - User does not have privileges to debug the package. - Object does not exist in the package.
error_indexed_table	The object is a table, but no index was provided.

Example

This example illustrates how to get the value with a given package PACK in schema SCOTT, containing variable VAR:

```
DECLARE
   handle     dbms_debug.program_info;
   resultbuf  VARCHAR2(500);
   retval     BINARY_INTEGER;
BEGIN
   handle.Owner     := 'SCOTT';
   handle.Name      := 'PACK';
   handle.namespace := dbms_debug.namespace_pkgspec_or_toplevel;
   retval           := dbms_debug.get_value('VAR', handle, resultbuf, NULL);
END;
```

SET_VALUE function

This function sets a value in the currently-running program. There are two overloaded SET_VALUE functions.

Syntax

```
DBMS_DEBUG.SET_VALUE (
   frame#               IN binary_integer,
   assignment_statement IN varchar2)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–31 SET_VALUE Function Parameters

Parameter	Description
frame#	Frame in which the value is to be set; 0 means the currently executing frame.
assignment_statement	An assignment statement (which must be legal PL/SQL) to run in order to set the value. For example, 'x := 3;'.
	Only scalar values are supported in this release. The right side of the assignment statement must be a scalar.

Returns

Table 7–32 SET_VALUE Function Returns

Return	Description
success	
error_illegal_value	Not possible to set it to that value.
error_illegal_null	Cannot set to NULL because object type specifies it as 'not null'.
error_value_malformed	Value is not a scalar.
error_name_incomplete	The assignment statement does not resolve to a scalar. For example, 'x := 3;', if x is a record.

This form of SET_VALUE sets the value of a package variable.

Syntax

```
DBMS_DEBUG.SET_VALUE (
   handle               IN program_info,
   assignment_statement IN VARCHAR2)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–33 SET_VALUE Function Parameters

Parameter	Description
handle	Description of the package containing the variable.
assignment_statement	An assignment statement (which must be legal PL/SQL) to run in order to set the value. For example, 'x := 3;'.
	Only scalar values are supported in this release. The right side of the assignment statement must be a scalar.

Table 7–34 SET_VALUE Function Returns

Return	Description
error_no_such_object	Either:
	- Package does not exist.
	- Package is not instantiated.
	- User does not have privileges to debug the package.
	- Object does not exist in the package.

In some cases, the PL/SQL compiler uses temporaries to access package variables, and Probe does not guarantee to update such temporaries. It is possible, although unlikely, that modification to a package variable using SET_VALUE might not take effect for a line or two.

Example

To set the value of SCOTT.PACK.var to 6:

```
DECLARE
   handle  dbms_debug.program_info;
   retval  BINARY_INTEGER;
BEGIN
   handle.Owner     := 'SCOTT';
   handle.Name      := 'PACK';
   handle.namespace := dbms_debug.namespace_pkgspec_or_toplevel;
   retval           := dbms_debug.set_value(handle, 'var := 6;');
END;
```

DETACH_SESSION procedure

This procedure stops debugging the target program. This procedure may be called at any time, but it does not notify the target session that the debug session is detaching itself, and it does not abort execution of the target session. Therefore, care should be taken to ensure that the target session does not hang itself.

Syntax

```
DBMS_DEBUG.DETACH_SESSION;
```

Parameters

None.

GET_RUNTIME_INFO function

This function returns information about the current program. It is only needed if the info_requested parameter to SYNCHRONIZE or CONTINUE was set to 0.

> **Note:** This is currently only used by client-side PL/SQL.

Syntax

```
DBMS_DEBUG.GET_RUNTIME_INFO (
   info_requested  IN  BINARY_INTEGER,
   run_info        OUT runtime_info)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–35 GET_RUNTIME_INFO Function Parameters

Parameter	Description
info_requested	Which information should be returned in run_info when the program stops. See "Information Flags" on page 7-10.
run_info	Information about the state of the program.

GET_INDEXES function

Given a name of a variable or parameter, this function returns the set of its indexes, if it is an indexed table. An error is returned if it is not an indexed table.

Syntax

```
DBMS_DEBUG.GET_INDEXES (
   varname   IN  VARCHAR2,
   frame#    IN  BINARY_INTEGER,
   handle    IN  program_info,
   entries   OUT index_table)
  RETURN BINARY_INTEGER;
```

Parameters

Table 7–36 GET_INDEXES Function Parameters

Parameter	Description
varname	Name of the variable to get index information about.
frame#	Number of frame in which the variable or parameter resides; NULL for a package variable.
handle	Package description, if object is a package variable.
entries	1-based table of the indexes. If non-NULL, then entries(1) contains the first index of the table, entries(2) contains the second index, and so on.

Returns

Table 7–37 GET_INDEXES Function Returns

Return	Description
error_no_such_object	Either: - The package does not exist. - The package is not instantiated. - The user does not have privileges to debug the package. - The object does not exist in the package.

EXECUTE procedure

This procedure executes SQL or PL/SQL code in the target session. The target session is assumed to be waiting at a breakpoint (or other event). The call to DBMS_DEBUG.EXECUTE occurs in the debug session, which then asks the target session to execute the code.

Syntax

```
DBMS_DEBUG.EXECUTE (
   what         IN VARCHAR2,
   frame#       IN BINARY_INTEGER,
   bind_results IN BINARY_INTEGER,
   results      IN OUT NOCOPY dbms_debug_vc2coll,
   errm         IN OUT NOCOPY VARCHAR2);
```

Parameters

Table 7–38 EXECUTE Procedure Parameters

Parameter	Description
what	SQL or PL/SQL source to execute.
frame#	The context in which to execute the code. Only -1 (global context) is supported at this time.
bind_results	Whether the source wants to bind to results in order to return values from the target session. 0 = No 1 = Yes

Table 7–38 EXECUTE Procedure Parameters

Parameter	Description
results	Collection in which to place results, if bind_results is not 0.
errm	Error message, if an error occurred; otherwise, NULL.

Example 1

This example executes a SQL statement. It returns no results.

```
DECLARE
   coll sys.dbms_debug_vc2coll; -- results (unused)
   errm VARCHAR2(100);
BEGIN
   dbms_debug.execute('insert into emp(ename,empno,deptno) ' ||
                      'values(''LJE'', 1, 1)',
                      -1, 0, coll, errm);
END;
```

Example 2

This example executes a PL/SQL block, and it returns no results. The block is an autonomous transaction, which means that the value inserted into the table becomes visible in the debug session.

```
DECLARE
   coll sys.dbms_debug_vc2coll;
   errm VARCHAR2(100);
BEGIN
   dbms_debug.execute(
       'DECLARE PRAGMA autonomous_transaction; ' ||
       'BEGIN ' ||
       '   insert into emp(ename, empno, deptno) ' ||
       '   values(''LJE'', 1, 1); ' ||
       ' COMMIT; ' ||
       'END;',
       -1, 0, coll, errm);
END;
```

Example 3

This example executes a PL/SQL block, and it returns some results.

```
DECLARE
   coll sys.dbms_debug_vc2coll;
   errm VARCHAR2(100);
BEGIN
   dbms_debug.execute(
      'DECLARE ' ||
      '   pp SYS.dbms_debug_vc2coll := SYS.dbms_debug_vc2coll(); ' ||
      '   x  PLS_INTEGER; ' ||
      '   i  PLS_INTEGER := 1; ' ||
      'BEGIN ' ||
      '   SELECT COUNT(*) INTO x FROM emp; ' ||
      '   pp.EXTEND(x * 6); ' ||
      '   FOR c IN (SELECT * FROM emp) LOOP ' ||
      '      pp(i) := ''Ename: '' || c.ename; i := i+1; ' ||
      '      pp(i) := ''Empno: '' || c.empno; i := i+1; ' ||
      '      pp(i) := ''Job:   '' || c.job;   i := i+1; ' ||
      '      pp(i) := ''Mgr:   '' || c.mgr;   i := i+1; ' ||
      '      pp(i) := ''Sal:   '' || c.sal;   i := i+1; ' ||
      '      pp(i) := null;                   i := i+1; ' ||
      '   END LOOP; ' ||
      '   :1 := pp;' ||
      'END;',
       -1, 1, coll, errm);
   each := coll.FIRST;
   WHILE (each IS NOT NULL) LOOP
      dosomething(coll(each));
      each := coll.NEXT(each);
   END LOOP;
END;
```

8

DBMS_DEFER

DBMS_DEFER is the user interface to a replicated transactional deferred remote procedure call facility. Replicated applications use the calls in this interface to queue procedure calls for later transactional execution at remote nodes.

These routines are typically called from either after row triggers or application specified update procedures.

Summary of Subprograms

Table 8–1 DBMS_DEFER Package Subprograms

Subprogram	Description
CALL procedure on page 8-2	Builds a deferred call to a remote procedure.
COMMIT_WORK procedure on page 8-4	Performs a transaction commit after checking for well-formed deferred remote procedure calls.
datatype_ARG procedure on page 8-4	Provides the data that is to be passed to a deferred remote procedure call.
TRANSACTION procedure on page 8-5	Indicates the start of a new deferred transaction.

CALL procedure

This procedure builds a deferred call to a remote procedure.

Syntax

```
DBMS_DEFER.CALL (
   schema_name       IN   VARCHAR2,
   package_name      IN   VARCHAR2,
   proc_name         IN   VARCHAR2,
   arg_count         IN   NATURAL,
   { nodes           IN   node_list_t,
   | group_name      IN   VARCHAR2 :=''});
```

Parameters

Table 8–2 CALL Procedure Parameters

Parameter	Description
schema_name	Name of the schema in which the stored procedure is located.
package_name	Name of the package containing the stored procedure. The stored procedure must be part of a package. Deferred calls to standalone procedures are not supported.
proc_name	Name of the remote procedure to which you want to defer a call.
arg_count	Number of parameters for the procedure. You must have one call to DBMS_DEFER.datatype_ARG for each of these parameters.
nodes	A PL/SQL table of fully qualified database names to which you want to propagate the deferred call. The table is indexed starting at position 1 and ending when a NULL entry is found, or the NO_DATA_FOUND exception is raised. The data in the table is case insensitive. This argument is optional.
group_name	Reserved for internal use.

Note: The CALL procedure is overloaded. The nodes and group_name parameters are mutually exclusive.

Exceptions

Table 8–3 CALL Procedure Exceptions

Exception	Description
ORA-23304 (malformedcall)	Previous call was not correctly formed.
ORA-23319	Parameter value is not appropriate.
ORA-23352	Destination list (specified by nodes or by a previous DBMS_DEFER.TRANSACTION call) contains duplicates.

COMMIT_WORK procedure

This procedure performs a transaction commit after checking for well-formed deferred remote procedure calls.

Syntax

```
DBMS_DEFER.COMMIT_WORK (
   commit_work_comment IN VARCHAR2);
```

Parameters

Table 8–4 COMMIT_WORK Procedure Parameters

Parameter	Description
commit_work_comment	Equivalent to SQL COMMIT COMMENT statement.

Exceptions

Table 8–5 COMMIT_WORK Procedure Exceptions

Exception	Description
ORA-23304 (malformedcall)	Transaction was not correctly formed or terminated.

*datatype*_ARG procedure

This procedure provides the data that is to be passed to a deferred remote procedure call. Depending upon the type of the data that you need to pass to a procedure, you must call one of the following procedures for each argument to the procedure.

Syntax

```
DBMS_DEFER.NUMBER_ARG       (arg  IN NUMBER);
DBMS_DEFER.DATE_ARG         (arg  IN DATE);
DBMS_DEFER.VARCHAR2_ARG     (arg  IN VARCHAR2);
DBMS_DEFER.CHAR_ARG         (arg  IN CHAR);
DBMS_DEFER.ROWID_ARG        (arg  IN ROWID);
DBMS_DEFER.RAW_ARG          (arg  IN RAW);
DBMS_DEFER.BLOB_ARG         (arg  IN BLOB);
DBMS_DEFER.CLOB_ARG         (arg  IN CLOB);
DBMS_DEFER.NCLOB_ARG        (arg  IN NCLOB);
DBMS_DEFER.NCHAR_ARG        (arg  IN NCHAR);
DBMS_DEFER.NVARCHAR2_ARG    (arg  IN NVARCHAR2);
DBMS_DEFER.ANY_CLOB_ARG     (arg  IN CLOB);
DBMS_DEFER.ANY_VARCHAR2_ARG (arg  IN VARCHAR2);
DBMS_DEFER.ANY_CHAR_ARG     (arg  IN CHAR);
```

Parameters

Table 8–6 datatype_ARG Procedure Parameters

Parameter	Description
arg	Value of the parameter that you want to pass to the remote procedure to which you previously deferred a call.

Exceptions

Table 8–7 datatype_ARG Procedure Exceptions

Exception	Description
ORA-23323	Argument value is too long.

TRANSACTION procedure

This procedure indicates the start of a new deferred transaction. If you omit this call, then Oracle considers your first call to DBMS_DEFER.CALL to be the start of a new transaction.

Syntax

```
DBMS_DEFER.TRANSACTION (
   nodes  IN   node_list_t);
```

Parameters

Table 8–8 TRANSACTION Procedure Parameters

Parameter	Description
nodes	A PL/SQL table of fully qualified database names to which you want to propagate the deferred calls of the transaction. The table is indexed starting at position 1 until a NULL entry is found, or the NO_DATA_FOUND exception is raised. The data in the table is case insensitive.

Exceptions

Table 8–9 TRANSACTION Procedure Exceptions

Exception	Description
ORA-23304 (malformedcall)	Previous transaction was not correctly formed or terminated.
ORA-23319	Parameter value is not appropriate.
ORA-23352	Raised by DBMS_DEFER.CALL if the node list contains duplicates.

Usage Notes

The TRANSACTION procedure is overloaded. The behavior of the version without an input parameter is similar to that of the version with an input parameter, except that the former uses the nodes in the DEFDEFAULTDEST view instead of using the nodes in the nodes parameter.

9

DBMS_DEFER_QUERY

DBMS_DEFER_QUERY enables querying the deferred RPC queue data that is not exposed through views.

Summary of Subprograms

Table 9–1 DBMS_DEFER_QUERY Package Subprograms

Subprogram	Description
GET_ARG_FORM function on page 9-2	Determines the form of an argument in a deferred call.
GET_ARG_TYPE function on page 9-3	Determines the type of an argument in a deferred call.
GET_CALL_ARGS procedure on page 9-5	Returns the text version of the various arguments for the given call.
GET_datatype_ARG function on page 9-6	Determines the value of an argument in a deferred call.

GET_ARG_FORM function

This function determines the form of an argument in a deferred call. This function returns the character set ID of a deferred call parameter.

> **See Also:** For more about displaying deferred transactions, see "Displaying Deferred Transactions" in the *Oracle8i Replication* manual. For more information about displaying error transactions, see "Displaying Error Transactions" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_DEFER_QUERY.GET_ARG_FORM (
   callno                IN   NUMBER,
   arg_no                IN   NUMBER,
   deferred_tran_id      IN   VARCHAR2)
  RETURN NUMBER;
```

Parameters

Table 9–2 GET_ARG_FORM Function Parameters

Parameter	Description
callno	Call identifier from the DEFCALL view.
arg_no	Position of desired parameter in calls argument list. Parameter positions are 1..*number* of parameters in call.

Table 9–2 GET_ARG_FORM Function Parameters

Parameter	Description
deferred_tran_id	Deferred transaction ID.

Exceptions

Table 9–3 GET_ARG_FORM Function Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.

Returns

Table 9–4 GET_ARG_Form Function Returns

Return	Corresponding Datatype
1	CHAR, VARCHAR2, CLOB
2	NCHAR, NVARCHAR2, NCLOB

GET_ARG_TYPE function

This function determines the type of an argument in a deferred call. The type of the deferred RPC parameter is returned.

> **See Also:** For more about displaying deferred transactions, see "Displaying Deferred Transactions" in the *Oracle8i Replication* manual. For more information about displaying error transactions, see "Displaying Error Transactions" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_DEFER_QUERY.GET_ARG_TYPE (
   callno            IN   NUMBER,
   arg_no            IN   NUMBER,
   deferred_tran_id  IN   VARCHAR2)
  RETURN NUMBER;
```

Parameters

Table 9–5 GET_ARG_TYPE Function Parameters

Parameter	Description
callno	ID number from the DEFCALL view of the deferred remote procedure call.
arg_no	Numerical position of the argument to the call whose type you want to determine. The first argument to a procedure is in position 1.
deferred_tran_id	Identifier of the deferred transaction.

Exceptions

Table 9–6 GET_ARG_TYPE Function Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.

Returns

Table 9–7 GET_ARG_TYPE Function Returns

Return	Corresponding Datatype
1	VARCHAR2
2	NUMBER
11	ROWID
12	DATE
23	RAW
96	CHAR
112	CLOB
113	BLOB

GET_CALL_ARGS procedure

This procedure returns the text version of the various arguments for the given call. The text version is limited to the first 2000 bytes.

Syntax

```
DBMS_DEFER_QUERY.GET_CALL_ARGS (
   callno    IN  NUMBER,
   startarg  IN  NUMBER := 1,
   argcnt    IN  NUMBER,
   argsize   IN  NUMBER,
   tran_id   IN  VARCHAR2,
   date_fmt  IN  VARCHAR2,
   types     OUT TYPE_ARY,
   forms     OUT TYPE_ARY,
   vals      OUT VAL_ARY);
```

Parameters

Table 9–8 GET_CALL_ARGS Procedure Parameters

Parameter	Description
callno	ID number from the DEFCALL view of the deferred RPC.
startarg	Numerical position of the first argument you want described.
argcnt	Number of arguments in the call.
argsize	Maximum size of returned argument.
tran_id	Identifier of the deferred transaction.
date_fmt	Format in which the date should be returned.
types	Array containing the types of arguments.
forms	Array containing the character set forms of arguments.
vals	Array containing the values of the arguments in a textual form.

Exceptions

Table 9–9 GET_CALL_ARGS Procedure Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.

GET_*datatype*_ARG function

This function determines the value of an argument in a deferred call.

> **See Also:** For more about displaying deferred transactions, see "Displaying Deferred Transactions" in the *Oracle8i Replication* manual. For more information about displaying error transactions, see "Displaying Error Transactions" in the *Oracle8i Replication* manual.

Syntax

Depending upon the type of the argument value that you want to retrieve, the syntax for the appropriate function is as follows. Each of these functions returns the value of the specified argument.

```
DBMS_DEFER_QUERY.GET_datatype_ARG (
   callno             IN   NUMBER,
   arg_no             IN   NUMBER,
   deferred_tran_id   IN   VARCHAR2 DEFAULT NULL)
  RETURN datatype;
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| ROWID
| BLOB
| CLOB
| NCLOB
| NCHAR
| NVARCHAR2 }
```

Parameters

Table 9–10 GET_datatype_ARG Function Parameters

Parameter	Description
callno	ID number from the DEFCALL view of the deferred remote procedure call.
arg_no	Numerical position of the argument to the call whose value you want to determine. The first argument to a procedure is in position one.
deferred_tran_id	Identifier of the deferred transaction. Defaults to the last transaction identifier passed to GET_ARG_TYPE. The default is NULL.

Exceptions

Table 9–11 GET_datatype_ARG Function Exceptions

Exception	Description
NO_DATA_FOUND	Input parameters do not correspond to a parameter of a deferred call.
ORA-26564	Argument in this position is not of the specified type.

10

DBMS_DEFER_SYS

DBMS_DEFER_SYS procedures manage default replication node lists.

This package is the system administrator interface to a replicated transactional deferred remote procedure call facility. Administrators and replication daemons can execute transactions queued for remote nodes using this facility, and administrators can control the nodes to which remote calls are destined.

Summary of Subprograms

Table 10–1 DBMS_DEFER_SYS Package Subprograms

Subprogram	Description
ADD_DEFAULT_DEST procedure on page 10-3	Adds a destination database to the DEFDEFAULTDEST view.
DELETE_DEFAULT_DEST procedure on page 10-3	Removes a destination database from the DEFDEFAULTDEST view.
DELETE_DEF_DESTINATION procedure on page 10-4	Removes a destination database from the DEFSCHEDULE view.
DELETE_ERROR procedure on page 10-4	Deletes a transaction from the DEFERROR view.
DELETE_TRAN procedure on page 10-5	Deletes a transaction from the DEFTRANDEST view.
DISABLED function on page 10-6	Determines whether propagation of the deferred transaction queue from the current site to a given site is enabled.
EXCLUDE_PUSH function on page 10-6	Acquires an exclusive lock that prevents deferred transaction PUSH.
EXECUTE_ERROR procedure on page 10-7	Re-executes a deferred transaction that did not initially complete successfully.
EXECUTE_ERROR_AS_USER procedure on page 10-8	Re-executes a deferred transaction that did not initially complete successfully.
PURGE function on page 10-9	Purges pushed transactions from the deferred transaction queue at your current master or snapshot site.
PUSH function on page 10-11	Forces a deferred remote procedure call queue at your current master or snapshot site to be pushed to another master site.
REGISTER_PROPAGATOR procedure on page 10-13	Registers the given user as the propagator for the local database.
SCHEDULE_PURGE procedure on page 10-14	Schedules a job to purge pushed transactions from the deferred transaction queue at your current master or snapshot site.
SCHEDULE_PUSH procedure on page 10-15	Schedules a job to push the deferred transaction queue to a remote master destination.
SET_DISABLED procedure on page 10-17	Disables or enables propagation of the deferred transaction queue from the current site to a given destination site.

Table 10–1 DBMS_DEFER_SYS Package Subprograms

Subprogram	Description
UNREGISTER_PROPAGATOR procedure on page 10-18	Unregister a user as the propagator from the local database.
UNSCHEDULE_PURGE procedure on page 10-19	Stops automatic purges of pushed transactions from the deferred transaction queue at a snapshot or master site.
UNSCHEDULE_PUSH procedure on page 10-19	Stops automatic pushes of the deferred transaction queue from a snapshot or master site to another master site.

ADD_DEFAULT_DEST procedure

This procedure adds a destination database to the DEFDEFAULTDEST view.

Syntax

```
DBMS_DEFER_SYS.ADD_DEFAULT_DEST (
   dblink   IN   VARCHAR2);
```

Parameters

Table 10–2 ADD_DEFAULT_DEST Procedure Parameters

Parameter	Description
dblink	The fully qualified database name of the node that you want to add to the DEFDEFAULTDEST view.

Exceptions

Table 10–3 ADD_DEFAULT_DEST Procedure Exceptions

Exception	Description
ORA-23352	The dblink that you specified is already in the default list.

DELETE_DEFAULT_DEST procedure

This procedure removes a destination database from the DEFDEFAULTDEST view.

Syntax

```
DBMS_DEFER_SYS.DELETE_DEFAULT_DEST (
   dblink   IN   VARCHAR2);
```

Parameters

Table 10–4 DELETE_DEFAULT_DEST Procedure Parameters

Parameter	Description
dblink	The fully qualified database name of the node that you want to delete from the DEFDEFAULTDEST view. If Oracle does not find this dblink in the view, then no action is taken.

DELETE_DEF_DESTINATION procedure

This procedure removes a destination database from the DEFSCHEDULE view.

Syntax

```
DBMS_DEFER_SYS.DELETE_DEF_DESTINATION (
   destination   IN   VARCHAR2,
   force         IN   BOOLEAN := FALSE);
```

Parameters

Table 10–5 DELETE_DEF_DESTINATION Procedure Parameters

Parameter	Description
destination	The fully qualified database name of the destination that you want to delete from the DefSchedule view. If Oracle does not find this destination in the view, then no action is taken.
force	When set to TRUE, Oracle ignores all safety checks and deletes the destination.

DELETE_ERROR procedure

To delete a transaction from the DEFERROR view.

Syntax

```
DBMS_DEFER_SYS.DELETE_ERROR(
   deferred_tran_id     IN   VARCHAR2,
   destination          IN   VARCHAR2);
```

Parameters

Table 10–6 DELETE_ERROR Procedure Parameters

Parameter	Description
deferred_tran_id	ID number from the DEFERROR view of the deferred transaction that you want to remove from the DEFERROR view. If this parameter is NULL, then all transactions meeting the requirements of the other parameter are removed.
destination	The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. If this parameter is NULL, then all transactions meeting the requirements of the other parameter are removed from the DEFERROR view.

DELETE_TRAN procedure

To delete a transaction from the DEFTRANDEST view. If there are no other DEFTRANDEST or DEFERROR entries for the transaction, then the transaction is deleted from the DEFTRAN and DEFCALL views as well.

Syntax

```
DBMS_DEFER_SYS.DELETE_TRAN (
   deferred_tran_id     IN   VARCHAR2,
   destination          IN   VARCHAR2);
```

Parameters

Table 10–7 DELETE_TRAN Procedure Parameters

Parameter	Description
deferred_tran_id	ID number from the DEFTRAN view of the deferred transaction that you want to delete. If this is NULL, then all transactions meeting the requirements of the other parameter are deleted.
destination	The fully qualified database name from the DEFTRANDEST view of the database to which the transaction was originally queued. If this is NULL, then all transactions meeting the requirements of the other parameter are deleted.

DISABLED function

To determine whether propagation of the deferred transaction queue from the current site to a given site is enabled. The DISABLED function returns TRUE if the deferred remote procedure call (RPC) queue is disabled for the given destination.

Syntax

```
DBMS_DEFER_SYS.DISABLED (
   destination  IN   VARCHAR2)
  RETURN BOOLEAN;
```

Parameters

Table 10–8 DISABLED Function Parameters

Parameter	Description
destination	The fully qualified database name of the node whose propagation status you want to check.

Returns

Table 10–9 DISABLED Function Returns

Return	Description
TRUE	Propagation to this site from the current site is disabled.
FALSE	Propagation to this site from the current site is enabled.

Exceptions

Table 10–10 DISABLED Function Exceptions

Exception	Description
NO_DATA_FOUND	DESTINATION does not appear in the DEFSCHEDULE view.

EXCLUDE_PUSH function

To acquire an exclusive lock that prevents deferred transaction PUSH (either serial or parallel). This function does a commit when acquiring the lock. The lock is acquired with RELEASE_ON_COMMIT => TRUE, so that pushing of the deferred transaction queue can resume after the next commit.

Syntax

```
DBMS_DEFER_SYS.EXCLUDE_PUSH (
   timeout   IN   INTEGER)
  RETURN INTEGER;
```

Parameters

Table 10–11 EXCLUDE_PUSH Function Parameters

Parameter	Description
timeout	Timeout in seconds. If the lock cannot be acquired within this time period (either because of an error or because a PUSH is currently under way), then the call returns a value of 1. A timeout value of DBMS_LOCK.MAXWAIT waits indefinitely.

Returns

Table 10–12 EXCLUDE_PUSH Function Returns

Return	Description
0	Success, lock acquired.
1	Timeout, no lock acquired.
2	Deadlock, no lock acquired.
4	Already own lock.

EXECUTE_ERROR procedure

To reexecute a deferred transaction that did not initially complete successfully. This procedure raises an ORA-24275 error when illegal combinations of NULL and non-NULL parameters are used.

Syntax

```
DBMS_DEFER_SYS.EXECUTE_ERROR (
   deferred_tran_id IN   VARCHAR2,
   destination      IN   VARCHAR2);
```

Parameters

Table 10–13 EXECUTE_ERROR Procedure Parameters

Parameter	Description
deferred_tran_id	ID number from the DEFERROR view of the deferred transaction that you want to re-execute. If this is NULL, then all transactions queued for destination are re-executed.
destination	The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. This must not be NULL.

Exceptions

Table 10–14 EXECUTE_ERROR Procedure Exceptions

Exception	Description
badparam	Parameter value missing or invalid (for example, if destination is NULL).
missinguser	Invalid user.

EXECUTE_ERROR_AS_USER procedure

To reexecute a deferred transaction that did not initially complete successfully. Each transaction is executed in the security context of the connected user. This procedure raises an ORA-24275 error when illegal combinations of NULL and non-NULL parameters are used.

Syntax

```
DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER (
   deferred_tran_id IN   VARCHAR2,
   destination      IN   VARCHAR2);
```

Parameters

Table 10–15 EXECUTE_ERROR_AS_USER Procedure Parameters

Parameter	Description
deferred_tran_id	ID number from the DEFERROR view of the deferred transaction that you want to re-execute. If this is NULL, then all transactions queued for destination are re-executed.

Table 10–15 EXECUTE_ERROR_AS_USER Procedure Parameters

Parameter	Description
destination	The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. This must not be NULL.

Exceptions

Table 10–16 EXECUTE_ERROR_AS_USER Procedure Exceptions

Exception	Description
badparam	Parameter value missing or invalid (for example, if destination is NULL).
missinguser	Invalid user.

PURGE function

To purge pushed transactions from the deferred transaction queue at your current master or snapshot site.

Syntax

```
DBMS_DEFER_SYS.PURGE (
   purge_method         IN  BINARY_INTEGER := purge_method_quick,
   rollback_segment     IN  VARCHAR2       := NULL,
   startup_seconds      IN  BINARY_INTEGER := 0,
   execution_seconds    IN  BINARY_INTEGER := seconds_infinity,
   delay_seconds        IN  BINARY_INTEGER := 0,
   transaction_count    IN  BINARY_INTEGER := transactions_infinity,
   write_trace          IN  BOOLEAN        := NULL);
  RETURN BINARY_INTEGER;
```

Parameters

Table 10–17 PURGE Function Parameters

Parameter	Description
purge_method	Controls how to purge the deferred transaction queue; purge_method_quick costs less, while purge_method_precise offers better precision.

Table 10–17 PURGE Function Parameters

Parameter	Description
rollback_segment	Name of rollback segment to use for the purge, or NULL for default.
startup_seconds	Maximum number of seconds to wait for a previous purge of the same deferred transaction queue.
execution_seconds	If >0, then stop purge cleanly after the specified number of seconds of real time.
delay_seconds	Stop purge cleanly after the deferred transaction queue has no transactions to purge for delay_seconds.
transaction_count	If > 0, then shutdown cleanly after purging transaction_count number of transactions.
write_trace	When set to TRUE, Oracle records the result value returned by the PURGE function in the server's trace file.

Returns

Table 10–18 PURGE Function Returns

Return	Description
0	OK, terminated after delay_seconds expired.
1	Terminated by lock timeout while starting.
2	Terminated by exceeding execution_seconds.
3	Terminated by exceeding transaction_count.
5	Terminated after errors.

Exceptions

Table 10–19 PURGE Function Exceptions

Exception	Description
argoutofrange	Parameter value is out of a valid range.
executiondisabled	Execution of purging is disabled.
defererror	Internal error.

PUSH function

This function forces a deferred remote procedure call queue at your current master or snapshot site to be pushed (executed, propagated) to another master site using either serial or parallel propagation.

Syntax

```
DBMS_DEFER_SYS.PUSH (
   destination          IN  VARCHAR2,
   parallelism          IN  BINARY_INTEGER := 0,
   heap_size            IN  BINARY_INTEGER := 0,
   stop_on_error        IN  BOOLEAN        := FALSE,
   write_trace          IN  BOOLEAN        := FALSE,
   startup_seconds      IN  BINARY_INTEGER := 0,
   execution_seconds    IN  BINARY_INTEGER := seconds_infinity,
   delay_seconds        IN  BINARY_INTEGER := 0,
   transaction_count    IN  BINARY_INTEGER := transactions_infinity,
   delivery_order_limit IN  NUMBER         := delivery_order_infinity)
  RETURN BINARY_INTEGER;
```

Parameters

Table 10–20 PUSH Function Parameters

Parameter	Description
destination	The fully qualified database name of the master to which you are forwarding changes.
parallelism	0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set the parameter unless so directed by Oracle Worldwide Support.
stop_on_error	The default, FALSE, indicates that the executor should continue even if errors, such as conflicts, are encountered. If TRUE, then shutdown (cleanly if possible) at the first indication that a transaction encountered an error at the destination site.
write_trace	When set to TRUE, Oracle records the result value returned by the function in the server's trace file.

Table 10–20 PUSH Function Parameters

Parameter	Description
startup_seconds	Maximum number of seconds to wait for a previous push to the same destination.
execution_seconds	If >0, then stop push cleanly after the specified number of seconds of real time. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue.
delay_seconds	Do not return before the specified number of seconds have elapsed, even if the queue is empty. Useful for reducing execution overhead if PUSH is called from a tight loop.
transaction_count	If > 0, then the maximum number of transactions to be pushed before stopping. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue that need to be pushed.
delivery_order_limit	Stop execution cleanly before pushing a transaction where delivery_order >= delivery_order_limit

Returns

Table 10–21 PUSH Function Returns

Return	Description
0	OK, terminated after delay_seconds expired.
1	Terminated by lock timeout while starting.
2	Terminated by exceeding execution_seconds.
3	Terminated by exceeding transaction_count.
4	Terminated by exceeding delivery_order_limit.
5	Terminated after errors.

Exceptions

Table 10–22 PUSH Function Exceptions

Exception	Description
deferror incompleteparallelpush	Serial propagation requires that parallel propagation shuts down cleanly.
executiondisabled	Execution of deferred RPCs is disabled at the destination.
crt_err_err	Error while creating entry in DEFERROR.
deferred_rpc_qiesce	Replication activity for object group is suspended.
commfailure	Communication failure during deferred RPC.
missingpropator	A propagator does not exist.

REGISTER_PROPAGATOR procedure

This procedure registers the given user as the propagator for the local database. It also grants to the given user CREATE SESSION, CREATE PROCEDURE, CREATE DATABASE LINK, and EXECUTE ANY PROCEDURE privileges (so that the user can create wrappers).

Syntax

```
DBMS_DEFER_SYS.REGISTER_PROPAGATOR (
   username  IN  VARCHAR2);
```

Parameters

Table 10–23 REGISTER_PROPAGATOR Procedure Parameters

Parameter	Description
username	Name of the user.

Exceptions

Table 10–24 REGISTER_PROPAGATOR Procedure Exceptions

Exception	Description
missinguser	Given user does not exist.
alreadypropagator	Given user is already the propagator.

Table 10–24 REGISTER_PROPAGATOR Procedure Exceptions

Exception	Description
duplicatepropagator	There is already a different propagator.

SCHEDULE_PURGE procedure

This procedure schedules a job to purge pushed transactions from the deferred transaction queue at your current master or snapshot site. You should schedule one purge job.

Syntax

```
DBMS_DEFER_SYS.SCHEDULE_PURGE (
   interval             IN  VARCHAR2,
   next_date            IN  DATE,
   reset                IN  BOOLEAN        := NULL,
   purge_method         IN  BINARY_INTEGER := NULL,
   rollback_segment     IN  VARCHAR2       := NULL,
   startup_seconds      IN  BINARY_INTEGER := NULL,
   execution_seconds    IN  BINARY_INTEGER := NULL,
   delay_seconds        IN  BINARY_INTEGER := NULL,
   transaction_count    IN  BINARY_INTEGER := NULL,
   write_trace          IN  BOOLEAN        := NULL);
```

Parameters

Table 10–25 SCHEDULE_PURGE Procedure Parameters

Parameter	Description
interval	Allows you to provide a function to calculate the next time to purge. This value is stored in the `interval` field of the `DEFSCHEDULE` view and calculates the `next_date` field of this view. If you use the default value for this parameter, `NULL`, then the value of this field remains unchanged. If the field had no previous value, then it is created with a value of `NULL`. If you do not supply a value for this field, then you must supply a value for `next_date`.

Table 10–25 SCHEDULE_PURGE Procedure Parameters

Parameter	Description
next_date	Allows you to specify a given time to purge pushed transactions from the site's queue. This value is stored in the next_date field of the DEFSCHEDULE view. If you use the default value for this parameter, NULL, then the value of this field remains unchanged. If this field had no previous value, then it is created with a value of NULL. If you do not supply a value for this field, then you must supply a value for interval.
reset	Set to TRUE to reset LAST_TXN_COUNT, LAST_ERROR, and LAST_MSG to NULL.
purge_method	Controls how to purge the deferred transaction queue; purge_method_quick costs less, while purge_method_precise offers better precision.
rollback_segment	Name of rollback segment to use for the purge, or NULL for default.
startup_seconds	Maximum number of seconds to wait for a previous purge of the same deferred transaction queue.
execution_seconds	If >0, then stop purge cleanly after the specified number of seconds of real time.
delay_seconds	Stop purge cleanly after the deferred transaction queue has no transactions to purge for delay_seconds.
transaction_count	If > 0, then shutdown cleanly after purging transaction_count number of transactions.
write_trace	When set to TRUE, Oracle records the result value returned by the PURGE function in the server's trace file.

SCHEDULE_PUSH procedure

This procedure schedules a job to push the deferred transaction queue to a remote master destination. This procedure does a COMMIT.

Syntax

```
DBMS_DEFER_SYS.SCHEDULE_PUSH (
   destination          IN  VARCHAR2,
   interval             IN  VARCHAR2,
   next_date            IN  DATE,
   reset                IN  BOOLEAN        := FALSE,
   parallelism          IN  BINARY_INTEGER := NULL,
   heap_size            IN  BINARY_INTEGER := NULL,
   stop_on_error        IN  BOOLEAN        := NULL,
   write_trace          IN  BOOLEAN        := NULL,
   startup_seconds      IN  BINARY_INTEGER := NULL,
   execution_seconds    IN  BINARY_INTEGER := NULL,
   delay_seconds        IN  BINARY_INTEGER := NULL,
   transaction_count    IN  BINARY_INTEGER := NULL);
```

Parameters

Table 10–26 SCHEDULE_PUSH Procedure Parameters

Parameter	Description
destination	The fully qualified database name of the master to which you are forwarding changes.
interval	Allows you to provide a function to calculate the next time to push. This value is stored in the interval field of the DEFSCHEDULE view and calculates the next_date field of this view. If you use the default value for this parameter, NULL, then the value of this field remains unchanged. If the field had no previous value, then it is created with a value of NULL. If you do not supply a value for this field, then you must supply a value for next_date.
next_date	Allows you to specify a given time to push deferred transactions to the master site destination. This value is stored in the next_date field of the DEFSCHEDULE view. If you use the default value for this parameter, NULL, then the value of this field remains unchanged. If this field had no previous value, then it is created with a value of NULL. If you do not supply a value for this field, then you must supply a value for interval.
reset	Set to TRUE to reset LAST_TXN_COUNT, LST_ERROR, and LAST_MSG to NULL.
parallelism	0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process.

Table 10–26 SCHEDULE_PUSH Procedure Parameters

Parameter	Description
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set the parameter unless so directed by Oracle Worldwide Support.
stop_on_error	The default, FALSE, indicates that the executor should continue even if errors, such as conflicts, are encountered. If TRUE, then shutdown (cleanly if possible) at the first indication that a transaction encountered an error at the destination site.
write_trace	When set to TRUE, Oracle records the result value returned by the function in the server's trace file.
startup_seconds	Maximum number of seconds to wait for a previous push to the same destination.
execution_seconds	If >0, then stop execution cleanly after the specified number of seconds of real time. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue.
delay_seconds	Do not return before the specified number of seconds have elapsed, even if the queue is empty. Useful for reducing execution overhead if PUSH is called from a tight loop.
transaction_count	If > 0, then the maximum number of transactions to be pushed before stopping. If transaction_count and execution_seconds are zero (the default), then transactions are executed until there are no more in the queue that need to be pushed.

SET_DISABLED procedure

To disable or enable propagation of the deferred transaction queue from the current site to a given destination site. If the disabled parameter is TRUE, then the procedure disables propagation to the given destination and future invocations of PUSH do not push the deferred remote procedure call (RPC) queue. SET_DISABLED eventually affects a session already pushing the queue to the given destination, but does not affect sessions appending to the queue with DBMS_DEFER.

If the disabled parameter is FALSE, then the procedure enables propagation to the given destination and, although this does not push the queue, it permits future invocations to PUSH to push the queue to the given destination. Whether the disabled parameter is TRUE or FALSE, a COMMIT is required for the setting to take effect in other sessions.

Syntax

```
DBMS_DEFER_SYS.SET_DISABLED (
   destination   IN   VARCHAR2,
   disabled      IN   BOOLEAN := TRUE);
```

Parameters

Table 10–27 SET_DISABLED Procedure Parameters

Parameter	Description
destination	The fully qualified database name of the node whose propagation status you want to change.
disabled	By default, this parameter disables propagation of the deferred transaction queue from your current site to the given destination. Set this to FALSE to enable propagation.

Exceptions

Table 10–28 SET_DISABLED Procedure Exceptions

Exception	Description
NO_DATA_FOUND	No entry was found in the DEFSCHEDULE view for the given destination.

UNREGISTER_PROPAGATOR procedure

To unregister a user as the propagator from the local database. This procedure

- Deletes the given propagator from DEFPROPAGATOR.
- Revokes privileges granted by REGISTER_PROPAGATOR from the given user (including identical privileges granted independently).
- Drops any generated wrappers in the schema of the given propagator, and marks them as dropped in the replication catalog.

Syntax

```
DBMS_DEFER_SYS.UNREGISTER_PROPAGATOR (
   username  IN  VARCHAR2,
   timeout   IN  INTEGER DEFAULT DBMS_LOCK.MAXWAIT);
```

Parameters

Table 10–29 UNREGISTER_PROPAGATOR Procedure Parameters

Parameter	Description
username	Name of the propagator user.
timeout	Timeout in seconds. If the propagator is in use, then the procedure waits until timeout. The default is DBMS_LOCK.MAXWAIT.

Exceptions

Table 10–30 UNREGISTER_PROPAGATOR Procedure Exceptions

Parameter	Description
missingpropagator	Given user is not a propagator.
propagator_inuse	Propagator is in use, and thus cannot be unregistered. Try later.

UNSCHEDULE_PURGE procedure

This procedure stops automatic purges of pushed transactions from the deferred transaction queue at a snapshot or master site.

Syntax

```
DBMS_DEFER_SYS.UNSCHEDULE_PURGE;
```

Parameters

None

UNSCHEDULE_PUSH procedure

This procedure stops automatic pushes of the deferred transaction queue from a snapshot or master site to another master site.

Syntax

```
DBMS_DEFER_SYS.UNSCHEDULE_PUSH (
   dblink   IN   VARCHAR2);
```

Parameters

Table 10–31 UNSCHEDULE_PUSH Procedure Parameters

Parameter	Description
dblink	Fully qualified pathname to master database site at which you want to unschedule periodic execution of deferred remote procedure calls.

Table 10–32 UNSCHEDULE_PUSH Procedure Exceptions

Exception	Description
NO_DATA_FOUND	No entry was found in the DEFSCHEDULE view for the given dblink.

11

DBMS_DESCRIBE

You can use the DBMS_DESCRIBE package to get information about a PL/SQL object. When you specify an object name, DBMS_DESCRIBE returns a set of indexed tables with the results. Full name translation is performed and security checking is also checked on the final object.

This package provides the same functionality as the Oracle Call Interface OCIDescribeAny call.

See Also: *Oracle Call Interface Programmer's Guide*

Security

This package is available to PUBLIC and performs its own security checking based on the schema object being described.

Types

The DBMS_DESCRIBE package declares two PL/SQL table types, which are used to hold data returned by DESCRIBE_PROCEDURE in its OUT parameters. The types are:

```
TYPE VARCHAR2_TABLE IS TABLE OF VARCHAR2(30)
    INDEX BY BINARY_INTEGER;

TYPE NUMBER_TABLE IS TABLE OF NUMBER
    INDEX BY BINARY_INTEGER;
```

Errors

DBMS_DESCRIBE can raise application errors in the range -20000 to -20004.

Table 11–1 DBMS_DESCRIBE Errors

Error	Description
ORA-20000	ORU 10035: cannot describe a package ('X') only a procedure within a package.
ORA-20001	ORU-10032: procedure 'X' within package 'Y' does not exist.
ORA-20002	ORU-10033: object 'X' is remote, cannot describe; expanded name 'Y'.
ORA-20003	ORU-10036: object 'X' is invalid and cannot be described.
ORA-20004	Syntax error attempting to parse 'X'.

Summary of Subprograms

DBMS_DESCRIBE contains only one procedure: DESCRIBE_PROCEDURE.

DESCRIBE_PROCEDURE procedure

The procedure DESCRIBE_PROCEDURE accepts the name of a stored procedure, a description of the procedure, and each of its parameters.

Syntax

```
DBMS_DESCRIBE.DESCRIBE_PROCEDURE(
   object_name    IN  VARCHAR2,
   reserved1      IN  VARCHAR2,
   reserved2      IN  VARCHAR2,
   overload       OUT NUMBER_TABLE,
   position       OUT NUMBER_TABLE,
   level          OUT NUMBER_TABLE,
   argument_name  OUT VARCHAR2_TABLE,
   datatype       OUT NUMBER_TABLE,
   default_value  OUT NUMBER_TABLE,
   in_out         OUT NUMBER_TABLE,
   length         OUT NUMBER_TABLE,
   precision      OUT NUMBER_TABLE,
   scale          OUT NUMBER_TABLE,
   radix          OUT NUMBER_TABLE,
   spare          OUT NUMBER_TABLE);
```

Parameters

Table 11–2 DBMS_DESCRIBE.DESCRIBE_PROCEDURE Parameters

Parameter	Description
object_name	Name of the procedure being described. The syntax for this parameter follows the rules used for identifiers in SQL. The name can be a synonym. This parameter is required and may not be null. The total length of the name cannot exceed 197 bytes. An incorrectly specified OBJECT_NAME can result in one of the following exceptions: ORA-20000 - A package was specified. You can only specify a stored procedure, stored function, packaged procedure, or packaged function. ORA-20001 - The procedure or function that you specified does not exist within the given package. ORA-20002 - The object that you specified is a remote object. This procedure cannot currently describe remote objects. ORA-20003 - The object that you specified is invalid and cannot be described. ORA-20004 - The object was specified with a syntax error.
reserved1 reserved2	Reserved for future use -- must be set to NULL or the empty string.
overload	A unique number assigned to the procedure's signature. If a procedure is overloaded, then this field holds a different value for each version of the procedure.
position	Position of the argument in the parameter list. Position 0 returns the values for the return type of a function.
level	If the argument is a composite type, such as record, then this parameter returns the level of the datatype. See the *Oracle Call Interface Programmer's Guide* for a description of the ODESSP call for an example.
argument_name	Name of the argument associated with the procedure that you are describing.

Table 11–2 DBMS_DESCRIBE.DESCRIBE_PROCEDURE Parameters

Parameter	Description
datatype	Oracle datatype of the argument being described. The datatypes and their numeric type codes are: 0 placeholder for procedures with no arguments 1 VARCHAR, VARCHAR, STRING 2 NUMBER, INTEGER, SMALLINT, REAL, FLOAT, DECIMAL 3 BINARY_INTEGER, PLS_INTEGER, POSITIVE, NATURAL 8 LONG 11 ROWID 12 DATE 23 RAW 24 LONG RAW 96 CHAR (ANSI FIXED CHAR), CHARACTER 106 MLSLABEL 250 PL/SQL RECORD 251 PL/SQL TABLE 252 PL/SQL BOOLEAN
default_value	1 if the argument being described has a default value; otherwise, the value is 0.
in_out	Describes the mode of the parameter: 0 IN 1 OUT 2 IN OUT
length	Data length, in bytes, of the argument being described.
precision	If the argument being described is of datatype 2 (NUMBER), then this parameter is the precision of that number.
scale	If the argument being described is of datatype 2 (NUMBER, etc.), then this parameter is the scale of that number.
radix	If the argument being described is of datatype 2 (NUMBER, etc.), then this parameter is the radix of that number.
spare	Reserved for future functionality.

Return Values

All values from DESCRIBE_PROCEDURE are returned in its OUT parameters. The datatypes for these are PL/SQL tables, in order to accommodate a variable number of parameters.

Examples

One use of the DESCRIBE_PROCEDURE procedure would be as an external service interface.

For example, consider a client that provides an OBJECT_NAME of SCOTT.ACCOUNT_UPDATE where ACCOUNT_UPDATE is an overloaded function with specification:

```
table account (account_no number, person_id number,
               balance number(7,2))
table person  (person_id number(4), person_nm varchar2(10))

function ACCOUNT_UPDATE (account_no   number,
                         person       person%rowtype,
                         amounts      dbms_describe.number_table,
                         trans_date   date)
                         return       accounts.balance%type;

function ACCOUNT_UPDATE (account_no   number,
                         person       person%rowtype,
                         amounts      dbms_describe.number_table,
                         trans_no     number)
                         return       accounts.balance%type;
```

The describe of this procedure might look similar to the output shown below.

```
overload position  argument level  datatype length prec scale rad
-------- --------- -------- ------ -------- ------ ---- ----- ---
       1         0                0        2     22    7     2  10
       1         1   ACCOUNT      0        2      0    0     0   0
       1         2   PERSON       0      250      0    0     0   0
       1         1   PERSON_ID    1        2     22    4     0  10
       1         2   PERSON_NM    1        1     10    0     0   0
       1         3   AMOUNTS      0      251      0    0     0   0
       1         1                1        2     22    0     0   0
       1         4   TRANS_DATE   0       12      0    0     0   0
       2         0                0        2     22    7     2  10
       2         1   ACCOUNT_NO   0        2     22    0     0   0
       2         2   PERSON       0        2     22    4     0  10
       2         3   AMOUNTS      0      251     22    4     0  10
       2         1                1        2      0    0     0   0
       2         4   TRANS_NO     0        2      0    0     0   0
```

The following PL/SQL procedure has as its parameters all of the PL/SQL datatypes:

```
CREATE OR REPLACE PROCEDURE p1 (
        pvc2    IN     VARCHAR2,
        pvc     OUT    VARCHAR,
        pstr    IN OUT STRING,
        plong   IN     LONG,
        prowid  IN     ROWID,
        pchara  IN     CHARACTER,
        pchar   IN     CHAR,
        praw    IN     RAW,
        plraw   IN     LONG RAW,
        pbinint IN     BINARY_INTEGER,
        pplsint IN     PLS_INTEGER,
        pbool   IN     BOOLEAN,
        pnat    IN     NATURAL,
        ppos    IN     POSITIVE,
        pposn   IN     POSITIVEN,
        pnatn   IN     NATURALN,
        pnum    IN     NUMBER,
        pintgr  IN     INTEGER,
        pint    IN     INT,
        psmall  IN     SMALLINT,
        pdec    IN     DECIMAL,
        preal   IN     REAL,
        pfloat  IN     FLOAT,
        pnumer  IN     NUMERIC,
        pdp     IN     DOUBLE PRECISION,
        pdate   IN     DATE,
        pmls    IN     MLSLABEL) AS

BEGIN
    NULL;
END;
```

If you describe this procedure using the package below:

```
CREATE OR REPLACE PACKAGE describe_it AS

    PROCEDURE desc_proc (name VARCHAR2);

END describe_it;

CREATE OR REPLACE PACKAGE BODY describe_it AS

  PROCEDURE prt_value(val VARCHAR2, isize INTEGER) IS
    n INTEGER;
```

```
BEGIN
  n := isize - LENGTHB(val);
  IF n < 0 THEN
    n := 0;
  END IF;
  DBMS_OUTPUT.PUT(val);
  FOR i in 1..n LOOP
    DBMS_OUTPUT.PUT(' ');
  END LOOP;
END prt_value;

PROCEDURE desc_proc (name VARCHAR2) IS

    overload     DBMS_DESCRIBE.NUMBER_TABLE;
    position     DBMS_DESCRIBE.NUMBER_TABLE;
    c_level      DBMS_DESCRIBE.NUMBER_TABLE;
    arg_name     DBMS_DESCRIBE.VARCHAR2_TABLE;
    dty          DBMS_DESCRIBE.NUMBER_TABLE;
    def_val      DBMS_DESCRIBE.NUMBER_TABLE;
    p_mode       DBMS_DESCRIBE.NUMBER_TABLE;
    length       DBMS_DESCRIBE.NUMBER_TABLE;
    precision    DBMS_DESCRIBE.NUMBER_TABLE;
    scale        DBMS_DESCRIBE.NUMBER_TABLE;
    radix        DBMS_DESCRIBE.NUMBER_TABLE;
    spare        DBMS_DESCRIBE.NUMBER_TABLE;
    idx          INTEGER := 0;

BEGIN
    DBMS_DESCRIBE.DESCRIBE_PROCEDURE(
            name,
            null,
            null,
            overload,
            position,
            c_level,
            arg_name,
            dty,
            def_val,
            p_mode,
            length,
            precision,
            scale,
            radix,
            spare);
```

```
      DBMS_OUTPUT.PUT_LINE('Position    Name        DTY  Mode');
      LOOP
          idx := idx + 1;
          prt_value(TO_CHAR(position(idx)), 12);
          prt_value(arg_name(idx), 12);
          prt_value(TO_CHAR(dty(idx)), 5);
          prt_value(TO_CHAR(p_mode(idx)), 5);
          DBMS_OUTPUT.NEW_LINE;
      END LOOP;
  EXCEPTION
     WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.NEW_LINE;
        DBMS_OUTPUT.NEW_LINE;

  END desc_proc;
END describe_it;
```

Then, the results, as shown below, list all the numeric codes for the PL/SQL datatypes:

```
Position  Name     Datatype_Code  Mode
1         PVC2       1              0
2         PVC        1              1
3         PSTR       1              2
4         PLONG      8              0
5         PROWID     11             0
6         PCHARA     96             0
7         PCHAR      96             0
8         PRAW       23             0
9         PLRAW      24             0
10        PBININT    3              0
11        PPLSINT    3              0
12        PBOOL      252            0
13        PNAT       3              0
14        PPOS       3              0
15        PPOSN      3              0
16        PNATN      3              0
17        PNUM       2              0
18        PINTGR     2              0
19        PINT       2              0
20        PSMALL     2              0
21        PDEC       2              0
22        PREAL      2              0
23        PFLOAT     2              0
24        PNUMER     2              0
```

```
25          PDP         2               0
26          PDATE       12              0
27          PMLS        106             0
```

Usage Notes

There is currently no way from a third generation language to directly bind to an argument of type `record` or `boolean`. For Booleans, there are the following work-arounds:

- Assume function `F` returns a Boolean. `G` is a procedure with one `IN` Boolean argument, and `H` is a procedure which has one `OUT` Boolean argument. Then, you can execute these functions, binding in `DTYINTs` (native integer) as follows, where `0=>FALSE` and `1=>TRUE`:

```
begin :dtyint_bind_var := to_number(f); end;

begin g(to_boolean(:dtyint_bind_var)); end;

declare b boolean; begin h(b); if b then :dtyint_bind_var := 1;
else :dtyint_bind_var := 0; end if; end;
```

- Access to procedures with arguments of type `record` require writting a wrapper similar to that in the last example above (see funciton `H`).

12

DBMS_DISTRIBUTED_TRUST_ADMIN

DBMS_DISTRIBUTED_TRUST_ADMIN procedures maintain the Trusted Database List. It is used to define the databases that are (or are *not*) to be trusted.

> **Note:** This list is used in conjunction with the list at the Central Authority (CA) to determine if a privileged database link from a particular server can be accepted. A particular server can be listed locally in the Trusted Database List, regardless of its listing at the CA.

Requirements

To execute DBMS_DISTRIBUTED_TRUST_ADMIN, the EXECUTE_CATALOG_ROLE role must be granted to the DBA. To select from the view TRUSTED_SERVERS, the SELECT_CATALOG_ROLE role must be granted to the DBA.

It is important to know whether all servers are trusted or not trusted. Trusting a particular server with the ALLOW_SERVER procedure does not have any impact if the database already trusts all databases, or if that database is already trusted. Similarly, denying a particular server with the DENY_SERVER procedure does not have any effect if the database already doesn't trust any database or if that database is already untrusted.

The procedures DENY_ALL and ALLOW_ALL delete all entries (i.e. server names) that are explicitly allowed or denied using the ALLOW_SERVER procedure or DENY_SERVER procedure respectively.

Summary of Subprograms

Table 12–1 DBMS_DISTRIBUTED_TRUST_ADMIN Package Subprograms

Subprogram	Description
ALLOW_ALL procedure on page 12-2	Empties the list, and inserts a row indicating that all servers should be untrusted.
ALLOW_SERVER procedure on page 12-3	Enables a specific server to be allowed access, even though deny all is indicated in the list.
DENY_ALL procedure on page 12-4	Empties the list, and inserts a row indicating that all servers should be trusted.
DENY_SERVER procedure on page 12-4	Enables a specific server to be denied access, even though allow is indicated in the list.

ALLOW_ALL procedure

This procedure empties the Trusted Database List, and specifies that all servers trusted by the central authority, such as Oracle Security Server, are allowed access.

The view TRUSTED_SERVERS will show "TRUSTED ALL" indicating that all servers are currently trusted by the central authority, such as Oracle Security Server.

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_ALL;
```

Parameters

None.

Exceptions

None.

Usage Notes

ALLOW_ALL only applies to the servers listed as trusted at the Central Authority.

ALLOW_SERVER procedure

This procedure ensures that the specified server is considered trusted (even if you have previously specified "deny all").

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER (
   server IN VARCHAR2);
```

Parameters

Table 12–2 ALLOW_SERVER Procedure Parameters

Parameter	Description
server	Unique, fully-qualified name of the server to be trusted.

Exceptions

None.

Usage Notes

If the Trusted Servers List contains the entry "deny all", then this procedure adds a specification indicating that a specific database (for example, DBx) is to be trusted.

If the Trusted Servers List contains the entry "allow all", and if there is no "deny DBx" entry in the list, then executing this procedure causes no change.

If the Trusted Servers List contains the entry "allow all", and if there is a "deny DBx" entry in the list, then that entry is deleted.

DENY_ALL procedure

This procedure enables a specific server to be allowed access, even though deny all is indicated in the list.

The view TRUSTED_SERVERS will show "UNTRUSTED ALL" indicating that no servers are currently trusted.

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL;
```

Parameters

None.

Exceptions

None.

DENY_SERVER procedure

This procedure ensures that the specified server is considered untrusted (even if you have previously specified "`allow all`").

Syntax

```
DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER (
   server IN VARCHAR2);
```

Parameters

Table 12–3 DENY_SERVER Procedure Parameters

Parameter	Description
server	Unique, fully-qualified name of the server to be untrusted.

Exceptions

None.

Usage Notes

If the Trusted Servers List contains the entry "allow all", then this procedure adds an entry indicating that the specified database (for example, DBx) is not to be trusted.

If the Trusted Servers List contains the entry "deny all", and if there is no "allow DBx" entry in the list, then this procedure causes no change.

If the Trusted Servers List contains the entry "deny all", and if there is an "allow DBx" entry, then this procedure causes that entry to be deleted.

Example

If you have not yet used the package DBMS_DISTRIBUTED_TRUST_ADMIN to change trust, the default is that all servers defined in the Oracle Security Server are considered trusted:

```
SELECT * FROM TRUSTED_SERVERS;
TRUST     NAME
--------- --------------------
Trusted   All

1 row selected.
```

Because all servers are currently trusted, you can execute the DENY_SERVER procedure and specify that a particular server is not trusted:

```
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_SERVER
                          ('SALES.US.AMERICAS.ACME_AUTO.COM');

Statement processed.

SELECT * FROM TRUSTED_SERVERS;

TRUST      NAME
--------- ----------------------------------------------
Untrusted SALES.US.AMERICAS.ACME_AUTO.COM

1 row selected
```

By executing the DENY_ALL procedure, you can choose to not trust any database server:

```
EXECUTE DBMS_DISTRIBUTED_TRUST_ADMIN.DENY_ALL;
```

```
Statement processed.

SELECT * FROM TRUSTED_SERVERS;

TRUST      NAME
---------- ------------------------------------------------
Untrusted All

1 row selected.
```

The ALLOW_SERVER procedure can be used to specify that one particular database is to be trusted:

```
EXECUTE
DBMS_DISTRIBUTED_TRUST_ADMIN.ALLOW_SERVER
                            ('SALES.US.AMERICAS.ACME_AUTO.COM');

Statement processed.

SELECT * FROM TRUSTED_SERVERS;

TRUST      NAME
---------- ------------------------------------------------
Trusted    SALES.US.AMERICAS.ACME_AUTO.COM

1 row selected.
```

13

DBMS_HS

DBMS_HS contains subprograms to set and unset Heterogeneous Services (HS) initialization parameters, capabilities, instance names, and class names.

This packages are installed by SYS (connect internal). The HS tables are created by CATHS.SQL and are owned by the system.

See Also: *Oracle8i Distributed Database Systems*

Exceptions

```
miss_base_caps exception;
pragma exception_init(miss_base_caps, -24274);
miss_base_caps_num number := -24274;  dupl_base_caps exception;
pragma exception_init(dupl_base_caps, -24270);
dupl_base_caps_num number := -24270;
dupl_base_caps_msg varchar2(76) := 'HS$_BASE_CAPS';

miss_base_dd exception;
pragma exception_init(miss_base_dd, -24274);
miss_base_dd_num number := -24274;
miss_base_dd_msg varchar2(76) := 'HS$_BASE_DD';

dupl_base_dd exception;
pragma exception_init(dupl_base_dd, -24270);
dupl_base_dd_num number := -24270;
dupl_base_dd_msg varchar2(76) := 'HS$_BASE_DD';

miss_external_object exception;
pragma exception_init(miss_external_object, -24274);
miss_external_object_num number := -24274;
miss_external_object_msg varchar2(76) := 'HS$_EXTERNAL_OBJECTS';

dupl_external_object exception;
pragma exception_init(dupl_external_object, -24270);
dupl_external_object_num number := -24270;
dupl_external_object_msg varchar2(76) := 'HS$_EXTERNAL_OBJECTS';

miss_granted_user exception;
pragma exception_init(miss_granted_user, -24274);
miss_granted_user_num number := -24274;
miss_granted_user_msg varchar2(76) := 'HS$_GRANTED_USERS';

dupl_granted_user exception;
pragma exception_init(dupl_granted_user, -24270);
dupl_granted_user_num number := -24270;
dupl_granted_user_msg varchar2(76) := 'HS$_GRANTED_USERS';

miss_access_grantee exception;
pragma exception_init(miss_access_grantee, -24274);
miss_access_grantee_num number := -24274;
miss_access_grantee_msg varchar2(76) := 'HS$_ACCESS_GRANTEES';

dupl_access_grantee exception;
```

```
pragma exception_init(dupl_access_grantee, -24270);
dupl_access_grantee_num number := -24270;
dupl_access_grantee_msg varchar2(76) := 'HS$_ACCESS_GRANTEES';

miss_create_grantee exception;
pragma exception_init(miss_create_grantee, -24274);
miss_create_grantee_num number := -24274;
miss_create_grantee_msg varchar2(76) := 'HS$_CREATE_GRANTEES';

dupl_create_grantee exception;
pragma exception_init(dupl_create_grantee, -24270);
dupl_create_grantee_num number := -24270;
dupl_create_grantee_msg varchar2(76) := 'HS$_CREATE_GRANTEES';

miss_privilege exception;
pragma exception_init(miss_privilege, -24274);
miss_privilege_num number := -24274;
miss_privilege_msg varchar2(76) := 'HS$_PRIVILEGES';

dupl_privilege exception;
pragma exception_init(dupl_privilege, -24270);
dupl_privilege_num number := -24270;
dupl_privilege_msg varchar2(76) := 'HS$_PRIVILEGES';

miss_class_caps exception;
pragma exception_init(miss_class_caps, -24274);
miss_class_caps_num number := -24274;
miss_class_caps_msg varchar2(76) := 'HS$_CLASS_CAPS';


dupl_class_caps exception;
pragma exception_init(dupl_class_caps, -24270);
dupl_class_caps_num number := -24270;
dupl_class_caps_msg varchar2(76) := 'HS$_CLASS_CAPS';

miss_class_dd exception;
pragma exception_init(miss_class_dd, -24274);
miss_class_dd_num number := -24274;
miss_class_dd_msg varchar2(76) := 'HS$_CLASS_DD';

dupl_class_dd exception;
pragma exception_init(dupl_class_dd, -24270);
dupl_class_dd_num number := -24270;
dupl_class_dd_msg varchar2(76) := 'HS$_CLASS_DD';
```

```
bad_TRANSLATION_TYPE exception;
pragma exception_init(bad_TRANSLATION_TYPE, -24271);
bad_TRANSLATION_TYPE_num number := -24271;
bad_TRANSLATION_TYPE_msg varchar2(76) := 'NULL';

bad_TRANSLATION_TEXT exception;
pragma exception_init(bad_TRANSLATION_TEXT, -24273);
bad_TRANSLATION_TEXT_num number := -24273;
bad_TRANSLATION_TEXT_msg varchar2(76) := 'NULL';

miss_class_init exception;
pragma exception_init(miss_class_init, -24274);
miss_class_init_num number := -24274;
miss_class_init_msg varchar2(76) := 'HS$_CLASS_INIT';

dupl_class_init exception;
pragma exception_init(dupl_class_init, -24270);
dupl_class_init_num number := -24270;
dupl_class_init_msg varchar2(76) := 'HS$_CLASS_INIT';

bad_INIT_VALUE_TYPE exception;
pragma exception_init(bad_INIT_VALUE_TYPE, -24272);
bad_INIT_VALUE_TYPE_num number := -24272;
bad_INIT_VALUE_TYPE_msg varchar2(76) := 'NULL';

miss_fds_class exception;
pragma exception_init(miss_fds_class, -24274);
miss_fds_class_num number := -24274;
miss_fds_class_msg varchar2(76) := 'HS$_FDS_CLASS';

dupl_fds_class exception;
pragma exception_init(dupl_fds_class, -24270);
dupl_fds_class_num number := -24270;
dupl_fds_class_msg varchar2(76) := 'HS$_FDS_CLASS';

miss_fds_inst exception;
pragma exception_init(miss_fds_inst, -24274);
miss_fds_inst_num number := -24274;
miss_fds_inst_msg varchar2(76) := 'HS$_FDS_INST';

dupl_fds_inst exception;
pragma exception_init(dupl_fds_inst, -24270);
dupl_fds_inst_num number := -24270;
dupl_fds_inst_msg varchar2(76) := 'HS$_FDS_INST';
```

```
miss_inst_caps exception;
pragma exception_init(miss_inst_caps, -24274);
miss_inst_caps_num number := -24274;
miss_inst_caps_msg varchar2(76) := 'HS$_INST_CAPS';

dupl_inst_caps exception;
pragma exception_init(dupl_inst_caps, -24270);
dupl_inst_caps_num number := -24270;
dupl_inst_caps_msg varchar2(76) := 'HS$_INST_CAPS';

miss_inst_dd exception;
pragma exception_init(miss_inst_dd, -24274);
miss_inst_dd_num number := -24274;
miss_inst_dd_msg varchar2(76) := 'HS$_INST_DD';

dupl_inst_dd exception;
pragma exception_init(dupl_inst_dd, -24270);
dupl_inst_dd_num number := -24270;
dupl_inst_dd_msg varchar2(76) := 'HS$_INST_DD';

miss_inst_init exception;
pragma exception_init(miss_inst_init, -24274);
miss_inst_init_num number := -24274;
miss_inst_init_msg varchar2(76) := 'HS$_INST_INIT';

dupl_inst_init exception;
pragma exception_init(dupl_inst_init, -24270);
dupl_inst_init_num number := -24270;
dupl_inst_init_msg varchar2(76) := 'HS$_INST_INIT';

no_privilege exception;
pragma exception_init(no_privilege, -24277);
no_privilege_num number := -24277;
no_privilege_msg varchar2(76) := null;

privilege_mismatch exception;
pragma exception_init(privilege_mismatch, -24278);
privilege_mismatch_num number := -24278;
privilege_mismatch_msg varchar2(76) := null;

lib_priv_mismatch exception;
pragma exception_init(lib_priv_mismatch, -24279);
lib_priv_mismatch_num number := -24279;
lib_priv_mismatch_msg varchar2(76) := null;
```

Summary of Subprograms

Table 13–1 DBMS_HS Package Subprograms

Subprogram	Description
ALTER_BASE_CAPS procedure on page 13-8	Alters a row in the HS$_BASE_CAPS table.
ALTER_BASE_DD procedure on page 13-9	Alters a row in the HS$_BASE_DD table.
ALTER_CLASS_CAPS procedure on page 13-9	Alters the contents of the HS$_CLASS_CAPS table.
ALTER_CLASS_DD procedure on page 13-9	Modifies the contents of the HS$_CLASS_DD table.
ALTER_CLASS_INIT procedure on page 13-9	Alters the contents of the HS$_CLASS_INIT table.
ALTER_FDS_CLASS procedure on page 13-10	Alters the contents of the HS$_FDS_CLASS table.
ALTER_FDS_INST procedure on page 13-10	Modifies the contents of the HS$_FDS_INST table.
ALTER_INST_CAPS procedure on page 13-10	Modifies the contents of the $HS_INST_CAPS table.
ALTER_INST_DD procedure on page 13-11	Alters the contents of the HS$_INST_DD table.
ALTER_INST_INIT procedure on page 13-11	Alters the contents of the HS$_INST_INIT table.
COPY_CLASS procedure on page 13-12	Copies everything for a class to another class.
COPY_INST procedure on page 13-12	Copies everything for an HS$_FDS_INST to a new instance in the same FDS_CLASS.
CREATE_BASE_CAPS procedure on page 13-12	Creates a row in the HS$_BASE_CAPS table.
CREATE_BASE_CAPS procedure on page 13-12	Creates a row in the HS$_BASE_DD table.
CREATE_BASE_DD procedure on page 13-12	Creates a row in the HS$_BASE_DD table.

Table 13–1 DBMS_HS Package Subprograms

Subprogram	Description
CREATE_CLASS_CAPS procedure on page 13-12	Creates a row in the HS$_CLASS_CAPS table.
CREATE_CLASS_DD procedure on page 13-13	Creates a row in the HS$_CLASS_DD table.
CREATE_CLASS_INIT procedure on page 13-13	Creates a row in the HS$_CLASS_INIT table.
CREATE_FDS_CLASS procedure on page 13-14	Creates a row in the HS$_FDS_CLASS table.
CREATE_FDS_INST procedure on page 13-14	Creates a row in the HS$_FDS_INST table.
CREATE_INST_CAPS procedure on page 13-14	Creates a row in the HS$_INST_CAPS table.
CREATE_INST_DD procedure on page 13-14	Creates a row in the HS$_INST_DD table.
CREATE_INST_INIT procedure on page 13-15	Creates a row in the HS$_INST_INIT table.
DROP_BASE_CAPS procedure on page 13-15	Drops a row from the HS$_BASE_CAPS table as specified by the CAP_NUMBER parameter.
DROP_BASE_DD procedure on page 13-16	Drops a row from the HS$_BASE_DD table as specified by table_name.
DROP_CLASS_CAPS procedure on page 13-16	Deletes a row from the HS$_CLASS_CAPS table as specified by the FDS_CLASS_NAME and CAP_NUMBER.
DROP_CLASS_DD procedure on page 13-16	Deletes row in HS$_CLASS_DD specified by FDS_CLASS_NAME and DD_TABLE_NAME.
DROP_CLASS_INIT procedure on page 13-16	Drops row in HS$_CLASS_INIT as specified by FDS_CLASS_NAME and INIT_VALUE_NAME.
DROP_FDS_CLASS procedure on page 13-17	Drops row in HS$_FDS_CLASS as specified by FDS_CLASS_NAME.
DROP_FDS_INST procedure on page 13-17	Drops row in HS$_FDS_INST table as specified by FDS_INST_NAME and FDS_CLASS_NAME.
DROP_INST_CAPS procedure on page 13-17	Deletes rows in HS$_INST_CAPS specified by FDS_INST_NAME, FDS_CLASS_NAME and CAP_NUMBER.

Table 13–1 DBMS_HS Package Subprograms

Subprogram	Description
DROP_INST_DD procedure on page 13-17	Drops rows from HS$_INST_DD specified by FDS_INST_NAME, FDS_CLASS_NAME and DD_TABLE_NAME.
DROP_INST_INIT procedure on page 13-17	Drops rows from HS$_INST_INIT table as specified by FDS_INST_NAME, FDS_CLASS_NAME, and INIT_VALUE_NAME.
REPLACE_BASE_CAPS procedure on page 13-18	Creates or replaces a row in the HS$_BASE_CAPS table.
REPLACE_BASE_DD procedure on page 13-18	Creates or replaces a row in the HS$_BASE_DD table.
REPLACE_CLASS_CAPS procedure on page 13-18	Performs a create or replace on the HS$_CLASS_CAPS table.
REPLACE_CLASS_DD procedure on page 13-19	Performs a create or replace on the HS$_CLASS_DD table.
REPLACE_CLASS_INIT procedure on page 13-19	Creates or updates a row in the HS$_CLASS_INIT table.
REPLACE_FDS_CLASS procedure on page 13-20	Performs create or replace operations on the HS$_FDS_CLASS table.
REPLACE_FDS_INST procedure on page 13-20	Creates or replaces rows in the HS$_FDS_INST table.
REPLACE_INST_CAPS procedure on page 13-20	Performs a create or replace on the HS$_INST_CAPS table.
REPLACE_INST_DD procedure on page 13-21	Performs a create or replace operation on the HS$_INST_DD table.
REPLACE_INST_INIT procedure on page 13-21	Performs a create or replace on the HS$_INST_INIT table.

ALTER_BASE_CAPS procedure

This procedure alters a row in the HS$_BASE_CAPS table.

Syntax

```
DBMS_HS.ALTER_BASE_CAPS (
   CAP_NUMBER          IN NUMBER,
   new_CAP_NUMBER      IN NUMBER   := -1e-130,
   new_CAP_DESCRIPTION IN VARCHAR2 := '-');
```

ALTER_BASE_DD procedure

This procedure alters a row in the HS$_BASE_DD table.

Syntax

```
DBMS_HS.ALTER_BASE_DD (
   DD_TABLE_NAME     IN VARCHAR2,
   new_DD_TABLE_NAME IN VARCHAR2 := '-',
   new_DD_TABLE_DESC IN VARCHAR2 := '-');
```

ALTER_CLASS_CAPS procedure

This procedure alters the contents of the HS$_CLASS_CAPS table.

Syntax

```
DBMS_HS.ALTER_CLASS_CAPS(
   FDS_CLASS_NAME      IN VARCHAR2,
   CAP_NUMBER          IN NUMBER,
   new_FDS_CLASS_NAME  IN VARCHAR2 := '-',
   new_CAP_NUMBER      IN NUMBER   := -1e-130,
   new_CONTEXT         IN NUMBER   := -1e-130,
   new_TRANSLATION     IN VARCHAR2 := '-',
   new_ADDITIONAL_INFO IN NUMBER   := -1e-130);
```

ALTER_CLASS_DD procedure

This procedure modifies the contents of the HS$_CLASS_DD table.

Syntax

```
DBMS_HS.ALTER_CLASS_DD (
   FDS_CLASS_NAME       IN VARCHAR2,
   DD_TABLE_NAME        IN VARCHAR2,
   new_FDS_CLASS_NAME   IN VARCHAR2 := '-',
   new_DD_TABLE_NAME    IN VARCHAR2 := '-',
   new_TRANSLATION_TYPE IN CHAR     := '-',
   new_TRANSLATION_TEXT IN VARCHAR2 := '-');
```

ALTER_CLASS_INIT procedure

This procedure alters the contents of the HS$_CLASS_INIT table.

Syntax

```
DBMS_HS.ALTER_CLASS_INIT (
   FDS_CLASS_NAME      IN VARCHAR2,
   INIT_VALUE_NAME     IN VARCHAR2,
   new_FDS_CLASS_NAME  IN VARCHAR2 := '-',
   new_INIT_VALUE_NAME IN VARCHAR2 := '-',
   new_INIT_VALUE      IN VARCHAR2 := '-',
   new_INIT_VALUE_TYPE IN VARCHAR2 := '-');
```

ALTER_FDS_CLASS procedure

This procedure alters the contents of the HS$_FDS_CLASS table.

Syntax

```
DBMS_HS.ALTER_FDS_CLASS(
   FDS_CLASS_NAME         IN VARCHAR2,
   new_FDS_CLASS_NAME     IN VARCHAR2 := '-',
   new_FDS_CLASS_COMMENTS IN VARCHAR2 := '-');
```

ALTER_FDS_INST procedure

This procedure modifies the contents of the HS$_FDS_INST table.

Syntax

```
DBMS_HS.ALTER_FDS_INST (
   FDS_INST_NAME         IN VARCHAR2,
   FDS_CLASS_NAME        IN VARCHAR2,
   new_FDS_INST_NAME     IN VARCHAR2 := '-',
   new_FDS_CLASS_NAME    IN VARCHAR2 := '-',
   new_FDS_INST_COMMENTS IN VARCHAR2 := '-');
```

ALTER_INST_CAPS procedure

This procedure modifies the contents of the $HS_INST_CAPS table.

Syntax

```
DBMS_HS.ALTER_INST_CAPS (
   FDS_INST_NAME       IN VARCHAR2,
   FDS_CLASS_NAME      IN VARCHAR2,
   CAP_NUMBER          IN NUMBER,
   new_FDS_INST_NAME   IN VARCHAR2 := '-',
   new_FDS_CLASS_NAME  IN VARCHAR2 := '-',
   new_CAP_NUMBER      IN NUMBER   := -1e-130,
   new_CONTEXT         IN NUMBER   := -1e-130,
   new_TRANSLATION     IN VARCHAR2 := '-',
   new_ADDITIONAL_INFO IN NUMBER   := -1e-130);
```

ALTER_INST_DD procedure

This procedure alters the contents of the HS$_INST_DD table.

Syntax

```
DBMS_HS.ALTER_INST_DD (
   FDS_INST_NAME        IN VARCHAR2,
   FDS_CLASS_NAME       IN VARCHAR2,
   DD_TABLE_NAME        IN VARCHAR2,
   new_FDS_INST_NAME    IN VARCHAR2 := '-',
   new_FDS_CLASS_NAME   IN VARCHAR2 := '-',
   new_DD_TABLE_NAME    IN VARCHAR2 := '-',
   new_TRANSLATION_TYPE IN CHAR     := '-',
   new_TRANSLATION_TEXT IN VARCHAR2 := '-');
```

ALTER_INST_INIT procedure

This procedure alters the contents of the HS$_INST_INIT table.

Syntax

```
DBMS_HS.ALTER_INST_INIT (
   FDS_INST_NAME       IN VARCHAR2,
   FDS_CLASS_NAME      IN VARCHAR2,
   INIT_VALUE_NAME     IN VARCHAR2,
   new_FDS_INST_NAME   IN VARCHAR2 := '-',
   new_FDS_CLASS_NAME  IN VARCHAR2 := '-',
   new_INIT_VALUE_NAME IN VARCHAR2 := '-',
   new_INIT_VALUE      IN VARCHAR2 := '-',
   new_INIT_VALUE_TYPE IN VARCHAR2 := '-');
```

COPY_CLASS procedure

This procedure copies everything for a class to another class.

Syntax

```
DBMS_HS.COPY_CLASS (
   old_fds_class_name VARCHAR2,
```

COPY_INST procedure

This procedure copies everything for an HS$_FDS_INST to a new instance in the same FDS_CLASS.

Syntax

```
DBMS_HS.COPY_INST (
   FDS_INST_NAME     IN VARCHAR2,
   FDS_CLASS_NAME    IN VARCHAR2,
   new_FDS_INST_NAME IN VARCHAR2,
   new_FDS_COMMENTS  IN VARCHAR2 default '-');
```

CREATE_BASE_CAPS procedure

This procedure creates a row in the HS$_BASE_CAPS table.

Syntax

```
DBMS_HS.CREATE_BASE_CAPS (
   CAP_NUMBER      IN NUMBER,
   CAP_DESCRIPTION IN VARCHAR2 := NULL);
```

CREATE_BASE_DD procedure

This procedure creates a row in the HS$_BASE_DD table.

Syntax

```
DBMS_HS.CREATE_BASE_DD (
   DD_TABLE_NAME IN VARCHAR2,
   DD_TABLE_DESC IN VARCHAR2 := NULL);
```

CREATE_CLASS_CAPS procedure

This procedure creates a row in the HS$_CLASS_CAPS table.

The FDS_CLASS_NAME must be exist in the HS$_FDS_CLASS table. The CAP_NUMBER must be defined in the HS$_BASE_CAPS table.

Syntax

```
DBMS_HS.CREATE_CLASS_CAPS (
   FDS_CLASS_NAME  IN VARCHAR2,
   CAP_NUMBER      IN NUMBER,
   CONTEXT         IN NUMBER   := NULL,
   TRANSLATION     IN VARCHAR2 := NULL,
   ADDITIONAL_INFO IN NUMBER   := NULL);
```

CREATE_CLASS_DD procedure

This procedure creates a row in the HS$_CLASS_DD table.

The FDS_CLASS_NAME must exist in the HS$_FDS_CLASS table. The DD_TABLE_NAME must exist in the HS$_BASE_DD table. TRANSLATION_TYPE must be either 'T' (translated) or 'M' (mimicked). If TRANSLATION_TYPE = 'T', then the TRANSLATION_TEXT string must be supplied.

Syntax

```
DBMS_HS.CREATE_CLASS_DD (
   FDS_CLASS_NAME   IN VARCHAR2,
   DD_TABLE_NAME    IN VARCHAR2,
   TRANSLATION_TYPE IN CHAR,
   TRANSLATION_TEXT IN VARCHAR2 := NULL);
```

CREATE_CLASS_INIT procedure

This procedure creates a row in the HS$_CLASS_INIT table.

The FDS_CLASS_NAME must exist in the HS$_FDS_CLASS table. The INIT_VALUE_TYPE must either 'F' (environment variable) or 'M' (not an environment variable).

Syntax

```
DBMS_HS.CREATE_CLASS_INIT (
   FDS_CLASS_NAME  IN VARCHAR2,
   INIT_VALUE_NAME IN VARCHAR2,
   INIT_VALUE      IN VARCHAR2,
   INIT_VALUE_TYPE IN VARCHAR2);
```

CREATE_FDS_CLASS procedure

This procedure creates a row in the HS$_FDS_CLASS table.

Syntax

```
DBMS_HS.CREATE_FDS_CLASS (
   FDS_CLASS_NAME     IN VARCHAR2,
   FDS_CLASS_COMMENTS IN VARCHAR2 := NULL);
```

CREATE_FDS_INST procedure

This procedure creates a row in the HS$_FDS_INST table.

The FDS_CLASS_NAME must exist in the HS$_FDS_CLASS table.

Syntax

```
DBMS_HS.CREATE_FDS_INST (
   FDS_INST_NAME     IN VARCHAR2,
   FDS_CLASS_NAME    IN VARCHAR2,
   FDS_INST_COMMENTS IN VARCHAR2 := NULL);
```

CREATE_INST_CAPS procedure

This procedure creates a row in the HS$_INST_CAPS table.

The FDS_INST_NAME must exist in the HS$_FDS_INST table and be defined for the row in HS$_FDS_CLASS specified by the FDS_CLASS_NAME row. The CAP_NUMBER must be defined in the HS$_BASE_CAPS table.

Syntax

```
DBMS_HS.CREATE_INST_CAPS (
   FDS_INST_NAME   IN VARCHAR2,
   FDS_CLASS_NAME  IN VARCHAR2,
   CAP_NUMBER      IN NUMBER,
   CONTEXT         IN NUMBER   := NULL,
   TRANSLATION     IN VARCHAR2 := NULL,
   ADDITIONAL_INFO IN NUMBER   := NULL);
```

CREATE_INST_DD procedure

This procedure creates a row in the HS$_INST_DD table.

The FDS_INST_NAME must be defined in the HS$_FDS_INST table and must belong to the FDS_CLASS specified by the HS$_FDS_CLASS parameter. The DD_TABLE_NAME must be defined in the HS$_BASE_DD table. TRANSLATION_TYPE must be either 'T' (translated) or 'M' (mimicked). If TRANSLATION_TYPE is 'T', then TRANSLATION_TEXT must be supplied.

Syntax

```
DBMS_HS.CREATE_INST_DD (
   FDS_INST_NAME    IN VARCHAR2,
   FDS_CLASS_NAME   IN VARCHAR2,
   DD_TABLE_NAME    IN VARCHAR2,
   TRANSLATION_TYPE IN CHAR,
   TRANSLATION_TEXT IN VARCHAR2 := NULL);
```

CREATE_INST_INIT procedure

This procedure creates a row in the HS$_INST_INIT table.

The FDS_INST_NAME must exist in the HS$_FDS_INST table and must be exist in the HS$_FDS_CLASS table as specified by the FDS_CLASS_NAME parameter. The INIT_VALUE_TYPE must be defined either 'F' or 'T'.

Syntax

```
DBMS_HS.CREATE_INST_INIT(
   FDS_INST_NAME   IN VARCHAR2,
   FDS_CLASS_NAME  IN VARCHAR2,
   INIT_VALUE_NAME IN VARCHAR2,
   INIT_VALUE      IN VARCHAR2,
   INIT_VALUE_TYPE IN VARCHAR2);
```

DROP_BASE_CAPS procedure

This procedure drops a row from the HS$_BASE_CAPS table as specified by the CAP_NUMBER parameter.

Syntax

```
DBMS_HS.DROP_BASE_CAPS (
   CAP_NUMBER IN NUMBER);
```

DROP_BASE_DD procedure

This procedure drops a row from the HS$_BASE_DD table as specified by table_name.

Syntax

```
DBMS_HS.DROP_BASE_DD (
   DD_TABLE_NAME IN VARCHAR2);
```

DROP_CLASS_CAPS procedure

This procedure deletes a row from the HS$_CLASS_CAPS table as specified by the FDS_CLASS_NAME and CAP_NUMBER.

Syntax

```
DBMS_HS.DROP_CLASS_CAPS (
   FDS_CLASS_NAME IN VARCHAR2,
   CAP_NUMBER     IN NUMBER);
```

DROP_CLASS_DD procedure

This procedure deletes rows in HS$_CLASS_DD specified by FDS_CLASS_NAME and DD_TABLE_NAME.

Syntax

```
DBMS_HS.DROP_CLASS_DD(
   FDS_CLASS_NAME IN VARCHAR2,
   DD_TABLE_NAME  IN VARCHAR2);
```

DROP_CLASS_INIT procedure

Drops row in HS$_CLASS_INIT as specified by FDS_CLASS_NAME and INIT_VALUE_NAME.

Syntax

```
DBMS_HS.DROP_CLASS_INIT (
   FDS_CLASS_NAME  IN VARCHAR2,
   INIT_VALUE_NAME IN VARCHAR2);
```

DROP_FDS_CLASS procedure

This procedure drops rows in HS$_FDS_CLASS as specified by FDS_CLASS_NAME;

Syntax

```
DBMS_HS.DROP_FDS_CLASS (
    FDS_CLASS_NAME IN VARCHAR2);
```

DROP_FDS_INST procedure

This procedure drops rows in HS$_FDS_INST table as specified by FDS_INST_NAME and FDS_CLASS_NAME.

```
DBMS_HS.DROP_FDS_INST (
   FDS_INST_NAME  IN VARCHAR2,
   FDS_CLASS_NAME IN VARCHAR2);
```

DROP_INST_CAPS procedure

This procedure deletes rows in HS$_INST_CAPS specified by FDS_INST_NAME, FDS_CLASS_NAME and CAP_NUMBER.

Syntax

```
DBMS_HS.DROP_INST_CAPS (
   FDS_INST_NAME  IN VARCHAR2,
   FDS_CLASS_NAME IN VARCHAR2,
   CAP_NUMBER     IN NUMBER);
```

DROP_INST_DD procedure

This procedure drops rows from HS$_INST_DD specified by FDS_INST_NAME, FDS_CLASS_NAME and DD_TABLE_NAME.

```
DBMS_HS.DROP_INST_DD (
   FDS_INST_NAME  IN VARCHAR2,
   FDS_CLASS_NAME IN VARCHAR2,
   DD_TABLE_NAME  IN VARCHAR2);
```

DROP_INST_INIT procedure

Drops rows from HS$_INST_INIT table as specified by FDS_INST_NAME, FDS_CLASS_NAME, and INIT_VALUE_NAME.

Syntax

```
DBMS_HS.DROP_INST_INIT (
   FDS_INST_NAME   IN     VARCHAR2,
   FDS_CLASS_NAME  IN     VARCHAR2,
   INIT_VALUE_NAME IN     VARCHAR2);
   new_fds_class_name     VARCHAR2,
   new_fds_class_comments VARCHAR2 default '-');
```

REPLACE_BASE_CAPS procedure

This procedure creates or replaces a row in the HS$_BASE_CAPS table.

It first attempts to update the row in HS$_BASE_CAPS. If the row does not exist, then it attempts to insert the row. The new_CAP_NUMBER parameter is ignored if the row specified by CAP_NUMBER does not exist.

Syntax

```
DBMS_HS.REPLACE_BASE_CAPS (
   CAP_NUMBER          IN NUMBER,
   new_CAP_NUMBER      IN NUMBER   := NULL,
   new_CAP_DESCRIPTION IN VARCHAR2 := NULL);
```

REPLACE_BASE_DD procedure

This procedure does a create or replace on a row in the HS$_BASE_DD table.

First, this procedure attempts to update the row. If the row does not exist, then it is inserted, and the new_DD_TABLE_NAME parameter is ignored.

Syntax

```
DBMS_HS.REPLACE_BASE_DD (
   DD_TABLE_NAME     IN VARCHAR2,
   new_DD_TABLE_NAME IN VARCHAR2 := NULL,
   new_DD_TABLE_DESC IN VARCHAR2 := NULL);
```

REPLACE_CLASS_CAPS procedure

This procedure does a 'create or replace' on the HS$_CLASS_CAPS table.

If a row exists for the FDS_CLASS_NAME and CAP_NUMBER, then it is updated. If a row does not exist, then it is inserted, and the new_FDS_CLASS_NAME and new_CAP_NUMBER parameters are ignored.

Syntax

```
DBMS_HS.REPLACE_CLASS_CAPS (
   FDS_CLASS_NAME      IN VARCHAR2,
   CAP_NUMBER          IN NUMBER,
   new_FDS_CLASS_NAME  IN VARCHAR2 := NULL,
   new_CAP_NUMBER      IN NUMBER   := NULL,
   new_CONTEXT         IN NUMBER   := NULL,
   new_TRANSLATION     IN VARCHAR2 := NULL,
   new_ADDITIONAL_INFO IN NUMBER   := NULL);
```

REPLACE_CLASS_DD procedure

This procedure performs a 'create or replace' on the HS$_CLASS_DD table.

If a row exists for the FDS_CLASS_NAME and DD_TABLE_NAME, then the row is updated. If a row does not exist, then it is inserted, and the new_FDS_CLASS_NAME and new_DD_TABLE_NAME parameters are ignored.

Syntax

```
DBMS_HS.REPLACE_CLASS_DD (
   FDS_CLASS_NAME       IN VARCHAR2,
   DD_TABLE_NAME        IN VARCHAR2,
   new_FDS_CLASS_NAME   IN VARCHAR2 := NULL,
   new_DD_TABLE_NAME    IN VARCHAR2 := NULL,
   new_TRANSLATION_TYPE IN CHAR     := NULL,
   new_TRANSLATION_TEXT IN VARCHAR2 := NULL);
```

REPLACE_CLASS_INIT procedure

This procedure creates or updates a row in the HS$_CLASS_INIT table.

If a row exists with the specified FDS_CLASS_NAME and INIT_VALUE_NAME, then it is updated. If the row does not exist, then it is inserted, and the new_FDS_CLASS_NAME and new_INIT_VALUE_NAME parameters are ignored.

Syntax

```
DBMS_HS.REPLACE_CLASS_INIT (
   FDS_CLASS_NAME      IN VARCHAR2,
   INIT_VALUE_NAME     IN VARCHAR2,
   new_FDS_CLASS_NAME  IN VARCHAR2 := NULL,
   new_INIT_VALUE_NAME IN VARCHAR2 := NULL,
   new_INIT_VALUE      IN VARCHAR2 := NULL,
   new_INIT_VALUE_TYPE IN VARCHAR2 := NULL);
```

REPLACE_FDS_CLASS procedure

This procedure does create or replace operations on the HS$_FDS_CLASS table.

If a row exists for the FDS_CLASS_NAME, then it is updated. If no row exists, then it is created, and the new_FDS_CLASS_NAME parameter is ignored.

Syntax

```
DBMS_HS.REPLACE_FDS_CLASS (
   FDS_CLASS_NAME          IN VARCHAR2,
   new_FDS_CLASS_NAME      IN VARCHAR2 := NULL,
   new_FDS_CLASS_COMMENTS IN VARCHAR2 := NULL);
```

REPLACE_FDS_INST procedure

This procedure creates or replaces rows in the HS$_FDS_INST table.

If a row exists for the FDS_INST_NAME and FDS_CLASS_NAME, then it is updated. If no row exists, then it is created, and the new_FDS_INST_NAME and new_FDS_CLASS_NAME parameters are ignored when performing the insert.

Syntax

```
DBMS_HS.REPLACE_FDS_INST (
   FDS_INST_NAME          IN VARCHAR2,
   FDS_CLASS_NAME         IN VARCHAR2,
   new_FDS_INST_NAME      IN VARCHAR2 := NULL,
   new_FDS_CLASS_NAME     IN VARCHAR2 := NULL,
   new_FDS_INST_COMMENTS IN VARCHAR2 := NULL);
```

REPLACE_INST_CAPS procedure

This procedure does a create or replace on the HS$_INST_CAPS table.

If no row exists for the FDS_INST_NAME, FDS_CLASS_NAME and CAP_NUMBER, then the row is created. If a row exists, then it is updated. In the case where an insert is performed, the new_FDS_INST_NAME, new_FDS_CLASS_NAME and new_CLASS_NUMBER parameters are ignored.

Syntax

```
DBMS_HS.REPLACE_INST_CAPS (
   FDS_INST_NAME       IN VARCHAR2,
   FDS_CLASS_NAME      IN VARCHAR2,
   CAP_NUMBER          IN NUMBER,
   new_FDS_INST_NAME   IN VARCHAR2 := NULL,
   new_FDS_CLASS_NAME  IN VARCHAR2 := NULL,
   new_CAP_NUMBER      IN NUMBER   := NULL,
   new_CONTEXT         IN NUMBER   := NULL,
   new_TRANSLATION     IN VARCHAR2 := NULL,
   new_ADDITIONAL_INFO IN NUMBER   := NULL);
```

REPLACE_INST_DD procedure

This procedure performs a create or replace operation on the HS$_INST_DD table.

If a row exists for the FDS_INST_NAME, FDS_CLASS_NAME and DD_TABLE_NAME, then it is updated. If no row exists, then it is created, and the new_FDS_INST_NAME, new_FDS_CLASS_NAME, and new_DD_TABLE_NAME values are ignored.

Syntax

```
DBMS_HS.REPLACE_INST_DD (
   FDS_INST_NAME       IN VARCHAR2,
   FDS_CLASS_NAME      IN VARCHAR2,
   DD_TABLE_NAME       IN VARCHAR2,
   new_FDS_INST_NAME   IN VARCHAR2 := NULL,
   new_FDS_CLASS_NAME  IN VARCHAR2 := NULL,
   new_DD_TABLE_NAME   IN VARCHAR2 := NULL,
   new_TRANSLATION_TYPE IN CHAR    := NULL,
   new_TRANSLATION_TEXT IN VARCHAR2 := NULL);
```

REPLACE_INST_INIT procedure

This procedure performs a create or replace on the HS$_INST_INIT table.

If a row exists with the FDS_INST_NAME, FDS_CLASS_NAME and INIT_VALUE_NAME, then it is updated. If a row does not exist, then it is created. In the creation case, the new_FDS_INST_NAME, new_FDS_CLASS_NAME and new_INIT_VALUE_NAME are ignored.

Syntax

```
DBMS_HS.REPLACE_INST_INIT (
   FDS_INST_NAME       IN VARCHAR2,
   FDS_CLASS_NAME      IN VARCHAR2,
   INIT_VALUE_NAME     IN VARCHAR2,
   new_FDS_INST_NAME   IN VARCHAR2 := NULL,
   new_FDS_CLASS_NAME  IN VARCHAR2 := NULL,
   new_INIT_VALUE_NAME IN VARCHAR2 := NULL,
   new_INIT_VALUE      IN VARCHAR2 := NULL,
   new_INIT_VALUE_TYPE IN VARCHAR2 := NULL);
```

14

DBMS_HS_PASSTHROUGH

The pass-through SQL feature allows an application developer to send a statement directly to a non-Oracle system without being interpreted by the Oracle server. This can be useful if the non-Oracle system allows for operations in statements for which there is no equivalent in Oracle.

You can run these statements directly at the non-Oracle system using the PL/SQL package `DBMS_HS_PASSTHROUGH`. Any statement executed with this package is run in the same transaction as regular "transparent" SQL statements.

> **See Also:** For detailed information on Heterogeneous Services and on binding variables, see *Oracle8i Distributed Database Systems.*

Security

The DBMS_HS_PASSTHROUGH package conceptually resides at the non-Oracle system. Procedures and functions in the package must be called by using the appropriate database link to the non-Oracle system.

Summary of Subprograms

Table 14–1 DBMS_HS_PASSTHROUGH Package Subprograms

Subprogram	Description
BIND_VARIABLE procedure on page 14-3	Binds an IN variable positionally with a PL/SQL program variable.
BIND_VARIABLE_RAW procedure on page 14-4	Binds IN variables of type RAW.
BIND_OUT_VARIABLE procedure on page 14-5	Binds an OUT variable with a PL/SQL program variable.
BIND_OUT_VARIABLE_RAW procedure on page 14-7	Binds an OUT variable of datatype RAW with a PL/SQL program variable.
BIND_INOUT_VARIABLE procedure on page 14-8	Binds IN OUT bind variables.
BIND_INOUT_VARIABLE_RAW procedure on page 14-9	Binds IN OUT bind variables of datatype RAW.
CLOSE_CURSOR procedure on page 14-10	Closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system.
EXECUTE_IMMEDIATE procedure on page 14-11	Runs a (non-SELECT) SQL statement immediately, without bind variables.
EXECUTE_NON_QUERY function on page 14-12	Runs a (non-SELECT) SQL statement.
FETCH_ROW function on page 14-13	Fetches rows from a query.
GET_VALUE procedure on page 14-14	Retrieves column value from SELECT statement, or retrieves OUT bind parameters.
GET_VALUE_RAW procedure on page 14-16	Similar to GET_VALUE, but for datatype RAW.
OPEN_CURSOR function on page 14-17	Opens a cursor for running a passthrough SQL statement at the non-Oracle system.

Table 14–1 DBMS_HS_PASSTHROUGH Package Subprograms

Subprogram	Description
PARSE procedure on page 14-17	Parses SQL statement at non-Oracle system.

BIND_VARIABLE procedure

This procedure binds an IN variable positionally with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE (
   c      IN BINARY_INTEGER NOT NULL,
   pos    IN BINARY_INTEGER NOT NULL,
   val    IN <dty>,
   name   IN VARCHAR2);
```

Where <dty> is either DATE, NUMBER, or VARCHAR2

> **See Also:** To bind RAW variables use BIND_VARIABLE_RAW procedure on page 14-4.

Parameters

Table 14–2 BIND_VARIABLE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	Value that must be passed to the bind variable name.
name	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 14–3 BIND_VARIABLE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined: WNDS, RNDS

BIND_VARIABLE_RAW procedure

This procedure binds IN variables of type RAW.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW (
   c      IN BINARY_INTEGER NOT NULL,
   pos    IN BINARY_INTEGER NOT NULL,
   val    IN RAW,
   name   IN VARCHAR2);
```

Parameters

Table 14–4 BIND_VARIABLE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	Value that must be passed to the bind variable.

Table 14–4 BIND_VARIABLE_RAW Procedure Parameters

Parameter	Description
name	(Optional) Name of the bind variable. For example, in `SELECT * FROM emp WHERE ename=:ename`, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 14–5 BIND_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined : WNDS, RNDS

BIND_OUT_VARIABLE procedure

This procedure binds an OUT variable with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (
   c        IN  BINARY_INTEGER NOT NULL,
   pos      IN  BINARY_INTEGER NOT NULL,
   val      OUT <dty>,
   name     IN  VARCHAR2);
```

Where <dty> is either DATE, NUMBER, or VARCHAR2

See Also: For binding OUT variables of datatype RAW, see BIND_OUT_VARIABLE_RAW procedure on page 14-7.

Parameters

Table 14–6 BIND_OUT_VARIABLE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	Variable in which the OUT bind variable stores its value. The package remembers only the "size" of the variable. After the SQL statement is run, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE.
name	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 14–7 BIND_OUT_VARIABLE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined : WNDS, RNDS

BIND_OUT_VARIABLE_RAW procedure

This procedure binds an OUT variable of datatype RAW with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (
   c       IN  BINARY_INTEGER NOT NULL,
   pos     IN  BINARY_INTEGER NOT NULL,
   val     OUT RAW,
   name    IN  VARCHAR2);
```

Parameters

Table 14–8 BIND_OUT_VARIABLE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	Variable in which the OUT bind variable stores its value. The package remembers only the "size" of the variable. After the SQL statement is run, you can use GET_VALUE to retrieve the value of the OUT parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using BIND_OUT_VARIABLE_RAW.
name	(Optional) Name of the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 14–9 BIND_OUT_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.

Table 14–9 BIND_OUT_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined : WNDS, RNDS

BIND_INOUT_VARIABLE procedure

This procedure binds IN OUT bind variables.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE (
   c       IN     BINARY_INTEGER NOT NULL,
   pos     IN     BINARY_INTEGER NOT NULL,
   val     IN OUT <dty>,
   name    IN     VARCHAR2);
```

Where <dty> is either DATE, NUMBER, or VARCHAR2

> **See Also:** For binding IN OUT variables of datatype RAW see BIND_INOUT_VARIABLE_RAW procedure on page 14-9.

Parameters

Table 14–10 BIND_INOUT_VARIABLE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	This value is used for two purposes: - To provide the IN value before the SQL statement is run. - To determine the size of the out value.

Table 14–10 BIND_INOUT_VARIABLE Procedure Parameters

Parameter	Description
name	(Optional) Name of the bind variable. For example, in `SELECT * FROM emp WHERE ename=:ename`, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 14–11 BIND_INOUT_VARIABLE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined : WNDS, RNDS

BIND_INOUT_VARIABLE_RAW procedure

This procedure binds IN OUT bind variables of datatype RAW.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE (
   c        IN     BINARY_INTEGER NOT NULL,
   pos      IN     BINARY_INTEGER NOT NULL,
   val      IN OUT RAW,
   name     IN     VARCHAR2);
```

Parameters

Table 14–12 BIND_INOUT_VARIABLE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed' using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable in the SQL statement: Starts at 1.
val	This value is used for two purposes: - To provide the IN value before the SQL statement is run. - To determine the size of the out value.
name	(Optional) Name the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 14–13 BIND_INOUT_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined : WNDS, RNDS

CLOSE_CURSOR procedure

This function closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system. If the cursor was not open, then the operation is a "no operation".

Syntax

```
DBMS_HS_PASSTHROUGH.CLOSE_CURSOR (
   c IN BINARY_INTEGER NOT NULL);
```

Parameters

Table 14–14 CLOSE_CURSOR Procedure Parameters

Parameter	Description
c	Cursor to be released.

Exceptions

Table 14–15 CLOSE_CURSOR Procedure Exceptions

Exception	Description
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined : WNDS, RNDS

EXECUTE_IMMEDIATE procedure

This function runs a SQL statement immediately. Any valid SQL command except SELECT can be run immediately. The statement must not contain any bind variables. The statement is passed in as a VARCHAR2 in the argument. Internally the SQL statement is run using the PASSTHROUGH SQL protocol sequence of OPEN_CURSOR, PARSE, EXECUTE_NON_QUERY, CLOSE_CURSOR.

Syntax

```
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE (
   S IN VARCHAR2 NOT NULL)
RETURN BINARY_INTEGER;
```

Parameters

Table 14–16 EXECUTE_IMMEDIATE Procedure Parameters

Parameter	Description
s	VARCHAR2 variable with the statement to be executed immediately.

Returns

The number of rows affected by the execution of the SQL statement.

Exceptions

Table 14–17 EXECUTE_IMMEDIATE Procedure Exceptions

Exception	Description
ORA-28551	SQL statement is invalid.
ORA-28544	Max open cursors.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

None

EXECUTE_NON_QUERY function

This function runs a SQL statement. The SQL statement cannot be a SELECT statement. A cursor has to be open and the SQL statement has to be parsed before the SQL statement can be run.

Syntax

```
DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY (
   c IN BINARY_INTEGER NOT NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 14–18 EXECUTE_NON_QUERY Function Parameters

Parameter	Description
C	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.

Returns

The number of rows affected by the SQL statement in the non-Oracle system

Exceptions

Table 14–19 EXECUTE_NON_QUERY Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	BIND_VARIABLE procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

None

FETCH_ROW function

This function fetches rows from a result set. The result set is defined with a SQL SELECT statement. When there are no more rows to be fetched, the exception NO_DATA_FOUND is raised. Before the rows can be fetched, a cursor has to be opened, and the SQL statement has to be parsed.

Syntax

```
DBMS_HS_PASSTHROUGH.FETCH_ROW (
   c       IN BINARY_INTEGER NOT NULL,
   first   IN BOOLEAN)
  RETURN BINARY_INTEGER;
```

Parameters

Table 14–20 FETCH_ROW Function Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
first	(Optional) Reexecutes SELECT statement. Possible values: - TRUE: reexecute SELECT statement. - FALSE: fetch the next row, or if run for the first time, then execute and fetch rows (default).

Returns

The returns the number of rows fetched. The function returns "0" if the last row was already fetched.

Exceptions

Table 14–21 FETCH_ROW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS
```

GET_VALUE procedure

This procedure has two purposes:

- It retrieves the select list items of SELECT statements, after a row has been fetched.
- It retrieves the OUT bind values, after the SQL statement has been run.

Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE (
   c      IN  BINARY_INTEGER NOT NULL,
   pos    IN  BINARY_INTEGER NOT NULL,
   val    OUT <dty>);
```

Where <dty> is either DATE, NUMBER, or VARCHAR2

> **See Also:** For retrieving values of datatype RAW, see GET_VALUE_RAW procedure on page 14-16.

Parameters

Table 14–22 GET_VALUE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable or select list item in the SQL statement: Starts at 1.
val	Variable in which the OUT bind variable or select list item stores its value.

Exceptions

Table 14–23 GET_VALUE Procedure Exceptions

Exception	Description
ORA-1403	Returns NO_DATA_FOUND exception when running the GET_VALUE after the last row was fetched (i.e., FETCH_ROW returned "0").
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

```
Purity level defined : WNDS
```

GET_VALUE_RAW procedure

This procedure is similar to GET_VALUE, but for datatype RAW.

Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (
   c      IN  BINARY_INTEGER NOT NULL,
   pos    IN  BINARY_INTEGER NOT NULL,
   val    OUT RAW);
```

Parameters

Table 14–24 GET_VALUE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
pos	Position of the bind variable or select list item in the SQL statement: Starts at 1.
val	Variable in which the OUT bind variable or select list item stores its value.

Exceptions

Table 14–25 GET_VALUE_RAW Procedure Exceptions

Exception	Description
ORA-1403	Returns NO_DATA_FOUND exception when running the GET_VALUE after the last row was fetched (i.e., FETCH_ROW returned "0").
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined : WNDS

OPEN_CURSOR function

This function opens a cursor for running a pass-through SQL statement at the non-Oracle system. This function must be called for any type of SQL statement

The function returns a cursor, which must be used in subsequent calls. This call allocates memory. To deallocate the associated memory, call the procedure CLOSE_CURSOR.

Syntax

```
DBMS_HS_PASSTHROUGH.OPEN_CURSOR
  RETURN BINARY_INTEGER;
```

Returns

The cursor to be used on subsequent procedure and function calls.

Exceptions

Table 14–26 OPEN_CURSOR Function Exceptions

Exception	Description
ORA-28554	Maximum number of open cursor has been exceeded. Increase Heterogeneous Services' OPEN_CURSORS initialization parameter.

Pragmas

Purity level defined : WNDS, RNDS

PARSE procedure

This procedure parses SQL statement at non-Oracle system.

Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (
   c       IN  BINARY_INTEGER NOT NULL,
   stmt    IN  VARCHAR2       NOT NULL);
```

Parameters

Table 14–27 PARSE Procedure Parameters

Parameter	Description
c	Cursor associated with the pass-through SQL statement. Cursor must be opened using function OPEN_CURSOR.
stmt	Statement to be parsed.

Exceptions

Table 14–28 GET_VALUE Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28551	SQL statement is illegal.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

Pragmas

Purity level defined : WNDS, RNDS

15

DBMS_IOT

The DBMS_IOT package creates a table into which references to the chained rows for an index organized table can be placed using the ANALYZE command. It can also create an exception table into which rows of an index-organized table that violate a constraint can be placed during the enable_constraint operation.

Summary of Subprograms

Table 15–1 DBMS_IOT Package Subprograms

Subprogram	Description
BUILD_CHAIN_ROWS_TABLE procedure on page 15-2	Creates a table into which references to the chained rows for an index-organized table can be placed using the ANALYZE command.
BUILD_EXCEPTIONS_TABLE procedure on page 15-3	Creates an exception table into which rows of an index-organized table that violate a constraint can be placed during the enable_constraint operation.

BUILD_CHAIN_ROWS_TABLE procedure

The BUILD_CHAIN_ROWS_TABLE procedure creates a table into which references to the chained rows for an index-organized table can be placed using the ANALYZE command.

Syntax

```
DBMS_IOT.BUILD_CHAIN_ROWS_TABLE (
   owner               IN VARCHAR2,
   iot_name            IN VARCHAR2,
   chainrow_table_name IN VARCHAR2 default 'IOT_CHAINED_ROWS');
```

Parameters

Table 15–2 BUILD_CHAIN_ROWS_TABLE Procedure Parameters

Parameter	Description
owner	Owner of the index-organized table.
iot_name	Index-organized table name.
chainrow_table_name	Intended name for the chained-rows table.

Example

```
CREATE TABLE l(a char(16),b char(16), c char(16), d char(240),
PRIMARY KEY(a,b,c)) ORGANIZATION INDEX pctthreshold 10 overflow;
EXECUTE DBMS_IOT.BUILD_CHAIN_ROWS_TABLE('SYS','L','LC');
```

A chained-row table is created with the following columns:

```
Column Name                    Null?    Type
------------------------------ -------- ----
OWNER_NAME                              VARCHAR2(30)
TABLE_NAME                              VARCHAR2(30)
CLUSTER_NAME                            VARCHAR2(30)
PARTITION_NAME                          VARCHAR2(30)
SUBPARTITION_NAME                       VARCHAR2(30)
HEAD_ROWID                              ROWID
TIMESTAMP                               DATE
A                                       CHAR(16)
B                                       CHAR(16)
C                                       CHAR(16)
```

BUILD_EXCEPTIONS_TABLE procedure

The BUILD_EXCEPTIONS_TABLE procedure creates an exception table into which rows of an index-organized table that violate a constraint can be placed during the enable_constraint operation.

A separate chained-rows table and an exception table should be created for each index-organized table to accommodate its primary key.

> **Note:** This form of chained-rows table and exception table are required only for servers running with Oracle8, Release 8.0 compatibility.

Syntax

```
DBMS_IOT.BUILD_EXCEPTIONS_TABLE (
   owner                 IN VARCHAR2,
   iot_name              IN VARCHAR2,
   exceptions_table_name IN VARCHAR2 default 'IOT_EXCEPTIONS');
```

Parameters

Table 15–3 BUILD_EXCEPTIONS_TABLE Procedure Parameters

Parameter	Description
owner	Owner of the index-organized table.
iot_name	Index-organized table name.
exceptions_table_name	Intended name for exception-table.

Example

```
EXECUTE DBMS_IOT.BUILD_EXCEPTIONS_TABLE('SYS','L','LE');
```

An exception table for the above index-organized table with the following columns:

```
Column Name                    Null?    Type
------------------------------ -------- ----
ROW_ID                                  VARCHAR2(30)
OWNER                                   VARCHAR2(30)
TABLE_NAME                              VARCHAR2(30)
CONSTRAINT                              VARCHAR2(30)
A                                       CHAR(16)
B                                       CHAR(16)
C                                       CHAR(16)
```

16

DBMS_JOB

DBMS_JOB subprograms schedule and manage jobs in the job queue.

See Also: For more information on the DBMS_JOB package and the job queue, see *Oracle8i Administrator's Guide*.

Requirements

There are no database privileges associated with jobs. The right to execute DBMS_JOB or takes the place. DBMS_JOB does not allow a user to touch any jobs except their own.

Summary of Subprograms

Table 16–1 DBMS_JOB Package Subprograms

Subprogram	Description
SUBMIT procedure on page 16-3	Submits a new job to the job queue.
REMOVE procedure on page 16-4	Removes specified job from the job queue.
CHANGE procedure on page 16-5	Alters any of the user-definable parameters associated with a job.
WHAT procedure on page 16-6	Alters the job description for a specified job.
NEXT_DATE procedure on page 16-7	Alters the next execution time for a specified job.
INSTANCE procedure on page 16-7	Assigns a job to be run by a instance.
INTERVAL procedure on page 16-8	Alters the interval between executions for a specified job.
BROKEN procedure on page 16-9	Disables job execution.
RUN procedure on page 16-9	Forces a specified job to run.
USER_EXPORT procedure on page 16-10	Recreates a given job for export.
USER_EXPORT procedure on page 16-10	Recreates a given job for export with instance affinity.

SUBMIT procedure

This procedure submits a new job. It chooses job from the sequence sys.jobseq.

Syntax

```
DBMS_JOB.SUBMIT (
   job       OUT BINARY_INTEGER,
   what      IN  VARCHAR2,
   next_date IN  DATE DEFAULT sysdate,
   interval  IN  VARCHAR2 DEFAULT 'null',
   no_parse  IN  BOOLEAN DEFAULT FALSE,
   instance  IN  BINARY_INTEGER DEFAULT any_instance,
   force     IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 16–2 SUBMIT Procedure Parameters

Parameter	Description
job	Number of the job being run.
what	PL/SQL procedure to run.
next_date	Next date when the job will be run.
interval	Date function that calculates the next time to run the job. The default is NULL. This must evaluate to a either a future point in time or NULL.
no_parse	A flag. The default is FALSE. If this is set to FALSE, then Oracle parses the procedure associated with the job. If this is set to TRUE, then Oracle parses the procedure associated with the job the first time that the job is run. For example, if you want to submit a job before you have created the tables associated with the job, then set this to TRUE.
instance	When a job is submitted, specifies which instance can run the job.
force	If this is TRUE, then any positive integer is acceptable as the job instance. If this is FALSE (the default), then the specified instance must be running; otherwise the routine raises an exception.

Usage Notes

The parameters `instance` and `force` are added for job queue affinity. Job queue affinity gives users the ability to indicate whether a particular instance or any instance can run a submitted job.

Example

This submits a new job to the job queue. The job calls the procedure DBMS_DDL.ANALYZE_OBJECT to generate optimizer statistics for the table DQUON.ACCOUNTS. The statistics are based on a sample of half the rows of the ACCOUNTS table. The job is run every 24 hours:

```
VARIABLE jobno number;
BEGIN
   DBMS_JOB.SUBMIT(:jobno,
      'dbms_ddl.analyze_object(''TABLE'',
      ''DQUON'', ''ACCOUNTS'',
      ''ESTIMATE'', NULL, 50);'
      SYSDATE, 'SYSDATE + 1');
   commit;
END;
/
Statement processed.
print jobno
JOBNO
----------
14144
```

REMOVE procedure

This procedure removes an existing job from the job queue. This currently does not stop a running job.

Syntax

```
DBMS_JOB.REMOVE (
   job       IN  BINARY_INTEGER );
```

Parameters

Table 16–3 REMOVE Procedure Parameters

Parameter	Description
job	Number of the job being run.

Example

```
EXECUTE DBMS_JOB.REMOVE(14144);
```

CHANGE procedure

This procedure changes any of the user-settable fields in a job.

Syntax

```
DBMS_JOB.CHANGE (
   job       IN  BINARY_INTEGER,
   what      IN  VARCHAR2,
   next_date IN  DATE,
   interval  IN  VARCHAR2,
   instance  IN  BINARY_INTEGER DEFAULT NULL,
   force     IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 16–4 CHANGE Procedure Parameters

Parameter	Description
job	Number of the job being run.
what	PL/SQL procedure to run.
next_date	Date of the next refresh.
interval	Date function; evaluated immediately before the job starts running.
instance	When a job is submitted, specifies which instance can run the job. This defaults to NULL, which indicates that instance affinity is not changed.

Table 16–4 CHANGE Procedure Parameters

Parameter	Description
force	If this is FALSE, then the specified instance (to which the instance number change) must be running. Otherwise, the routine raises an exception. If this is TRUE, then any positive integer is acceptable as the job instance.

Usage Notes

The parameters instance and force are added for job queue affinity. Job queue affinity gives users the ability to indicate whether a particular instance or any instance can run a submitted job.

If the parameters what, next_date, or interval are NULL, then leave that value as it is.

Example

```
EXECUTE DBMS_JOB.CHANGE(14144, null, null, 'sysdate+3');
```

WHAT procedure

This procedure changes what an existing job does, and replaces its environment.

Syntax

```
DBMS_JOB.WHAT (
   job       IN  BINARY_INTEGER,
   what      IN  VARCHAR2);
```

Parameters

Table 16–5 WHAT Procedure Parameters

Parameter	Description
job	Number of the job being run.
what	PL/SQL procedure to run.

Some legal values of what (assuming the routines exist) are:

- `'myproc( ''10-JAN-82'', next_date, broken);'`

- `'scott.emppackage.give_raise( ''JENKINS'', 30000.00);'`
- `'dbms_job.remove(job);'`

NEXT_DATE procedure

This procedure changes when an existing job next runs.

Syntax

```
DBMS_JOB.NEXT_DATE (
   job       IN  BINARY_INTEGER,
   next_date IN  DATE);
```

Parameters

Table 16–6 NEXT_DATE Procedure Parameters

Parameter	Description
job	Number of the job being run.
next_date	Date of the next refresh: it is when the job will be automatically run, assuming there are background processes attempting to run it.

INSTANCE procedure

This procedure changes job instance affinity.

Syntax

```
DBMS_JOB.INSTANCE (
   job        IN BINARY_INTEGER,
   instance   IN BINARY_INTEGER,
   force      IN BOOLEAN DEFAULT FALSE);
```

Parameters

Table 16–7 INSTANCE Procedure Parameters

Parameter	Description
job	Number of the job being run.

Table 16–7 INSTANCE Procedure Parameters

Parameter	Description
instance	When a job is submitted, a user can specify which instance can run the job.
force	If this is TRUE, then any positive integer is acceptable as the job instance. If this is FALSE (the default), then the specified instance must be running; otherwise the routine raises an exception.

INTERVAL procedure

This procedure changes how often a job runs.

Syntax

```
DBMS_JOB.INTERVAL (
   job       IN  BINARY_INTEGER,
   interval  IN  VARCHAR2);
```

Parameters

Table 16–8 INTERVAL Procedure Parameters

Parameter	Description
job	Number of the job being run.
interval	Date function, evaluated immediately before the job starts running.

Usage Notes

If the job completes successfully, then this new date is placed in `next_date`. `interval` is evaluated by plugging it into the statement select `interval` into `next_date` from dual;

The `interval` parameter must evaluate to a time in the future. Legal intervals include:

`'sysdate + 7'`	Run once a week.
`'next_day(sysdate,''TUESDAY'')'`	Run once every Tuesday.
`'null'`	Run only once.

If `interval` evaluates to `NULL` and if a job completes successfully, then the job is automatically deleted from the queue.

BROKEN procedure

This procedure sets the broken flag. Broken jobs are never run.

Syntax

```
DBMS_JOB.BROKEN (
   job       IN  BINARY_INTEGER,
   broken    IN  BOOLEAN,
   next_date IN  DATE DEFAULT SYSDATE);
```

Parameters

Table 16–9 Broken Procedure Parameters

Parameter	Description
job	Number of the job being run.
broken	Job broken: `IN` value is `FALSE`.
next_data	Date of the next refresh.

RUN procedure

This procedure runs job `JOB` now. It runs it even if it is broken.

Running the job recomputes `next_date`. See view `user_jobs`.

Syntax

```
DBMS_JOB.RUN (
   job       IN  BINARY_INTEGER,
   force     IN  BOOLEAN DEFAULT FALSE);
```

Parameters

Table 16–10 Run Procedure Parameters

Parameter	Description
job	Number of the job being run.

Table 16–10 Run Procedure Parameters

Parameter	Description
force	If this is TRUE, then instance affinity is irrelevant for running jobs in the foreground process. If this is FALSE, then the job can be run in the foreground only in the specified instance.

Example

```
EXECUTE DBMS_JOB.RUN(14144);
```

Caution: This reinitializes the current session's packages.

Exceptions

An exception is raised if force is FALSE, and if the connected instance is the wrong one.

USER_EXPORT procedure

This procedure produces the text of a call to recreate the given job.

Syntax

```
DBMS_JOB.USER_EXPORT (
   job    IN     BINARY_INTEGER,
   mycall IN OUT VARCHAR2);
```

Parameters

Table 16–11 USER_EXPORT Procedure Parameter

Parameter	Description
job	Number of the job being run.
mycall	Text of a call to recreate the given job.

USER_EXPORT procedure

This procedure alters instance affinity (8*i* and above) and preserves the compatibility.

Syntax

```
DBMS_JOB.USER_EXPORT (
   job      IN     BINARY_INTEGER,
   mycall   IN OUT VARCHAR2,
   myinst   IN OUT VARCHAR2);
```

Parameters

Table 16–12 USER_EXPORT Procedure Parameters

Parameter	Description
job	Number of the job being run.
mycall	Text of a call to recreate a given job.
myinst	Text of a call to alter instance affinity.

17

DBMS_LOB

The DBMS_LOB package provides subprograms to operate on BLOBs, CLOBs, NCLOBs, BFILEs, and temporary LOBs. You can use DBMS_LOB to access and manipulation specific parts of a LOB or complete LOBs.

DBMS_LOB can read and modify BLOBs, CLOBs, and NCLOBs; it provides read-only operations for BFILEs. The bulk of the LOB operations are provided by this package.

> **See Also:** *Oracle8i Application Developer's Guide - Large Objects (LOBs)*

Requirements

This package must be created under SYS (connect internal). Operations provided by this package are performed under the current calling user, not under the package owner SYS.

LOB Locators

All DBMS_LOB subprograms work based on LOB locators. For the successful completion of DBMS_LOB subprograms, you must provide an input locator that represents a LOB that already exists in the database tablespaces or external filesystem.

Internal LOBs For internal LOBs, you must first use SQL data definition language (DDL) to define tables that contain LOB columns, and, subsequently, use SQL data manipulation language (DML) to initialize or populate the locators in these LOB columns.

External LOBs For external LOBs, you must ensure that a DIRECTORY object representing a valid, existing physical directory has been defined, and that physical files exist with read permission for Oracle. If your operating system uses case-sensitive pathnames, then be sure you specify the directory in the correct format.

After the LOBs are defined and created, you may then SELECT a LOB locator into a local PL/SQL LOB variable and use this variable as an input parameter to DBMS_LOB for access to the LOB value.

Temporary LOBs For temporary LOBs, you must use the OCI, PL/SQL, or another programmatic interface to create or manipulate them. Temporary LOBs can be either BLOBs, CLOBs, or NCLOBs.

Datatypes

Parameters for the DBMS_LOB subprograms use these datatypes:

Table 17–1 DBMS_LOB datatypes

BLOB	A source or destination binary LOB.
RAW	A source or destination RAW buffer (used with BLOB).
CLOB	A source or destination character LOB (including NCLOB).
VARCHAR2	A source or destination character buffer (used with CLOB and NCLOB).
INTEGER	Specifies the size of a buffer or LOB, the offset into a LOB, or the amount to access.
BFILE	A large, binary object stored outside the database.

The DBMS_LOB package defines no special types. NCLOB is a special case of CLOBs for fixed-width and varying-width, multi-byte national character sets. The clause ANY_CS in the specification of DBMS_LOB subprograms for CLOBs enables them to accept a CLOB or NCLOB locator variable as input.

Constants

DBMS_LOB defines the following constants:

```
file_readonly CONSTANT BINARY_INTEGER := 0;
lob_readonly  CONSTANT BINARY_INTEGER := 0;
lob_readwrite CONSTANT BINARY_INTEGER := 1;
lobmaxsize    CONSTANT INTEGER        := 4294967295;
call          CONSTANT PLS_INTEGER    := 12;
session       CONSTANT PLS_INTEGER    := 10;
```

Oracle supports a maximum LOB size of 4 gigabytes (2^{32}). However, the amount and offset parameters of the package can have values between 1 and 4294967295 (2^{32}-1).

The PL/SQL 3.0 language specifies that the maximum size of a RAW or VARCHAR2 variable is 32767 *bytes*.

Note: The value 32767 bytes is represented by maxbufsize in the following sections.

Exceptions

Table 17–2 DBMS_LOB Exceptions

Exception	Code	Description
invalid_argval	21560	The argument is expecting a non-NULL, valid value but the argument value passed in is NULL, invalid, or out of range.
access_error	22925	You are trying to write too much data to the LOB: LOB size is limited to 4 gigabytes.
noexist_directory	22285	The directory leading to the file does not exist.
nopriv_directory	22286	The user does not have the necessary access privileges on the directory alias and / or the file for the operation.
invalid_directory	22287	The directory alias used for the current operation is not valid if being accessed for the first time, or if it has been modified by the DBA since the last access.
operation_failed	22288	The operation attempted on the file failed.
unopened_file	22289	The file is not open for the required operation to be performed.
open_toomany	22290	The number of open files has reached the maximum limit.

Security

Any DBMS_LOB subprogram called from an anonymous PL / SQL block is executed using the privileges of the current user. Any DBMS_LOB subprogram called from a stored procedure is executed using the privileges of the owner of the stored procedure.

With Oracle8*i*, when creating the procedure, users can set the AUTHID to indicate whether they want definer's rights or invoker's rights. For example:

```
CREATE PROCEDURE proc1 authid definer ...
```

or

```
CREATE PROCEDURE proc1 authid current_user ....
```

> **See Also:** For more information on AUTHID and privileges, see *PL/SQL User's Guide and Reference.*

You can provide secure access to BFILEs using the DIRECTORY feature discussed in BFILENAME function in the *Oracle8i Application Developer's Guide - Large Objects (LOBs)* and the *Oracle8i SQL Reference.*

Rules and Limitations

- The following rules apply in the specification of subprograms in this package:
  - length and offset parameters for subprograms operating on BLOBs and BFILEs must be specified in terms of *bytes*.
  - length and offset parameters for subprograms operating on CLOBs must be specified in terms of *characters*.
  - offset and amount parameters are always in *characters* for CLOBs / NCLOBs and in *bytes* for BLOBs / BFILEs.
- A subprogram raises an INVALID_ARGVAL exception if the following restrictions are not followed in specifying values for parameters (unless otherwise specified):
  1. Only positive, absolute offsets from the beginning of LOB data are permitted: Negative offsets from the tail of the LOB are not permitted.
  2. Only positive, non-zero values are permitted for the parameters that represent size and positional quantities, such as amount, offset, newlen, nth, etc. Negative offsets and ranges observed in Oracle SQL string functions and operators are not permitted.
  3. The value of offset, amount, newlen, nth must not exceed the value lobmaxsize (4GB-1) in any DBMS_LOB subprogram.
  4. For CLOBs consisting of fixed-width multi-byte characters, the maximum value for these parameters must not exceed (lobmaxsize / character_width_in_bytes) characters.

     For example, if the CLOB consists of 2-byte characters, such as:

     ```
     JA16SJISFIXED
     ```

     Then, the maximum amount value should not exceed:

     ```
     4294967295/2 = 2147483647 characters.
     ```

- PL / SQL language specifications stipulate an upper limit of 32767 bytes (not characters) for RAW and VARCHAR2 parameters used in DBMS_LOB subprograms. For example, if you declare a variable to be:

  ```
  charbuf VARCHAR2(3000)
  ```

Then, charbuf can hold 3000 single byte characters or 1500 2-byte fixed width characters. This has an important consequence for DBMS_LOB subprograms for CLOBs and NCLOBs.

- The %CHARSET clause indicates that the form of the parameter with %CHARSET must match the form of the ANY_CS parameter to which it refers.

  For example, in DBMS_LOB subprograms that take a VARCHAR2 buffer parameter, the form of the VARCHAR2 buffer must match the form of the CLOB parameter. If the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

  For DBMS_LOB subprograms that take two CLOB parameters, both CLOB parameters must have the same form; i.e., they must both be NCLOBs, or they must both be CLOBs.

- If the value of amount plus the offset exceeds 4 GB (i.e., lobmaxsize+1) for BLOBs and BFILEs, and (lobmaxsize/character_width_in_bytes)+1 for CLOBs in calls to update subprograms (i.e., APPEND, COPY, TRIM, WRITE and WRITEAPPEND subprograms), then access exceptions are raised.

  Under these input conditions, read subprograms, such as READ, COMPARE, INSTR, and SUBSTR, read until End of Lob/File is reached. For example, for a READ operation on a BLOB or BFILE, if the user specifies offset value of 3 GB and an amount value of 2 GB, then READ reads only ((4GB-1)-3GB) bytes.

- Functions with NULL or invalid input values for parameters return a NULL. Procedures with NULL values for destination LOB parameters raise exceptions.

- Operations involving patterns as parameters, such as COMPARE, INSTR, and SUBSTR do not support regular expressions or special matching characters (such as % in the LIKE operator in SQL) in the pattern parameter or substrings.

- The End Of LOB condition is indicated by the READ procedure using a NO_DATA_FOUND exception. This exception is raised only upon an attempt by the user to read beyond the end of the LOB/FILE. The READ buffer for the last read contains 0 bytes.

- For consistent LOB updates, you must lock the row containing the destination LOB before making a call to any of the procedures (mutators) that modify LOB data.

- Unless otherwise stated, the default value for an offset parameter is 1, which indicates the first byte in the BLOB or BFILE data, and the first character in the

CLOB or NCLOB value. No default values are specified for the amount parameter — you must input the values explicitly.

- You must lock the row containing the destination internal LOB before calling any subprograms that modify the LOB, such as APPEND, COPY, ERASE, TRIM, or WRITE. These subprograms do not implicitly lock the row containing the LOB.

BFILE-Specific Rules and Limitations

- The subprograms COMPARE, INSTR, READ, SUBSTR, FILECLOSE, FILECLOSEALL and LOADFROMFILE operate only on an *opened* BFILE locator; i.e., a successful FILEOPEN call must precede a call to any of these subprograms.
- For the functions FILEEXISTS, FILEGETNAME and GETLENGTH, a file's open/close status is unimportant; however, the file must exist physically, and you must have adequate privileges on the DIRECTORY object and the file.
- DBMS_LOB does not support any concurrency control mechanism for BFILE operations.
- In the event of several open files in the session whose closure has not been handled properly, you can use the FILECLOSEALL subprogram to close all files opened in the session and resume file operations from the beginning.
- If you are the creator of a DIRECTORY, or if you have system privileges, then use the CREATE OR REPLACE, DROP, and REVOKE statements in SQL with extreme caution.

  If you, or other grantees of a particular directory object, have several open files in a session, then any of the above commands can adversely affect file operations. In the event of such abnormal termination, your only choice is to invoke a program or anonymous block that calls FILECLOSEALL, reopen your files, and restart your file operations.
- All files opened during a user session are implicitly closed at the end of the session. However, Oracle strongly recommends that you close the files after *both* normal and abnormal termination of operations on the BFILE.

  In the event of normal program termination, proper file closure ensures that the number of files that are open simultaneously in the session remains less than SESSION_MAX_OPEN_FILES.

  In the event of abnormal program termination from a PL/SQL program, it is imperative that you provide an exception handler that ensures closure of all files opened in that PL/SQL program. This is necessary because after an

exception occurs, only the exception handler has access to the BFILE variable in its most current state.

After the exception transfers program control outside the PL/SQL program block, all references to the open BFILEs are lost. The result is a larger open file count which may or may not exceed the SESSION_MAX_OPEN_FILES value.

For example, consider a READ operation past the end of the BFILE value, which generates a NO_DATA_FOUND exception:

```
DECLARE
       fil BFILE;
       pos INTEGER;
       amt BINARY_INTEGER;
       buf RAW(40);
BEGIN
       SELECT f_lob INTO fil FROM lob_table WHERE key_value = 21;
       dbms_lob.open(fil, dbms_lob.lob_readonly);
       amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
       dbms_lob.read(fil, amt, pos, buf);
       dbms_output.put_line('Read F1 past EOF: '||
           utl_raw.cast_to_varchar2(buf));
       dbms_lob.close(fil);
END;

ORA-01403: no data found
ORA-06512: at "SYS.DBMS_LOB", line 373
ORA-06512: at line 10
```

After the exception has occurred, the BFILE locator variable file goes out of scope, and no further operations on the file can be done using that variable. Therefore, the solution is to use an exception handler:

```
DECLARE
     fil BFILE;
     pos INTEGER;
     amt BINARY_INTEGER;
     buf RAW(40);
BEGIN
     SELECT f_lob INTO fil FROM lob_table WHERE key_value = 21;
     dbms_lob.open(fil, dbms_lob.lob_readonly);
     amt := 40; pos := 1 + dbms_lob.getlength(fil); buf := '';
     dbms_lob.read(fil, amt, pos, buf);
     dbms_output.put_line('Read F1 past EOF: '||
          utl_raw.cast_to_varchar2(buf));
     dbms_lob.close(fil);
```

```
      exception
      WHEN no_data_found
      THEN
        BEGIN
          dbms_output.put_line('End of File reached. Closing file');
          dbms_lob.fileclose(fil);
          -- or dbms_lob.filecloseall if appropriate
        END;
END;
      /

Statement processed.
End of File reached. Closing file
```

In general, you should ensure that files opened in a PL/SQL block using DBMS_LOB are closed before normal or abnormal termination of the block.

Temporary LOBs

Oracle8*i* supports the definition, creation, deletion, access, and update of temporary LOBs. Your temporary tablespace stores the temporary LOB data. Temporary LOBs are not permanently stored in the database. Their purpose is mainly to perform transformations on LOB data.

A temporary LOB is empty when it is created. By default, all temporary LOBs are deleted at the end of the session in which they were created. If a process dies unexpectedly or if the database crashes, then temporary LOBs are deleted, and the space for temporary LOBs is freed.

In Oracle8*i*, there is also an interface to let you group temporary LOBs together into a logical bucket. The duration represents this logical store for temporary LOBs. Each temporary LOB can have separate storage characteristics, such as CACHE / NOCACHE. There is a default store for every session into which temporary LOBs are placed if you don't specify a specific duration. Additionally, you are able to perform a free operation on durations, which causes all contents in a duration to be freed.

There is no support for consistent read (CR), undo, backup, parallel processing, or transaction management for temporary LOBs. Because CR and rollbacks are not supported for temporary LOBs, you must free the temporary LOB and start over again if you encounter an error.

Because CR, undo, and versions are not generated for temporary LOBs, there is potentially a performance impact if you assign multiple locators to the same

temporary LOB. Semantically, each locator should have its own copy of the temporary LOB.

A copy of a temporary LOB is created if the user modifies the temporary LOB while another locator is also pointing to it. The locator on which a modification was performed now points to a new copy of the temporary LOB. Other locators no longer see the same data as the locator through which the modification was made. A deep copy was not incurred by permanent LOBs in these types of situations, because CR snapshots and version pages enable users to see their own versions of the LOB cheaply.

You can gain pseudo-REF semantics by using pointers to locators in OCI and by having multiple pointers to locators point to the same temporary LOB locator, if necessary. In PL/SQL, you must avoid using more than one locator per temporary LOB. The temporary LOB locator can be passed "by ref" to other procedures.

Because temporary LOBs are not associated with any table schema, there are no meanings to the terms in-row and out-of-row temporary LOBs. Creation of a temporary LOB instance by a user causes the engine to create and return a 'locator' to the LOB data. The PL/SQL DBMS_LOB package, PRO*C, OCI, and other programmatic interfaces operate on temporary LOBs through these locators just as they do for permanent LOBs.

There is no support for client side temporary LOBs. All temporary LOBs reside in the server.

Temporary LOBs do not support the EMPTY_BLOB or EMPTY_CLOB functions that are supported for permanent LOBs. The EMPTY_BLOB function specifies the fact that the LOB is initialized, but not populated with any data.

A temporary LOB instance can only be destroyed by using OCI or the DBMS_LOB package by using the appropriate FREETEMPORARY or OCIDurationEnd statement.

A temporary LOB instance can be accessed and modified using appropriate OCI and DBMS_LOB statements, just as for regular permanent internal LOBs. To make a temporary LOB permanent, you must explicitly use the OCI or DBMS_LOB COPY command, and copy the temporary LOB into a permanent one.

Security is provided through the LOB locator. Only the user who created the temporary LOB is able to see it. Locators are not expected to be able to pass from one user's session to another. Even if someone did pass a locator from one session to another, they would not access the temporary LOBs from the original session. Temporary LOB lookup is localized to each user's own session. Someone using a locator from somewhere else is only able to access LOBs within his own session that

have the same LOB ID. Users should not try to do this, but if they do, they are not able to affect anyone else's data.

Oracle keeps track of temporary LOBs per session in a v$ view called V$TEMPORARY_LOBS, which contains information about how many temporary LOBs exist per session. V$ views are for DBA use. From the session, Oracle can determine which user owns the temporary LOBs. By using V$TEMPORARY_LOBS in conjunction with DBA_SEGMENTS, a DBA can see how much space is being used by a session for temporary LOBs. These tables can be used by DBAs to monitor and guide any emergency cleanup of temporary space used by temporary LOBs.

Temporary LOBs Usage Notes

1. All functions in DBMS_LOB return NULL if any of the input parameters are NULL. All procedures in DBMS_LOB raise an exception if the LOB locator is input as NULL.
2. Operations based on CLOBs do not verify if the character set IDs of the parameters (CLOB parameters, VARCHAR2 buffers and patterns, etc.) match. It is the user's responsibility to ensure this.
3. Data storage resources are controlled by the DBA by creating different temporary tablespaces. DBAs can define separate temporary tablespaces for different users, if necessary.
4. Temporary LOBs still adhere to value semantics in order to be consistent with permanent LOBs and to try to conform to the ANSI standard for LOBs. As a result, each time a user does an OCILobLocatatorAssign, or the equivalent assignment in PL/SQL, the database makes a copy of the temporary LOB.

   Each locator points to its own LOB value. If one locator is used to create a temporary LOB, and if it is assigned to another LOB locator using OCILobLOcatorAssign in OCI or through an assignment operation in PL/SQL, then the database copies the original temporary LOB and causes the second locator to point to the copy.

   In order for users to modify the same LOB, they must go through the same locator. In OCI, this can be accomplished fairly easily by using pointers to locators and assigning the pointers to point to the same locator. In PL/SQL, the same LOB variable must be used to update the LOB to get this effect.

   The following example shows a place where a user incurs a copy, or at least an extra roundtrip to the server.

   ```
   DECLARE
   ```

```
  a blob;
  b blob;
BEGIN
  dbms_lob.createtemporary(b, TRUE, dbms_lob.session);
  -- the following assignment results in a deep copy
  a := b;
END;
```

The PL/SQL compiler makes temporary copies of actual arguments bound to OUT or IN OUT parameters. If the actual parameter is a temporary LOB, then the temporary copy is a deep (value) copy.

The following PL/SQL block illustrates the case where the user incurs a deep copy by passing a temporary LOB as an IN OUT parameter.

```
DECLARE
  a blob;
  procedure foo(parm IN OUT blob) is
  BEGIN
   ...
  END;
BEGIN
  dbms_lob.createtemporary(a, TRUE, dbms_lob.session);
  -- the following call results in a deep copy of the blob a
  foo(a);
END;
```

To minimize deep copies on PL/SQL parameter passing, use the NOCOPY compiler hint where possible.

> **See Also:** For more information on NOCOPY syntax, see *PL/SQL User's Guide and Reference.*

Temporary LOB Exceptions

Table 17–3 DBMS_LOB Package Exceptions

Exception	Code	Description
INVALID_ARGVAL	21560	Value for argument %s is not valid.
ACCESS_ERROR	22925	Attempt to read or write beyond maximum LOB size on %s.
NO_DATA_FOUND		EndofLob indicator for looping read operations. This is not a hard error.
VALUE_ERROR	6502	PL/SQL error for invalid values to subprogram's parameters.

Summary of Subprograms

Table 17–4 DBMS_LOB Subprograms **(Page 1 of 2)**

Subprogram	Description
APPEND procedure on page 17-14	Appends the contents of the source LOB to the destination LOB.
CLOSE procedure on page 17-16	Closes a previously opened internal or external LOB.
COMPARE function on page 17-17	Compares two entire LOBs or parts of two LOBs.
COPY procedure on page 17-19	Copies all, or part, of the source LOB to the destination LOB.
CREATETEMPORARY procedure on page 17-22	Creates a temporary BLOB or CLOB and its corresponding index in the user's default temporary tablespace.
ERASE procedure on page 17-23	Erases all or part of a LOB.
FILECLOSE procedure on page 17-25	Closes the file.
FILECLOSEALL procedure on page 17-26	Closes all previously opened files.
FILEEXISTS function on page 17-27	Checks if the file exists on the server.
FILEGETNAME procedure on page 17-29	Gets the directory alias and file name.
FILEISOPEN function on page 17-30	Checks if the file was opened using the input BFILE locators.
FILEOPEN procedure on page 17-32	Opens a file.
FREETEMPORARY procedure on page 17-33	Frees the temporary BLOB or CLOB in the user's default temporary tablespace.
GETCHUNKSIZE function on page 17-34	Returns the amount of space used in the LOB chunk to store the LOB value.
GETLENGTH function on page 17-35	Gets the length of the LOB value.

Table 17–4 DBMS_LOB Subprograms **(Page 2 of 2)**

Subprogram	Description
INSTR function on page 17-37	Returns the matching position of the *nth* occurrence of the pattern in the LOB.
ISOPEN function on page 17-40	Checks to see if the LOB was already opened using the input locator.
ISTEMPORARY function on page 17-41	Checks if the locator is pointing to a temporary LOB.
LOADFROMFILE procedure on page 17-42	Loads BFILE data into an internal LOB.
OPEN procedure on page 17-44	Opens a LOB (internal, external, or temporary) in the indicated mode.
READ procedure on page 17-45	Reads data from the LOB starting at the specified offset.
SUBSTR function on page 17-49	Returns part of the LOB value starting at the specified offset.
TRIM procedure on page 17-52	Trims the LOB value to the specified shorter length.
WRITE procedure on page 17-53	Writes data to the LOB from a specified offset.
WRITEAPPEND procedure on page 17-56	Writes a buffer to the end of a LOB.

APPEND procedure

This procedure appends the contents of a source internal LOB to a destination LOB. It appends the complete source LOB.

There are two overloaded APPEND procedures.

Syntax

```
DBMS_LOB.APPEND (
   dest_lob IN OUT  NOCOPY BLOB,
   src_lob  IN             BLOB);

DBMS_LOB.APPEND (
   dest_lob IN OUT  NOCOPY CLOB  CHARACTER SET ANY_CS,
   src_lob  IN             CLOB  CHARACTER SET dest_lob%CHARSET);
```

Pragmas

None.

Parameters

Table 17–5 APPEND Procedure Parameters

Parameter	Description
dest_lob	Locator for the internal LOB to which the data is to be appended.
src_lob	Locator for the internal LOB from which the data is to be read.

Exceptions

Table 17–6 APPEND Procedure Exceptions

Exception	Description
VALUE_ERROR	Either the source or the destination LOB is NULL.

Examples

```
CREATE OR REPLACE PROCEDURE Example_1a IS
    dest_lob, src_lob  BLOB;
BEGIN
    -- get the LOB locators
    -- note that the FOR UPDATE clause locks the row
    SELECT b_lob INTO dest_lob
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    SELECT b_lob INTO src_lob
        FROM lob_table
        WHERE key_value = 21;
    DBMS_LOB.APPEND(dest_lob, src_lob);
    COMMIT;
EXCEPTION
    WHEN some_exception
    THEN handle_exception;
END;

CREATE OR REPLACE PROCEDURE Example_1b IS
    dest_lob, src_lob  BLOB;
BEGIN
    -- get the LOB locators
```

```
    -- note that the FOR UPDATE clause locks the row
    SELECT b_lob INTO dest_lob
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    SELECT b_lob INTO src_lob
        FROM lob_table
        WHERE key_value = 12;
    DBMS_LOB.APPEND(dest_lob, src_lob);
    COMMIT;
EXCEPTION
    WHEN some_exception
    THEN handle_exception;
END;
```

CLOSE procedure

This procedure closes a previously opened internal or external LOB.

Syntax

```
DBMS_LOB.CLOSE (
   lob_loc    IN OUT NOCOPY BLOB);

DBMS_LOB.CLOSE (
   lob_loc    IN OUT NOCOPY CLOB CHARACTER SET ANY_CS);

DBMS_LOB.CLOSE (
   file_loc   IN OUT NOCOPY BFILE);
```

Pragmas

None.

Errors

No error is returned if the BFILE exists but is not opened. An error is returned if the LOB is not open.

Usage Requirements

CLOSE requires a round-trip to the server for both internal and external LOBs. For internal LOBs, CLOSE triggers other code that relies on the close call, and for external LOBs (BFILEs), CLOSE actually closes the server-side operating system file.

COMPARE function

This function compares two entire LOBs or parts of two LOBs. You can only compare LOBs of the same datatype (LOBs of BLOB type with other BLOBs, and CLOBs with CLOBs, and BFILEs with BFILEs). For BFILEs, the file must be already opened using a successful FILEOPEN operation for this operation to succeed.

COMPARE returns zero if the data exactly matches over the range specified by the offset and amount parameters. Otherwise, a non-zero INTEGER is returned.

For fixed-width *n*-byte CLOBs, if the input amount for COMPARE is specified to be greater than (4294967295 / *n*), then COMPARE matches characters in a range of size (4294967295 / *n*), or Max(length(clob1), length(clob2)), whichever is lesser.

Syntax

```
DBMS_LOB.COMPARE (
   lob_1            IN BLOB,
   lob_2            IN BLOB,
   amount           IN INTEGER := 4294967295,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.COMPARE (
   lob_1            IN CLOB  CHARACTER SET ANY_CS,
   lob_2            IN CLOB  CHARACTER SET lob_1%CHARSET,
   amount           IN INTEGER := 4294967295,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.COMPARE (
   lob_1            IN BFILE,
   lob_2            IN BFILE,
   amount           IN INTEGER,
   offset_1         IN INTEGER := 1,
   offset_2         IN INTEGER := 1)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(COMPARE, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 17–7 COMPARE Function Parameters

Parameter	Description
lob_1	LOB locator of first target for comparison.
lob_2	LOB locator of second target for comparison.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to compare.
offset_1	Offset in bytes or characters on the first LOB (origin: 1) for the comparison.
offset_2	Offset in bytes or characters on the first LOB (origin: 1) for the comparison.

Returns

- INTEGER: Zero if the comparison succeeds, non-zero if not.
- NULL, if
  - amount < 1
  - amount > LOBMAXSIZE
  - offset_1 or offset_2 < 1
    - offset_1 or offset_2 > LOBMAXSIZE

Exceptions

Table 17–8 COMPARE Function Exceptions for BFILE operations

Exception	Description
UNOPENED_FILE	File was not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Examples

```
CREATE OR REPLACE PROCEDURE Example2a IS
    lob_1, lob_2      BLOB;
    retval            INTEGER;
BEGIN
    SELECT b_col INTO lob_1 FROM lob_table
        WHERE key_value = 45;
    SELECT b_col INTO lob_2 FROM lob_table
        WHERE key_value = 54;
    retval := dbms_lob.compare(lob_1, lob_2, 5600, 33482,
         128);
    IF retval = 0 THEN
      ;    -- process compared code
    ELSE
      ;    -- process not compared code
    END IF;
END;

CREATE OR REPLACE PROCEDURE Example_2b IS
    fil_1, fil_2       BFILE;
    retval             INTEGER;
BEGIN

    SELECT f_lob INTO fil_1 FROM lob_table WHERE key_value = 45;
    SELECT f_lob INTO fil_2 FROM lob_table WHERE key_value = 54;
    dbms_lob.fileopen(fil_1, dbms_lob.file_readonly);
    dbms_lob.fileopen(fil_2, dbms_lob.file_readonly);
    retval := dbms_lob.compare(fil_1, fil_2, 5600,
                                  3348276, 2765612);
    IF (retval = 0)
    THEN
        ; -- process compared code
    ELSE
        ; -- process not compared code
    END IF;
    dbms_lob.fileclose(fil_1);
    dbms_lob.fileclose(fil_2);
END;
```

COPY procedure

This procedure copies all, or a part of, a source internal LOB to a destination internal LOB. You can specify the offsets for both the source and destination LOBs, and the number of bytes or characters to copy.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB or CLOB respectively. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

It is not an error to specify an amount that exceeds the length of the data in the source LOB. Thus, you can specify a large amount to copy from the source LOB, which copies data from the src_offset to the end of the source LOB.

Syntax

```
DBMS_LOB.COPY (
  dest_lob    IN OUT NOCOPY BLOB,
  src_lob     IN            BLOB,
  amount      IN            INTEGER,
  dest_offset IN            INTEGER := 1,
  src_offset  IN            INTEGER := 1);

DBMS_LOB.COPY (
  dest_lob    IN OUT NOCOPY CLOB  CHARACTER SET ANY_CS,
  src_lob     IN            CLOB  CHARACTER SET dest_lob%CHARSET,
  amount      IN            INTEGER,
  dest_offset IN            INTEGER := 1,
  src_offset  IN            INTEGER := 1);
```

Pragmas

None.

Parameters

Table 17–9 COPY Procedure Parameters

Parameter	Description
dest_lob	LOB locator of the copy target.
src_lob	LOB locator of source for the copy.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to copy.
dest_offset	Offset in bytes or characters in the destination LOB (origin: 1) for the start of the copy.
src_offset	Offset in bytes or characters in the source LOB (origin: 1) for the start of the copy.

Returns

None.

Exceptions

Table 17–10 COPY Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or invalid.
INVALID_ARGVAL	Either: - src_offset or dest_offset < 1 - src_offset or dest_offset > LOBMAXSIZE - amount < 1 - amount > LOBMAXSIZE

Examples

```
CREATE OR REPLACE PROCEDURE Example_3a IS
    lobd, lobs     BLOB;
    dest_offset    INTEGER := 1
    src_offset     INTEGER := 1
    amt            INTEGER := 3000;
BEGIN
    SELECT b_col INTO lobd
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    SELECT b_col INTO lobs
        FROM lob_table
        WHERE key_value = 21;
    DBMS_LOB.COPY(lobd, lobs, amt, dest_offset, src_offset);
    COMMIT;
   EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;

CREATE OR REPLACE PROCEDURE Example_3b IS
    lobd, lobs     BLOB;
    dest_offset    INTEGER := 1
    src_offset     INTEGER := 1
    amt            INTEGER := 3000;
BEGIN
```

```
    SELECT b_col INTO lobd
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    SELECT b_col INTO lobs
        FROM lob_table
        WHERE key_value = 12;
    DBMS_LOB.COPY(lobd, lobs, amt, dest_offset, src_offset);
    COMMIT;
   EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;
```

CREATETEMPORARY procedure

This procedure creates a temporary BLOB or CLOB and its corresponding index in your default temporary tablespace.

Syntax

```
DBMS_LOB.CREATETEMPORARY (
   lob_loc IN OUT NOCOPY BLOB,
   cache   IN            BOOLEAN,
   dur     IN            PLS_INTEGER := 10);

DBMS_LOB.CREATETEMPORARY (
   lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   cache   IN            BOOLEAN,
   dur     IN            PLS_INTEGER := 10);
```

Pragmas

None.

Parameters

Table 17–11 CREATETEMPORARY Procedure Parameters

Parameter	Description
lob_loc	LOB locator.
cache	Specifies if LOB should be read into buffer cache or not.

Table 17–11 CREATETEMPORARY Procedure Parameters

Parameter	Description
dur	1 of 2 predefined duration values (SESSION or CALL) which specifies whether the temporary LOB is cleaned up at the end of the session or call. If dur is omitted, then the session duration is used.

Returns

None.

Exceptions

None.

Example

```
DBMS_LOB.CREATETEMPORARY(Dest_Loc, TRUE, DBMS_LOB.SESSION)
```

ERASE procedure

This procedure erases an entire internal LOB or part of an internal LOB.

> **Note:** The length of the LOB is not decreased when a section of the LOB is erased.To decrease the length of the LOB value, see the "TRIM procedure" on page 17-52.

When data is erased from the middle of a LOB, zero-byte fillers or spaces are written for BLOBs or CLOBs respectively.

The actual number of bytes or characters erased can differ from the number you specified in the amount parameter if the end of the LOB value is reached before erasing the specified number. The actual number of characters or bytes erased is returned in the amount parameter.

Syntax

```
DBMS_LOB.ERASE (
   lob_loc          IN OUT   NOCOPY   BLOB,
   amount           IN OUT   NOCOPY   INTEGER,
   offset           IN                INTEGER := 1);
```

```
DBMS_LOB.ERASE (
   lob_loc          IN OUT   NOCOPY   CLOB CHARACTER SET ANY_CS,
   amount           IN OUT   NOCOPY   INTEGER,
   offset           IN                INTEGER := 1);
```

Pragmas

None.

Parameters

Table 17–12 ERASE Procedure Parameters

Parameter	Description
lob_loc	Locator for the LOB to be erased.
amount	Number of bytes (for BLOBs or BFILES) or characters (for CLOBs or NCLOBs) to be erased.
offset	Absolute offset (origin: 1) from the beginning of the LOB in bytes (for BLOBs) or characters (CLOBs).

Returns

None.

Exceptions

Table 17–13 ERASE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any input parameter is NULL.
INVALID_ARGVAL	Either: - amount < 1 or amount > LOBMAXSIZE - offset < 1 or offset > LOBMAXSIZE

Example

```
CREATE OR REPLACE PROCEDURE Example_4 IS
    lobd        BLOB;
    amt         INTEGER := 3000;
BEGIN
    SELECT b_col INTO lobd
```

```
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    dbms_lob.erase(dest_lob, amt, 2000);
    COMMIT;
END;
```

See Also: "TRIM procedure" on page 17-52

FILECLOSE procedure

This procedure closes a BFILE that has already been opened via the input locator.

Note: Oracle has only read-only access to BFILEs. This means that BFILEs cannot be written through Oracle.

Syntax

```
DBMS_LOB.FILECLOSE (
    file_loc IN OUT NOCOPY BFILE);
```

Pragmas

None.

Parameters

Table 17–14 FILECLOSE Procedure Parameter

Parameter	Description
file_loc	Locator for the BFILE to be closed.

Returns

None.

Exceptions

Table 17–15 FILECLOSE Procedure Exceptions

Exception	Description
VALUE_ERROR	NULL input value for file_loc.
UNOPENED_FILE	File was not opened with the input locator.

Table 17–15 FILECLOSE Procedure Exceptions

Exception	Description
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Example

```
CREATE OR REPLACE PROCEDURE Example_5 IS
    fil BFILE;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 99;
    dbms_lob.fileopen(fil);
    -- file operations
    dbms_lob.fileclose(fil);
    EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;
```

See Also: "FILEOPEN procedure" on page 17-32, "FILECLOSEALL procedure" on page 17-26

FILECLOSEALL procedure

This procedure closes all BFILEs opened in the session.

Syntax

```
DBMS_LOB.FILECLOSEALL;
```

Pragmas

None.

Returns

None.

Exceptions

Table 17–16 FILECLOSEALL Procedure Exception

Exception	Description
UNOPENED_FILE	No file has been opened in the session.

Example

```
CREATE OR REPLACE PROCEDURE Example_6 IS
    fil BFILE;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 99;
    dbms_lob.fileopen(fil);
    -- file operations
    dbms_lob.filecloseall;
    EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;
```

See Also: "FILEOPEN procedure" on page 17-32, "FILECLOSE procedure" on page 17-25

FILEEXISTS function

This function finds out if a given BFILE locator points to a file that actually exists on the server's filesystem.

Syntax

```
DBMS_LOB.FILEEXISTS (
   file_loc     IN    BFILE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(FILEEXISTS, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 17–17 FILEEXISTS Function Parameter

Parameter	Description
file_loc	Locator for the BFILE.

Returns

Table 17–18 FILEEXISTS Function Returns

Return	Description
0	Physical file does not exist.
1	Physical file exists.

Exceptions

Table 17–19 FILEEXISTS Function Exceptions

Exception	Description
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.

Example

```
CREATE OR REPLACE PROCEDURE Example_7 IS
    fil BFILE;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 12;
    IF (dbms_lob.fileexists(fil))
    THEN
        ; -- file exists code
    ELSE
        ; -- file does not exist code
    END IF;
    EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;
```

See Also: "FILEISOPEN function"

FILEGETNAME procedure

This procedure determines the directory alias and filename, given a BFILE locator. This function only indicates the directory alias name and filename assigned to the locator, not if the physical file or directory actually exists.

The maximum constraint values for the dir_alias buffer is 30, and for the entire pathname is 2000.

Syntax

```
DBMS_LOB.FILEGETNAME (
   file_loc   IN    BFILE,
   dir_alias  OUT   VARCHAR2,
   filename   OUT   VARCHAR2);
```

Pragmas

None.

Parameters

Table 17–20 FILEGETNAME Procedure Parameters

Parameter	Description
file_loc	Locator for the BFILE.
dir_alias	Directory alias.
filename	Name of the BFILE.

Returns

None.

Exceptions

Table 17–21 FILEGETNAME Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.

Table 17–21 FILEGETNAME Procedure Exceptions

Exception	Description
INVALID_ARGVAL	dir_alias or filename are NULL.

Example

```
CREATE OR REPLACE PROCEDURE Example_8 IS
    fil BFILE;
    dir_alias VARCHAR2(30);
    name VARCHAR2(2000);
BEGIN
    IF (dbms_lob.fileexists(fil))
    THEN
        dbms_lob.filegetname(fil, dir_alias, name);
        dbms_output.put_line("Opening " || dir_alias || name);
        dbms_lob.fileopen(fil, dbms_lob.file_readonly);
        -- file operations
        dbms_output.fileclose(fil);
    END IF;
END;
```

FILEISOPEN function

This function finds out whether a BFILE was opened with the given FILE locator.

If the input FILE locator was never passed to the FILEOPEN procedure, then the file is considered not to be opened by this locator. However, a different locator may have this file open. In other words, openness is associated with a specific locator.

Syntax

```
DBMS_LOB.FILEISOPEN (
   file_loc   IN    BFILE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(FILEISOPEN, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 17–22 FILEISOPEN Function Parameter

Parameter	Description
file_loc	Locator for the BFILE.

Returns

INTEGER: 0 = file is not open, 1 = file is open

Exceptions

Table 17–23 FILEISOPEN Function Exceptions

Exception	Description
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Example

```
CREATE OR REPLACE PROCEDURE Example_9 IS
DECLARE
    fil      BFILE;
    pos      INTEGER;
    pattern  VARCHAR2(20);
BEGIN
    SELECT f_lob INTO fil FROM lob_table
        WHERE key_value = 12;
    -- open the file
    IF (fileisopen(fil))
    THEN
        pos := dbms_lob.instr(fil, pattern, 1025, 6);
         -- more file operations
         dbms_lob.fileclose(fil);
   ELSE
        ; -- return error
    END IF;
END;
```

See Also: "FILEEXISTS function" on page 17-27

FILEOPEN procedure

This procedure opens a BFILE for read-only access. BFILEs may not be written through Oracle.

Syntax

```
DBMS_LOB.FILEOPEN (
   file_loc   IN OUT NOCOPY  BFILE,
   open_mode  IN             BINARY_INTEGER := file_readonly);
```

Pragmas

None.

Parameters

Table 17–24 FILEOPEN Procedure Parameters

Parameter	Description
file_loc	Locator for the BFILE.
open_mode	File access is read-only.

Returns

None.

Exceptions

Table 17–25 FILEOPEN Procedure Exceptions

Exception	Description
VALUE_ERROR	file_loc or open_mode is NULL.
INVALID_ARGVAL	open_mode is not equal to FILE_READONLY.
OPEN_TOOMANY	Number of open files in the session exceeds session_max_open_files.
NOEXIST_DIRECTORY	Directory associated with file_loc does not exist.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.

Table 17–25 FILEOPEN Procedure Exceptions

Exception	Description
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Example

```
CREATE OR REPLACE PROCEDURE Example_10 IS
    fil BFILE;
BEGIN
    -- open BFILE
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 99;
    IF (dbms_lob.fileexists(fil))
    THEN
        dbms_lob.fileopen(fil, dbms_lob.file_readonly);
        -- file operation
        dbms_lob.fileclose(fil);
    END IF;
    EXCEPTION
        WHEN some_exception
        THEN handle_exception;
END;
```

See Also: "FILECLOSE procedure" on page 17-25, "FILECLOSEALL procedure" on page 17-26

FREETEMPORARY procedure

This procedure frees the temporary BLOB or CLOB in your default temporary tablespace. After the call to FREETEMPORARY, the LOB locator that was freed is marked as invalid.

If an invalid LOB locator is assigned to another LOB locator using OCILobLocatorAssign in OCI or through an assignment operation in PL/SQL, then the target of the assignment is also freed and marked as invalid.

Syntax

```
DBMS_LOB.FREETEMPORARY (
   lob_loc  IN OUT  NOCOPY BLOB);

DBMS_LOB.FREETEMPORARY (
   lob_loc  IN OUT  NOCOPY CLOB CHARACTER SET ANY_CS);
```

Pragmas

None.

Parameters

Table 17–26 FREETEMPORARY Procedure Parameters

Parameter	Description
lob_loc	LOB locator.

Returns

None.

Exceptions

None.

Example

```
DECLARE
  a blob;
  b blob;
BEGIN
  dbms_lob.createtemporary(a, TRUE, dbms_lob.session);
  dbms_lob.createtemporary(b, TRUE, dbms_lob.session);
  ...
  -- the following call frees lob a
  dbms_lob.freetemporary(a);
  -- at this point lob locator a is marked as invalid
  -- the following assignment frees the lob b and marks it as invalid
also
  b := a;
END;
```

GETCHUNKSIZE function

When creating the table, you can specify the chunking factor, which can be a multiple of Oracle blocks. This corresponds to the chunk size used by the LOB data layer when accessing or modifying the LOB value. Part of the chunk is used to store system-related information, and the rest stores the LOB value.

This function returns the amount of space used in the LOB chunk to store the LOB value.

Syntax

```
DBMS_LOB.GETCHUNKSIZE (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.GETCHUNKSIZE (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(GETCHUNKSIZE, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 17–27 GETCHUNKSIZE Function Parameters

Parameter	Description
lob_loc	LOB locator.

Returns

The value returned for BLOBs is in terms of bytes. The value returned for CLOBs is in terms of characters.

Exceptions

None.

Usage Notes

Performance is improved if you enter read / write requests using a multiple of this chunk size. For writes, there is an added benefit, because LOB chunks are versioned, and if all writes are done on a chunk basis, then no extra or excess versioning is done or duplicated. You could batch up the WRITE until you have enough for a chunk, instead of issuing several WRITE calls for the same chunk.

GETLENGTH function

This function gets the length of the specified LOB. The length in bytes or characters is returned.

The length returned for a BFILE includes the EOF, if it exists. Any 0-byte or space filler in the LOB caused by previous ERASE or WRITE operations is also included in the length count. The length of an empty internal LOB is 0.

Syntax

```
DBMS_LOB.GETLENGTH (
   lob_loc    IN  BLOB)
  RETURN INTEGER;

DBMS_LOB.GETLENGTH (
   lob_loc    IN  CLOB   CHARACTER SET ANY_CS)
  RETURN INTEGER;

DBMS_LOB.GETLENGTH (
   lob_loc    IN  BFILE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(GETLENGTH, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 17–28 GETLENGTH Function Parameter

Parameter	Description
lob_loc	The locator for the LOB whose length is to be returned.

Returns

The length of the LOB in bytes or characters as an INTEGER. NULL is returned if the input LOB is NULL or if the input lob_loc is NULL. An error is returned in the following cases for BFILEs:

- lob_loc does not have the necessary directory and OS privileges
- lob_loc cannot be read because of an OS read error

Exceptions

None.

Examples

```
CREATE OR REPLACE PROCEDURE Example_11a IS
    lobd        BLOB;
    length      INTEGER;
BEGIN
    -- get the LOB locator
    SELECT b_lob INTO lobd FROM lob_table
        WHERE key_value = 42;
    length := dbms_lob.getlength(lob_loc);
    IF length IS NULL THEN
        dbms_output.put_line('LOB is null.');
    ELSE
        dbms_output.put_line('The length is '
            || length);
    END IF;
END;

CREATE OR REPLACE PROCEDURE Example_11b IS
DECLARE
    len INTEGER;
    fil BFILE;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 12;
    len := dbms_lob.length(fil);
END;
```

INSTR function

This function returns the matching position of the *nth* occurrence of the pattern in the LOB, starting from the offset you specify.

The form of the VARCHAR2 buffer (the `pattern` parameter) must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

For BFILEs, the file must be already opened using a successful FILEOPEN operation for this operation to succeed.

Operations that accept RAW or VARCHAR2 parameters for pattern matching, such as INSTR, do not support regular expressions or special matching characters (as in the case of SQL LIKE) in the pattern parameter or substrings.

Syntax

```
DBMS_LOB.INSTR (
   lob_loc    IN   BLOB,
   pattern    IN   RAW,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.INSTR (
   lob_loc    IN   CLOB      CHARACTER SET ANY_CS,
   pattern    IN   VARCHAR2  CHARACTER SET lob_loc%CHARSET,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;

DBMS_LOB.INSTR (
   lob_loc    IN   BFILE,
   pattern    IN   RAW,
   offset     IN   INTEGER := 1,
   nth        IN   INTEGER := 1)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(INSTR, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 17–29 INSTR Function Parameters

Parameter	Description
lob_loc	Locator for the LOB to be examined.
pattern	Pattern to be tested for. The pattern is a group of RAW bytes for BLOBs, and a character string (VARCHAR2) for CLOBs.The maximum size of the pattern is 16383 bytes.
offset	Absolute offset in bytes (BLOBs) or characters (CLOBs) at which the pattern matching is to start. (origin: 1)
nth	Occurrence number, starting at 1.

Returns

Table 17–30 INSTR Function Returns

Return	Description
INTEGER	Offset of the start of the matched pattern, in bytes or characters. It returns 0 if the pattern is not found.
NULL	Either: -any one or more of the IN parameters was NULL or INVALID. -offset < 1 or offset > LOBMAXSIZE. -nth < 1. -nth > LOBMAXSIZE.

Exceptions

Table 17–31 INSTR Function Exceptions for BFILES

Exception	Description
UNOPENED_FILE	File was not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Examples

```
CREATE OR REPLACE PROCEDURE Example_12a IS
    lobd        CLOB;
    pattern     VARCHAR2 := 'abcde';
    position    INTEGER  := 10000;
BEGIN
-- get the LOB locator
    SELECT b_col INTO lobd
        FROM lob_table
        WHERE key_value = 21;
    position := DBMS_LOB.INSTR(lobd,
                        pattern, 1025, 6);
    IF position = 0 THEN
        dbms_output.put_line('Pattern not found');
```

```
    ELSE
        dbms_output.put_line('The pattern occurs at '
                || position);
    END IF;
END;

CREATE OR REPLACE PROCEDURE Example_12b IS
DECLARE
    fil BFILE;
    pattern VARCHAR2;
    pos INTEGER;
BEGIN
    -- initialize pattern
    -- check for the 6th occurrence starting from 1025th byte
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 12;
    dbms_lob.fileopen(fil, dbms_lob.file_readonly);
    pos := dbms_lob.instr(fil, pattern, 1025, 6);
    dbms_lob.fileclose(fil);
END;
```

See Also: "SUBSTR function" on page 17-49

ISOPEN function

This function checks to see if the LOB was already opened using the input locator. This subprogram is for internal and external LOBs.

Syntax

```
DBMS_LOB.ISOPEN (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.ISOPEN (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;

DBMS_LOB.ISOPEN (
   file_loc IN BFILE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(ISOPEN, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 17–32 ISOPEN Function Parameters

Parameter	Description
lob_loc	LOB locator.
file_loc	File locator.

Exceptions

None.

Usage Notes

For BFILES, openness is associated with the locator. If the input locator was never passed to OPEN, then the BFILE is not considered to be opened by this locator. However, a different locator may have opened the BFILE. More than one OPEN can be performed on the same BFILE using different locators.

For internal LOBs, openness is associated with the LOB, not with the locator. If locator1 opened the LOB, then locator2 also sees the LOB as open. For internal LOBs, ISOPEN requires a round-trip, because it checks the state on the server to see if the LOB is indeed open.

For external LOBs (BFILEs), ISOPEN also requires a round-trip, because that's where the state is kept.

ISTEMPORARY function

Syntax

```
DBMS_LOB.ISTEMPORARY (
   lob_loc IN BLOB)
  RETURN INTEGER;

DBMS_LOB.ISTEMPORARY (
   lob_loc IN CLOB CHARACTER SET ANY_CS)
  RETURN INTEGER;
```

Pragmas

```
PRAGMA RESTRICT_REFERENCES(istemporary, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 17–33 ISTEMPORARY Procedure Parameters

Parameter	Description
lob_loc	LOB locator.
temporary	Boolean, which indicates whether the LOB is temporary or not.

Returns

This function returns TRUE in temporary if the locator is pointing to a temporary LOB. It returns FALSE otherwise.

Exceptions

None.

LOADFROMFILE procedure

This procedure copies all, or a part of, a source external LOB (BFILE) to a destination internal LOB.

You can specify the offsets for both the source and destination LOBs, and the number of bytes to copy from the source BFILE. The amount and src_offset, because they refer to the BFILE, are in terms of bytes, and the dest_offset is either in bytes or characters for BLOBs and CLOBs respectively.

> **Note:** The input BFILE must have been opened prior to using this procedure. No character set conversions are performed implicitly when binary BFILE data is loaded into a CLOB. The BFILE data must already be in the same character set as the CLOB in the database. No error checking is performed to verify this.

If the offset you specify in the destination LOB is beyond the end of the data currently in this LOB, then zero-byte fillers or spaces are inserted in the destination BLOB or CLOB respectively. If the offset is less than the current length of the destination LOB, then existing data is overwritten.

There is an error if the input amount plus offset exceeds the length of the data in the BFILE.

Syntax

```
DBMS_LOB.LOADFROMFILE (
   dest_lob    IN OUT NOCOPY BLOB,
   src_file    IN            BFILE,
   amount      IN            INTEGER,
   dest_offset IN            INTEGER  := 1,
   src_offset  IN            INTEGER  := 1);

DBMS_LOB.LOADFROMFILE(
   dest_lob    IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   src_file    IN            BFILE,
   amount      IN            INTEGER,
   dest_offset IN            INTEGER := 1,
   src_offset  IN            INTEGER := 1);
```

Pragmas

None.

Parameters

Table 17–34 LOADFROMFILE Procedure Parameters

Parameter	Description
dest_lob	LOB locator of the target for the load.
src_file	BFILE locator of the source for the load.
amount	Number of bytes to load from the BFILE.
dest_offset	Offset in bytes or characters in the destination LOB (origin: 1) for the start of the load.
src_offset	Offset in bytes in the source BFILE (origin: 1) for the start of the load.

Returns

None.

Exceptions

Table 17–35 LOADFROMFILE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of the input parameters are NULL or INVALID.

Table 17–35 LOADFROMFILE Procedure Exceptions

Exception	Description
INVALID_ARGVAL	Either: - src_offset or dest_offset < 1. - src_offset or dest_offset > LOBMAXSIZE. - amount < 1. - amount > LOBMAXSIZE.

Example

```
CREATE OR REPLACE PROCEDURE Example_l2f IS
  lobd       BLOB;
  fils       BFILE   := BFILENAME('SOME_DIR_OBJ','some_file');
  amt        INTEGER := 4000;
BEGIN
  SELECT b_lob INTO lobd FROM lob_table WHERE key_value = 42 FOR UPDATE;
  dbms_lob.fileopen(fils, dbms_lob.file_readonly);
  dbms_lob.loadfromfile(lobd, fils, amt);
  COMMIT;
  dbms_lob.fileclose(fils);
END;
```

OPEN procedure

This procedure opens a LOB, internal or external, in the indicated mode. Valid modes include read-only, and read-write. It is an error to open the same LOB twice.

Note: If the LOB was opened in read-only mode, and if you try to write to the LOB, then an error is returned. BFILE can only be opened with read-only mode.

In Oracle8.0, the constant file_readonly was the only valid mode in which to open a BFILE. For Oracle 8*i*, two new constants have been added to the DBMS_LOB package: lob_readonly and lob_readwrite.

Syntax

```
DBMS_LOB.OPEN (
   lob_loc   IN OUT NOCOPY BLOB,
   open_mode IN            BINARY_INTEGER);

DBMS_LOB.OPEN (
   lob_loc   IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   open_mode IN            BINARY_INTEGER);

DBMS_LOB.OPEN (
   file_loc  IN OUT NOCOPY BFILE,
   open_mode IN            BINARY_INTEGER := file_readonly);
```

Pragmas

None.

Parameters

Table 17–36 OPEN Procedure Parameters

Parameter	Description
lob_loc	LOB locator.
open_mode	Mode in which to open.

Usage Notes

OPEN requires a round-trip to the server for both internal and external LOBs. For internal LOBs, OPEN triggers other code that relies on the OPEN call. For external LOBs (BFILEs), OPEN requires a round-trip because the actual operating system file on the server side is being opened.

READ procedure

This procedure reads a piece of a LOB, and returns the specified amount into the buffer parameter, starting from an absolute offset from the beginning of the LOB.

The number of bytes or characters actually read is returned in the amount parameter. If the input offset points past the End of LOB, then amount is set to 0, and a NO_DATA_FOUND exception is raised.

Syntax

```
DBMS_LOB.READ (
   lob_loc   IN             BLOB,
   amount    IN OUT  NOCOPY BINARY_INTEGER,
   offset    IN             INTEGER,
   buffer    OUT            RAW);

DBMS_LOB.READ (
   lob_loc   IN             CLOB CHARACTER SET ANY_CS,
   amount    IN OUT  NOCOPY BINARY_INTEGER,
   offset    IN             INTEGER,
   buffer    OUT            VARCHAR2 CHARACTER SET lob_loc%CHARSET);

DBMS_LOB.READ (
   lob_loc   IN              BFILE,
   amount    IN OUT   NOCOPY BINARY_INTEGER,
   offset    IN              INTEGER,
   buffer    OUT             RAW);
```

Pragmas

None.

Parameters

Table 17–37 READ Procedure Parameters

Parameter	Description
lob_loc	Locator for the LOB to be read.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to read, or number that were read.
offset	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).
buffer	Output buffer for the read operation.

Returns

None.

Exceptions

Table 17–38 READ Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of lob_loc, amount, or offset parameters are NULL.
INVALID_ARGVAL	Either: - amount < 1 - amount > MAXBUFSIZE - offset < 1 - offset > LOBMAXSIZE - amount is greater, in bytes or characters, than the capacity of buffer.
NO_DATA_FOUND	End of the LOB is reached, and there are no more bytes or characters to read from the LOB: amount has a value of 0.

Exceptions for BFILEs

Table 17–39 READ Procedure Exceptions for BFILEs

Exception	Description
UNOPENED_FILE	File is not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Usage Notes

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.READ from the client (for example, in a BEGIN/END block from within SQL*Plus), the returned buffer contains data in the client's character set. Oracle converts the LOB value from the server's character set to the client's character set before it returns the buffer to the user.

Examples

```
CREATE OR REPLACE PROCEDURE Example_13a IS
    src_lob        BLOB;
    buffer         RAW(32767);
    amt            BINARY_INTEGER := 32767;
    pos            INTEGER := 2147483647;
BEGIN
    SELECT b_col INTO src_lob
        FROM lob_table
        WHERE key_value = 21;
    LOOP
        dbms_lob.read (src_lob, amt, pos, buffer);
        -- process the buffer
        pos := pos + amt;
    END LOOP;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            dbms_output.put_line('End of data');
END;

CREATE OR REPLACE PROCEDURE Example_13b IS
    fil BFILE;
    buf RAW(32767);
    amt BINARY_INTEGER := 32767;
    pos INTEGER := 2147483647;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 21;
    dbms_lob.fileopen(fil, dbms_lob.file_readonly);
    LOOP
        dbms_lob.read(fil, amt, pos, buf);
        -- process contents of buf
        pos := pos + amt;
    END LOOP;
    EXCEPTION
      WHEN NO_DATA_FOUND
      THEN
      BEGIN
        dbms_output.putline ('End of LOB value reached');
        dbms_lob.fileclose(fil);
      END;
END;
```

Example for efficient I/O on OS that performs better with block I/O rather than stream I/O:

```
CREATE OR REPLACE PROCEDURE Example_13c IS
    fil BFILE;
    amt BINARY_INTEGER := 1024; -- or n x 1024 for reading n
    buf RAW(1024); -- blocks at a time
    tmpamt BINARY_INTEGER;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 99;
    dbms_lob.fileopen(fil, dbms_lob.file_readonly);
    LOOP
        dbms_lob.read(fil, amt, pos, buf);
        -- process contents of buf
        pos := pos + amt;
    END LOOP;
    EXCEPTION
        WHEN NO_DATA_FOUND
        THEN
            BEGIN
                dbms_output.putline ('End of data reached');
                dbms_lob.fileclose(fil);
            END;
END;
```

SUBSTR function

This function returns `amount` bytes or characters of a `LOB`, starting from an absolute `offset` from the beginning of the `LOB`.

For fixed-width *n*-byte `CLOB`s, if the input amount for `SUBSTR` is specified to be greater than (32767 / *n*), then `SUBSTR` returns a character buffer of length (32767 / *n*), or the length of the `CLOB`, whichever is lesser.

Syntax

```
DBMS_LOB.SUBSTR (
   lob_loc     IN    BLOB,
   amount      IN    INTEGER := 32767,
   offset      IN    INTEGER := 1)
  RETURN RAW;

DBMS_LOB.SUBSTR (
   lob_loc     IN    CLOB   CHARACTER SET ANY_CS,
   amount      IN    INTEGER := 32767,
   offset      IN    INTEGER := 1)
  RETURN VARCHAR2 CHARACTER SET lob_loc%CHARSET;
```

```
DBMS_LOB.SUBSTR (
   lob_loc     IN    BFILE,
   amount      IN    INTEGER := 32767,
   offset      IN    INTEGER := 1)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(SUBSTR, WNDS, WNPS, RNDS, RNPS);
```

Parameters

Table 17–40 SUBSTR Function Parameters

Parameter	Description
lob_loc	Locator for the LOB to be read.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to be read.
offset	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1).

Returns

Table 17–41 SUBSTR Function Returns

Return	Description
RAW	Function overloading that has a BLOB or BFILE in parameter.
VARCHAR2	CLOB version.
NULL	Either: - any input parameter is NULL - amount < 1 - amount > 32767 - offset < 1 - offset > LOBMAXSIZE

Exceptions

Table 17–42 SUBSTR Function Exceptions for BFILE operations

Exception	Description
UNOPENED_FILE	File is not opened using the input locator.
NOEXIST_DIRECTORY	Directory does not exist.
NOPRIV_DIRECTORY	You do not have privileges for the directory.
INVALID_DIRECTORY	Directory has been invalidated after the file was opened.
INVALID_OPERATION	File does not exist, or you do not have access privileges on the file.

Usage Notes

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.SUBSTR from the client (for example, in a BEGIN/END block from within SQL*Plus), the returned buffer contains data in the client's character set. Oracle converts the LOB value from the server's character set to the client's character set before it returns the buffer to the user.

Examples

```
CREATE OR REPLACE PROCEDURE Example_14a IS
    src_lob        CLOB;
    pos            INTEGER := 2147483647;
    buf            VARCHAR2(32000);
BEGIN
    SELECT c_lob INTO src_lob FROM lob_table
        WHERE key_value = 21;
    buf := DBMS_LOB.SUBSTR(src_lob, 32767, pos);
    -- process the data
END;

CREATE OR REPLACE PROCEDURE Example_14b IS
    fil BFILE;
    pos INTEGER := 2147483647;
    pattern RAW;
BEGIN
    SELECT f_lob INTO fil FROM lob_table WHERE key_value = 21;
```

```
    dbms_lob.fileopen(fil, dbms_lob.file_readonly);
    pattern := dbms_lob.substr(fil, 255, pos);
    dbms_lob.fileclose(fil);
END;
```

See Also: "INSTR function" on page 17-37, "READ procedure" on page 17-45

TRIM procedure

This procedure trims the value of the internal LOB to the length you specify in the newlen parameter. Specify the length in bytes for BLOBs, and specify the length in characters for CLOBs.

Note: The TRIM procedure decreases the length of the LOB to the value specified in the newlen parameter.

If you attempt to TRIM an empty LOB, then nothing occurs, and TRIM returns no error. If the new length that you specify in newlen is greater than the size of the LOB, then an exception is raised.

Syntax

```
DBMS_LOB.TRIM (
   lob_loc         IN OUT  NOCOPY BLOB,
   newlen          IN              INTEGER);

DBMS_LOB.TRIM (
   lob_loc         IN OUT  NOCOPY CLOB CHARACTER SET ANY_CS,
   newlen          IN              INTEGER);
```

Pragmas

None.

Parameters

Table 17–43 TRIM Procedure Parameters

Parameter	Description
lob_loc	Locator for the internal LOB whose length is to be trimmed.

Table 17–43 TRIM Procedure Parameters

Parameter	Description
newlen	New, trimmed length of the LOB value in bytes for BLOBs or characters for CLOBs.

Returns

None.

Exceptions

Table 17–44 TRIM Procedure Exceptions

Exception	Description
VALUE_ERROR	lob_loc is NULL.
INVALID_ARGVAL	Either: - new_len < 0 - new_len > LOBMAXSIZE

Example

```
CREATE OR REPLACE PROCEDURE Example_15 IS
    lob_loc        BLOB;
BEGIN
-- get the LOB locator
    SELECT b_col INTO lob_loc
        FROM lob_table
        WHERE key_value = 42 FOR UPDATE;
    dbms_lob.trim(lob_loc, 4000);
    COMMIT;
END;
```

See Also: "ERASE procedure" on page 17-23, "WRITEAPPEND procedure" on page 17-56

WRITE procedure

This procedure writes a specified amount of data into an internal LOB, starting from an absolute offset from the beginning of the LOB. The data is written from the buffer parameter.

WRITE replaces (overwrites) any data that already exists in the LOB at the offset, for the length you specify.

There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer is written to the LOB. If the offset you specify is beyond the end of the data currently in the LOB, then zero-byte fillers or spaces are inserted in the BLOB or CLOB respectively.

Syntax

```
DBMS_LOB.WRITE (
   lob_loc  IN OUT NOCOPY  BLOB,
   amount   IN             BINARY_INTEGER,
   offset   IN             INTEGER,
   buffer   IN             RAW);

DBMS_LOB.WRITE (
   lob_loc  IN OUT  NOCOPY CLOB   CHARACTER SET ANY_CS,
   amount   IN             BINARY_INTEGER,
   offset   IN             INTEGER,
   buffer   IN             VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Pragmas

None.

Parameters

Table 17–45 WRITE Procedure Parameters

Parameter	Description
lob_loc	Locator for the internal LOB to be written to.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to write, or number that were written.
offset	Offset in bytes (for BLOBs) or characters (for CLOBs) from the start of the LOB (origin: 1) for the write operation.
buffer	Input buffer for the write.

Returns

None.

Exceptions

Table 17–46 WRITE Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of lob_loc, amount, or offset parameters are NULL, out of range, or INVALID.
INVALID_ARGVAL	Either: - amount < 1 - amount > MAXBUFSIZE - offset < 1 - offset > LOBMAXSIZE

Usage Notes

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.WRITE from the client (for example, in a BEGIN/END block from within SQL*Plus), the buffer must contain data in the client's character set. Oracle converts the client-side buffer to the server's character set before it writes the buffer data to the LOB.

Example

```
CREATE OR REPLACE PROCEDURE Example_16 IS
    lob_loc        BLOB;
    buffer         RAW;
    amt            BINARY_INTEGER := 32767;
    pos            INTEGER := 2147483647;
    i              INTEGER;
BEGIN
    SELECT b_col INTO lob_loc
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    FOR i IN 1..3 LOOP
        dbms_lob.write (lob_loc, amt, pos, buffer);
        -- fill in more data
        pos := pos + amt;
    END LOOP;
```

```
    EXCEPTION4
        WHEN some_exception
        THEN handle_exception;
END;
```

See Also: "APPEND procedure" on page 17-14, "COPY procedure" on page 17-19

WRITEAPPEND procedure

This procedure writes a specified amount of data to the end of an internal LOB. The data is written from the buffer parameter.

There is an error if the input amount is more than the data in the buffer. If the input amount is less than the data in the buffer, then only amount bytes or characters from the buffer are written to the end of the LOB.

Syntax

```
DBMS_LOB.WRITEAPPEND (
   lob_loc IN OUT NOCOPY BLOB,
   amount  IN            BINARY_INTEGER,
   buffer  IN            RAW);

DBMS_LOB.WRITEAPPEND (
   lob_loc IN OUT NOCOPY CLOB CHARACTER SET ANY_CS,
   amount  IN            BINARY_INTEGER,
   buffer  IN            VARCHAR2 CHARACTER SET lob_loc%CHARSET);
```

Pragmas

None.

Parameters

Table 17–47 WRITEAPPEND Procedure Parameters

Parameter	Description
lob_loc	Locator for the internal LOB to be written to.
amount	Number of bytes (for BLOBs) or characters (for CLOBs) to write, or number that were written.
buffer	Input buffer for the write.

Exceptions

Table 17–48 WRITEAPPEND Procedure Exceptions

Exception	Description
VALUE_ERROR	Any of lob_loc, amount, or offset parameters are NULL, out of range, or INVALID.
INVALID_ARGVAL	Either: - amount < 1 - amount > MAXBUFSIZE

Usage Notes

The form of the VARCHAR2 buffer must match the form of the CLOB parameter. In other words, if the input LOB parameter is of type NCLOB, then the buffer must contain NCHAR data. Conversely, if the input LOB parameter is of type CLOB, then the buffer must contain CHAR data.

When calling DBMS_LOB.WRITEAPPEND from the client (for example, in a BEGIN/END block from within SQL*Plus), the buffer must contain data in the client's character set. Oracle converts the client-side buffer to the server's character set before it writes the buffer data to the LOB.

Example

```
CREATE OR REPLACE PROCEDURE Example_17 IS
    lob_loc    BLOB;
    buffer     RAW;
    amt        BINARY_INTEGER := 32767;
    i          INTEGER;
BEGIN
    SELECT b_col INTO lob_loc
        FROM lob_table
        WHERE key_value = 12 FOR UPDATE;
    FOR i IN 1..3 LOOP
        -- fill the buffer with data to be written to the lob
        dbms_lob.writeappend (lob_loc, amt, buffer);
    END LOOP;
END;
```

See Also: "APPEND procedure" on page 17-14, "COPY procedure" on page 17-19, "WRITE procedure" on page 17-53

18

DBMS_LOCK

Oracle Lock Management services for your applications are available through procedures in the DBMS_LOCK package. You can request a lock of a specific mode, give it a unique name recognizable in another procedure in the same or another instance, change the lock mode, and release it.

Because a reserved user lock is the same as an Oracle lock, it has all the functionality of an Oracle lock, such as deadlock detection. Be certain that any user locks used in distributed transactions are released upon COMMIT, or an undetected deadlock may occur.

User locks never conflict with Oracle locks because they are identified with the prefix "UL". You can view these locks using the Enterprise Manager lock monitor screen or the appropriate fixed views. User locks are automatically released when a session terminates.

The lock identifier is a number in the range of 0 to 1073741823.

Some uses of user locks:

- Providing exclusive access to a device, such as a terminal
- Providing application-level enforcement of read locks
- Detecting when a lock is released and cleanup after the application
- Synchronizing applications and enforcing sequential processing

Requirements

DBMS_LOCK is most efficient with a limit of a few hundred locks per session. Oracle strongly recommends that you develop a standard convention for using these locks in order to avoid conflicts among procedures trying to use the same locks. For example, include your company name as part of your lock names.

Security

There might be operating system-specific limits on the maximum number of total locks available. This *must* be considered when using locks or making this package available to other users. Consider granting the EXECUTE privilege only to specific users or roles.

A better alternative would be to create a cover package limiting the number of locks used and grant EXECUTE privilege to specific users. An example of a cover package is documented in the DBMSLOCK.SQL package specification file.

Viewing and Monitoring Locks

Oracle provides two facilities to display locking information for ongoing transactions within an instance:

Enterprise Manager Monitors (Lock and Latch Monitors)	The Monitor feature of Enterprise Manager provides two monitors for displaying lock information of an instance. Refer to *Oracle Server Manager User's Guide* for complete information about the Enterprise Manager monitors.
UTLLOCKT.SQL	The UTLLOCKT.SQL script displays a simple character lock wait-for graph in tree structured fashion. Using any *ad hoc* SQL tool (such as SQL*Plus) to execute the script, it prints the sessions in the system that are waiting for locks and the corresponding blocking locks. The location of this script file is operating system dependent. (You must have run the CATBLOCK.SQL script before using UTLLOCKT.SQL.)

Constants

```
nl_mode  constant integer := 1;
ss_mode  constant integer := 2;       -- Also called 'Intended Share'
sx_mode  constant integer := 3;       -- Also called 'Intended Exclusive'
s_mode   constant integer := 4;
ssx_mode constant integer := 5;
x_mode   constant integer := 6;
```

These are the various lock modes (nl -> "NuLl", ss -> "Sub Shared", sx -> "Sub eXclusive", s -> "Shared", ssx -> "Shared Sub eXclusive", x -> "eXclusive").

A sub-share lock can be used on an aggregate object to indicate that share locks are being aquired on sub-parts of the object. Similarly, a sub-exclusive lock can be used on an aggregate object to indicate that exclusive locks are being aquired on sub-parts of the object. A share-sub-exclusive lock indicates that the entire aggregate object has a share lock, but some of the sub-parts may additionally have exclusive locks.

Lock Compatibility Rules When another process holds "held", an attempt to get "get" does the following:

Table 18–1 Lock Compatibility

HELD MODE	GET NL	GET SS	GET SX	GET S	GET SSX	GET X
NL	Success	Success	Success	Success	Success	Success
SS	Success	Success	Success	Success	Success	Fail
SX	Success	Success	Success	Fail	Fail	Fail
S	Success	Success	Fail	Success	Fail	Fail
SSX	Success	Success	Fail	Fail	Fail	Fail
X	Success	Fail	Fail	Fail	Fail	Fail

```
maxwait  constant integer := 32767;
```

The constant `maxwait` waits forever.

Summary of Subprograms

Table 18–2 DBMS_LOCK Package Subprograms

Subprogram	Description
ALLOCATE_UNIQUE procedure on page 18-4	Allocates a unique lock ID to a named lock.
REQUEST function on page 18-6	Requests a lock of a specific mode.
CONVERT function on page 18-7	Converts a lock from one mode to another.
RELEASE function on page 18-9	Releases a lock.
SLEEP procedure on page 18-10	Puts a procedure to sleep for a specific time.

ALLOCATE_UNIQUE procedure

This procedure allocates a unique lock identifier (in the range of 1073741824 to 1999999999) given a lock name. Lock identifiers are used to enable applications to coordinate their use of locks. This is provided because it may be easier for applications to coordinate their use of locks based on lock names rather than lock numbers.

If you choose to identify locks by name, you can use ALLOCATE_UNIQUE to generate a unique lock identification number for these named locks.

The first session to call ALLOCATE_UNIQUE with a new lock name causes a unique lock ID to be generated and stored in the dbms_lock_allocated table. Subsequent calls (usually by other sessions) return the lock ID previously generated.

A lock name is associated with the returned lock ID for at least expiration_secs (defaults to 10 days) past the last call to ALLOCATE_UNIQUE with the given lock name. After this time, the row in the dbms_lock_allocated table for this lock name may be deleted in order to recover space. ALLOCATE_UNIQUE performs a commit.

Caution: Named user locks may be less efficient, because Oracle uses SQL to determine the lock associated with a given name.

Syntax

```
DBMS_LOCK.ALLOCATE_UNIQUE (
   lockname         IN  VARCHAR2,
   lockhandle       OUT VARCHAR2,
   expiration_secs  IN  INTEGER   DEFAULT 864000);
```

Parameters

Table 18–3 ALLOCATE_UNIQUE Procedure Parameters

Parameter	Description
lockname	Name of the lock for which you want to generate a unique ID. Do not use lock names beginning with ORA$; these are reserved for products supplied by Oracle Corporation.
lockhandle	Returns the handle to the lock ID generated by ALLOCATE_UNIQUE. You can use this handle in subsequent calls to REQUEST, CONVERT, and RELEASE. A handle is returned instead of the actual lock ID to reduce the chance that a programming error accidentally creates an incorrect, but valid, lock ID. This provides better isolation between different applications that are using this package. LOCKHANDLE can be up to VARCHAR2 (128). All sessions using a lock handle returned by ALLOCATE_UNIQUE with the same lock name are referring to the same lock. Therefore, do not pass lock handles from one session to another.
expiration_specs	Number of seconds to wait after the last ALLOCATE_UNIQUE has been performed on a given lock, before permitting that lock to be deleted from the DBMS_LOCK_ALLOCATED table. The default waiting period is 10 days. You should not delete locks from this table. Subsequent calls to ALLOCATE_UNIQUE may delete expired locks to recover space.

Errors

ORA-20000, ORU-10003: Unable to find or insert lock <lockname> into catalog dbms_lock_allocated.

Exceptions

None.

REQUEST function

This function requests a lock with a given mode. REQUEST is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE_UNIQUE procedure.

Syntax

```
DBMS_LOCK.REQUEST(
   id                IN  INTEGER ||
   lockhandle        IN  VARCHAR2,
   lockmode          IN  INTEGER DEFAULT X_MODE,
   timeout           IN  INTEGER DEFAULT MAXWAIT,
   release_on_commit IN  BOOLEAN DEFAULT FALSE,
  RETURN INTEGER;
```

The current default values, such as X_MODE and MAXWAIT, are defined in the DBMS_LOCK package specification.

Parameters

Table 18–4 REQUEST Function Parameters

Parameter	Description
id or lockhandle	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change.
lockmode	Mode that you are requesting for the lock. The available modes and their associated integer identifiers are listed below. The abbreviations for these locks, as they appear in the V$ views and Enterprise Manager monitors are in parentheses. 1 - null mode 2 - row share mode (ULRS) 3 - row exclusive mode (ULRX) 4 - share mode (ULS) 5 - share row exclusive mode (ULRSX) 6 - exclusive mode (ULX) Each of these lock modes is explained in *Oracle8 Concepts*.

Table 18–4 REQUEST Function Parameters

Parameter	Description
timeout	Number of seconds to continue trying to grant the lock. If the lock cannot be granted within this time period, then the call returns a value of 1 (timeout).
release_on_commit	Set this parameter to TRUE to release the lock on commit or roll-back. Otherwise, the lock is held until it is explicitly released or until the end of the session.

Return Values

Table 18–5 REQUEST Function Return Values

Return Value	Description
0	Success
1	Timeout
2	Deadlock
3	Parameter error
4	Already own lock specified by id or lockhandle
5	Illegal lock handle

Exceptions

None.

CONVERT function

This function converts a lock from one mode to another. CONVERT is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE_UNIQUE procedure.

Syntax

```
DBMS_LOCK.CONVERT(
   id         IN INTEGER ||
   lockhandle IN VARCHAR2,
   lockmode   IN INTEGER,
   timeout    IN NUMBER DEFAULT MAXWAIT)
  RETURN INTEGER;
```

Parameters

Table 18–6 CONVERT Function Parameters

Parameter	Description
id or lockhandle	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change.
lockmode	New mode that you want to assign to the given lock. The available modes and their associated integer identifiers are listed below. The abbreviations for these locks, as they appear in the V$ views and Enterprise Manager monitors are in parentheses. 1 - null mode 2 - row share mode (ULRS) 3 - row exclusive mode (ULRX) 4 - share mode (ULS) 5 - share row exclusive mode (ULRSX) 6 - exclusive mode (ULX) Each of these lock modes is explained in *Oracle8 Concepts*.
timeout	Number of seconds to continue trying to change the lock mode. If the lock cannot be converted within this time period, then the call returns a value of 1 (timeout).

Return Values

Table 18–7 CONVERT Function Return Values

Return Value	Description
0	Success
1	Timeout
2	Deadlock
3	Parameter error
4	Don't own lock specified by `id` or `lockhandle`
5	Illegal lock handle

Exceptions

None.

RELEASE function

This function explicitly releases a lock previously acquired using the REQUEST function. Locks are automatically released at the end of a session. RELEASE is an overloaded function that accepts either a user-defined lock identifier, or the lock handle returned by the ALLOCATE_UNIQUE procedure.

Syntax

```
DBMS_LOCK.RELEASE (
   id         IN INTEGER)
  RETURN INTEGER;

DBMS_LOCK.RELEASE (
   lockhandle IN VARCHAR2)
  RETURN INTEGER;
```

Parameters

Table 18–8 RELEASE Function Parameter

Parameter	Description
id or lockhandle	User assigned lock identifier, from 0 to 1073741823, or the lock handle, returned by ALLOCATE_UNIQUE, of the lock mode you want to change.

Return Values

Table 18–9 RELEASE Function Return Values

Return Value	Description
0	Success
3	Parameter error
4	Do not own lock specified by id or lockhandle
5	Illegal lock handle

Exceptions

None.

SLEEP procedure

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

Syntax

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

Parameters

Table 18–10 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.

Example

This Pro*COBOL precompiler example shows how locks can be used to ensure that there are no conflicts when multiple people need to access a single device.

Print Check

Any cashier may issue a refund to a customer returning goods. Refunds under $50 are given in cash; anything above that is given by check. This code prints the check. The one printer is opened by all the cashiers to avoid the overhead of opening and closing it for every check. This means that lines of output from multiple cashiers could become interleaved if we don't ensure exclusive access to the printer. The DBMS_LOCK package is used to ensure exclusive access.

CHECK-PRINT

Get the lock "handle" for the printer lock:

```
MOVE "CHECKPRINT" TO LOCKNAME-ARR.
MOVE 10 TO LOCKNAME-LEN.
EXEC SQL EXECUTE
   BEGIN DBMS_LOCK.ALLOCATE_UNIQUE ( :LOCKNAME, :LOCKHANDLE );
   END; END-EXEC.
```

Lock the printer in exclusive mode (default mode):

```
EXEC SQL EXECUTE
   BEGIN DBMS_LOCK.REQUEST ( :LOCKHANDLE );
   END; END-EXEC.
```

We now have exclusive use of the printer, print the check:

```
...
```

Unlock the printer so other people can use it:

```
EXEC SQL EXECUTE
   BEGIN DBMS_LOCK.RELEASE ( :LOCKHANDLE );

   END; END-EXEC.
```

19

DBMS_LOGMNR

DBMS_LOGMNR supplies the log analyzer tool with the list of filenames and SCNs required to initialize the tool. After this procedure completes, the server is ready to process SELECTs against the V$LOGMNR_CONTENTS view.

Redo log data is especially important for recovery, because you can use it to pinpoint when a database became corrupted.You can then use this information to recover the database to the state just prior to corruption.

> **See Also:** *Oracle8i Administrator's Guide* and *Oracle8i Backup and Recovery Guide*

Using the LogMiner

After you have created a dictionary file with DBMS_LOGMNR_D, you can begin analyzing archived redo logs.

1. Start an Oracle instance, with the database either mounted or unmounted.
2. Specify the log file or files that you want to read by running the DBMS_LOGMNR.ADD_LOGFILE procedure. You can view information about specified log files with V$LOGMNR_FILES.
3. Start the log reader with the DBMS_LOGMNR.START_LOGMNR procedure. You can set the start and end SCN and time parameters in the START_LOGMNR command to filter the redo records you will analyze. You can set the V$LOGMNR_PARAMETERS view to view the parameters.
4. View the output through the V$LOGMNR_CONTENTS table. LogMiner returns all rows in SCN order, which is the same order applied in media recovery.

See Also: "Example" on page 19-9 and Chapter 19, "DBMS_LOGMNR"

Constants

Constants for ADD_LOGFILE Options flag

NEW	DBMS_LOGMNR.NEW purges the existing list of logfiles, if any. Place the logfile specified in the list of logfiles to be analyzed.
ADDFILE	DBMS_LOGMNR.ADDFILE adds this logfile to the list of logfiles to be analyzed. This only works if there was at least one invocation of ADD_LOGFILE with the Options parameter set to NEW.
REMOVEFILE	DBMS_LOGMNR.REMOVEFILE removes the logfile from the list of logfiles to be analyzed. This has no effect if the logfile was not previously added to the list.

Constants for START_LOGMNR Options flag

USER_COLMAP	DBMS_LOGMNR.USE_COLMAP uses the column map specified in the logmnr.opt file. This file must be in the same directory as the dictionary file specified by DictFileName.

Using Place Holder Columns

The V$LOGMNR_CONTENTS table includes multiple sets of place holder columns. Each place holder column set contains a name column, a redo value column, and an undo value column. Each place holder column set can be assigned to a table and column via an optional LogMiner assignment file (logmnr.opt). After a place holder column is assigned, it can be used to select changes to the assigned column and table from the redo log stream.

For example, the assignment "colmap = SCOTT EMP (1, EMPNO);" assigns the PH1 place holder column set to the table and column; SCOTT.EMP, column EMPNO. After being assigned, it is possible to select changes from the redo stream for the EMPNO column of the EMP table;

```
SELECT scn FROM V$LOGMNR_CONTENTS
WHERE ph1_name='EMPNO'
AND ph1_redo='12345';
```

The redo stream is processed, and any changes setting the EMPNO column of the EMP table to the value 12345 are returned.

It is possible to have multiple assignments for each place holder column set. For example:

```
colmap = SCOTT EMP (1, EMPNO);
```

followed by

```
colmap = ACCOUNTING CUSTOMER (1, CUSTID);
```

In this case, the PH1 place holder column set has two assignments: to select only changes to the EMP table, and to add the EMP table name to the SELECT;

```
SELECT scn FROM V$LOGMNR_CONTENTS
WHERE seg_name = 'EMP'
AND ph1_name='EMPNO'
AND ph1_redo='12345';
```

or

```
SELECT scn FROM V$LOGMNR_CONTENTS
WHERE seg_name = 'CUSTOMER'
AND ph1_name='CUSTID'
AND ph1_redo='12345';
```

Using the logmnr.opt Place Holder Column

The logmnr.opt file is processed when the DBMS_LOGMNR.START_LOGMNR procedure is performed and Options is set to USE_COLMAP (Options = USE_COLMAP). Setting USE_COLMAP in Options instructs the LogMiner to read and process the logmnr.opt file. The logmnr.opt file should be located in the same directory as the LogMiner dictionary file (UTL_FILE_DIR).

After the place holder column assignment file (logmnr.opt) is processed, all subsequent selects from the V$LOGMNR_CONTENTS table can use the assigned place holder columns. To change the assignments, update the logmnr.opt file, and re-start the LogMiner.

As the logmnr.opt file is processed the assigned columns are verified against the current LogMiner dictionary. If they do not exist, then the start fails.

Syntax Rules for logmnr.opt

```
line =  'colmap' <sp> '=' <sp> <schema> <sp> <table> <sp> '(' map ')' ';'
map =   <num> ',' <colname> [<num> ',' <colname>]
```

<sp> Space

Words in quotes are fixed symbols:

<num>	Any number (limited to the number of place holder column sets)
<table>	Name of the table
<schema>	Schema name
<colname>	Column name in the specified <schema>.<table>

You can repeat <num> ',' <colname> inside the parentheses up to the number of place holder columns in V$LOGMNR_CONTENTS table.

<table>, <schema> and <colname> *must* be in all uppercase.

Valid logmnr.opt Syntax

- The user wants some columns of either table to be in the place holder:

```
colmap = SCOTT EMP (1, EMPNO, 2 SAL, 3 JOB, 4 MGR, 5 COMM);
colmap = SCOTT DEPT (1, DEPTNO);
```

- The colmap can contain duplicate values. Only the first ph1_redo, ph1_undo and ph4_redo, ph4_undo get filled:

```
colmap = SCOTT EMP (1, EMPNO, 2, EMPNO, 3, EMPNO, 4, MGR);
```

Invalid logmnr.opt Syntax

- Syntax error: No space between "colmap" and "=":

```
colmap= SCOTT EMP (1, EMPNO, 2, SAL);
```

- Syntax error: map not specified correctly:

```
colmap = SCOTT EMP (1, EMPNO, 2);
```

- Syntax error: statement does not end in ';':

```
colmap = SCOTT EMP (1, EMPNO)
```

- Error: schema in lowercase:

```
colmap = scott EMP (1, EMPNO, 2 SAL);
```

- Error: tablename in lowercase:

```
colmap = SCOTT emp (1, EMPNO, 2 SAL);
```

- Error: more than 5 entries in map:

```
colmap = SCOTT EMP (1, EMPNO, 2 SAL, 3 JOB, 4 MGR, 5 COMM, 6 ENAME);
```

- Error: column REGION is not part of the table:

```
colmap = SCOTT EMP (1, EMPNO, 2 SAL, 3 JOB, 4 REGION);
```

Summary of Subprograms

Table 19–1 DBMS_LOGMNR Package Subprograms

Subprogram	Description
ADD_LOGFILE procedure on page 19-6	Adds a file to the existing or newly created list of archive files to process.
START_LOGMNR procedure on page 19-7	Initializes the log analyzer tool.
END_LOGMNR procedure on page 19-9	Finishes a session.

ADD_LOGFILE procedure

This procedure adds a file to the existing or newly created list of archive files to process.

In order to select information from the V$LOGMNR_CONTENTS view, the LogMiner session must be set up with some information. This procedure tells the LogMiner session the list of logfiles to analyze.

Note: If you want to analyze five logfiles, you must call the ADD_LOGFILE procedure five times.

Syntax

```
DBMS_LOGMNR.ADD_LOGFILE(
   LogFileName         IN VARCHAR2,
   Options             IN BINARY_INTEGER default ADDFILE );
```

Parameters

Table 19–2 ADD_LOGFILE Procedure Parameters

Parameter	Description
LogFileName	Name of the logfile that must be added to the list of logfiles to be analyzed by this session.
Options	Either: - Starts a new list (DBMS_LOGMNR.NEW) - Adds a file to an existing list (DBMS_LOGMNR.ADDFILE), or - Removes a logfile (DBMS_LOGMNR.REMOVEFILE) See "Constants for ADD_LOGFILE Options flag" on page 19-2.

START_LOGMNR procedure

This procedure starts a LogMiner session.

> **Note:** This procedure fails if you did not specify a list of logfiles to be analyzed previously through the ADD_LOGFILE procedure.

Syntax

```
DBMS_LOGMNR.START_LOGMNR(
   startScn            IN NUMBER default 0,
   endScn              IN NUMBER default 0,
   startTime           IN DATE default '01-jan-1988',
   endTime             IN DATE default '01-jan-2988',
   DictFileName        IN VARCHAR2 default '',
   Options             IN BINARY_INTEGER default 0 );
```

Parameters

Table 19–3 START_LOGMNR Procedure Parameters

Parameter	Description
startScn	Only consider redo records with SCN greater than or equal to the startSCN specified. This fails if there is no logfile with an SCN range (i.e, the LOW_SCN and NEXT_SCN associated with the logfile as shown in V$LOGMNR_LOGS view) containing the startScn.
endScn	Only consider redo records with SCN less than or equal to the endSCN specified. This fails if there is no logfile with an SCN range (i.e, the LOW_SCN and NEXT_SCN associated with the logfile as shown in V$LOGMNR_LOGS view) containing the endScn.
startTime	Only consider redo records with timestamp greater than or equal to the startTime specified. This fails if there is no logfile with a time range (i.e, the LOW_TIME and HIGH_TIME associated with the logfile as shown in V$LOGMNR_LOGS view) containing the startTime. This parameter is ignored if startScn is specified.
endTime	Only consider redo records with timestamp less than or equal to the endTime specified. This fails if there is no logfile with a time range (i.e, the LOW_TIME and HIGH_TIME associated with the logfile as shown in V$LOGMNR_LOGS view) containing the endTime. This parameter is ignored if endScn is specified.

Table 19–3 START_LOGMNR Procedure Parameters

Parameter	Description
DictFileName	This flat file contains a snapshot of the database catalog. This must be specified if you expect to see reconstructed SQL_REDO and SQL_UNDO columns in V$LOGMNR_CONTENTS, as well as fully translated SEG_NAME, SEG_OWNER, SEG_TYPE_NAME and TABLE_SPACE columns. The fully qualified pathname for the dictionary file must be specified (This file must have been created previously through the DBMS_LOGMNR_D.BUILD procedure).
Options	DBMS_LOGMNR.USE_COLMAP: Use the column map specified in logmnr.opt file. This file *must* be in the same directory as the dictionary file specified by DictFileName. See "Constants for START_LOGMNR Options flag" on page 19-2 and "Using the logmnr.opt Place Holder Column" on page 19-4.

Exceptions

The procedure fails with ORA-1280 for the following reasons:

1. No logfile has (LOW_SCN, NEXT_SCN) range containing the startScn specified.
2. No logfile has (LOW_SCN, NEXT_SCN) range containing the endScn specified.
3. No logfile has (LOW_TIME, HIGH_TIME) range containing the startTime specified.
4. No logfile has (LOW_TIME, HIGH_TIME) range containing the endTime specified.
5. The DictFileName does not exist.
6. The database generating the redo logs is different from the one that generated the dictionary file specified by DictFilename.
7. DBMS_LOGMNR.USE_COLMAP is set without a logmnr.opt file.
8. DBMS_LOGMNR.USE_COLMAP is set and there are syntax errors in logmnr.opt file.
9. endScn is less than startScn.
10. endTime is less than startTime (and startScn and endScn were not specified).

END_LOGMNR procedure

This procedure finishes a session.

Syntax

```
DBMS_LOGMNR.END_LOGMNR;
```

Parameters

None.

Example

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(
   LogFileName => '/oracle/logs/log1.f',
   Options => dbms_logmnr.NEW);

EXECUTE DBMS_LOGMNR.ADD_LOGFILE(
   LogFileName => '/oracle/logs/log2.f',
   Options => dbms_logmnr.ADDFILE);

EXECUTE DBMS_LOGMNR.START_LOGMNR(
   DictFileName =>'/oracle/dictionary.ora');

SELECT sql_redo
FROM V$LOGMNR_CONTENTS;
```

20

DBMS_LOGMNR_D

DBMS_LOGMNR_D contains the LogMnr procedure to create the LogMnr dictionary file. This procedure queries the dictionary tables of the current database and creates a text-based file containing their contents. The external dictionary file is created in preparation of future analysis of log files using the LogMnr tool.

> **See Also:** *Oracle8i Administrator's Guide* and *Oracle8i Backup and Recovery Guide*

Creating a Dictionary File

You create a dictionary file by mounting a database and then extracting dictionary information into an external file. You must create the dictionary file from the same database that generated the log files you want to analyze. After it is created, you can use the dictionary file to analyze log files.

1. Mount and then open the database whose files you want to analyze.
2. Run the PL/SQL procedure DBMS_LOGMNR_D.BUILD. This procedure creates the dictionary file, which you should use to analyze log files.

Summary of Subprograms

DBMS_LOGMNR_D contains one procedure: BUILD. This creates the dictionary file, which you should use to analyze log files.

BUILD procedure

This procedure queries the dictionary tables of the current database, and creates a text based file containing their contents.

Syntax

```
DBMS_LOGMNR_D.BUILD (
   dictionary_filename IN VARCHAR2,
   dictionary_location IN VARCHAR2);
```

Parameters

Table 20–1 BUILD Procedure Parameters

Parameter	Description
dictionary_filename	Name of the dictionary file.
dictionary_location	Path to file directory.

Usage Notes

The dictionary file should be created after all dictionary changes to a database and prior to the creation of any log files that are to be analyzed.

The BUILD procedure uses the UTL_FILE package which requires setting the UTIL_FILE_DIR parameter in init.ora.

SET SERVER OUTPUT ON to monitor progress of the dictionary build.

Some tables written to the dictionary file do not exist on pre-8*i* databases. In this case, one or more errors (942) may be issued during the dictionary build. This is expected behavior.

Example

This example creates a dictionary file as:

```
/oracle/database/l_dictionary.ora
```

```
SVRMGR> EXECUTE dbms_logmnr_d.build('l_dictionary.ora',
SVRMGR> '/oracle/database/');
```

See Also: Chapter 19, "DBMS_LOGMNR"

21

DBMS_OFFLINE_OG

The DBMS_OFFLINE_OG package contains public APIs for offline instantiation of master groups.

Note: These procedures are used in performing an offline instantiation of a master table in a multimaster replication environment.

These procedure should not be confused with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a snapshot) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Summary of Subprograms

Table 21–1 DBMS_OFFLINE_OG Package Subprograms

Subprogram	Description
BEGIN_INSTANTIATION procedure on page 21-2	Starts offline instantiation of a replicated object group.
BEGIN_LOAD procedure on page 21-3	Disables triggers while data is imported to new master site as part of offline instantiation.
END_INSTANTIATION procedure on page 21-4	Completes offline instantiation of a replicated object group.
END_LOAD procedure on page 21-5	Re-enables triggers after importing data to new master site as part of offline instantiation.
RESUME_SUBSET_OF_MASTERS procedure on page 21-6	Resumes replication activity at all existing sites except the new site during offline instantiation of a replicated object group.

BEGIN_INSTANTIATION procedure

This procedure starts offline instantiation of a replicated master group. You must call this procedure from the master definition site.

Syntax

```
DBMS_OFFLINE_OG.BEGIN_INSTANTIATION (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2,
   fname     IN   VARCHAR2);
```

Parameters

Table 21–2 BEGIN_INSTANTIATION Procedure Parameters

Parameter	Description
gname	Name of the object group that you want to replicate to the new site.
new_site	The fully qualified database name of the new site to which you want to replicate the object group.
fname	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 21–3 BEGIN_INSTANTIATION Procedure Exceptions

Exception	Description
badargument	NULL or empty string for object group or new master site name.
dbms_repcat.nonmasterdef	This procedure must be called from the master definition site.
sitealreadyexists	Given site is already a master site for this object group.
wrongstate	Status of master definition site must be QUIESCED.
dbms_repcat.missingrepgroup	gname does not exist as a replicated master group.
dbms_repcat.missing_flavor	If you receive this exception, contact Oracle Worldwide Support.

BEGIN_LOAD procedure

This procedure disables triggers while data is imported to new master site as part of offline instantiation. You must call this procedure from the new master site.

Syntax

```
DBMS_OFFLINE_OG.BEGIN_LOAD (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2);
```

Parameters

Table 21–4 BEGIN_LOAD Procedure Parameters

Parameter	Description
gname	Name of the object group whose members you are importing.
new_site	The fully qualified database name of the new site at which you will be importing the object group members.

Exceptions

Table 21–5 BEGIN_LOAD Procedure Exceptions

Exception	Description
badargument	Null or empty string for object group or new master site name.
wrongsite	This procedure must be called from the new master site.
unknownsite	Given site is not recognized by object group.
wrongstate	Status of the new master site must be QUIESCED.
dbms_repcat.missingrepgroup	gname does not exist as a replicated master group.

END_INSTANTIATION procedure

This procedure completes offline instantiation of a replicated master group. You must call this procedure from the master definition site.

Syntax

```
DBMS_OFFLINE_OG.END_INSTANTIATION (
   gname     IN  VARCHAR2,
   new_site  IN  VARCHAR2);
```

Parameters

Table 21–6 END_INSTANTIATION Procedure Parameters

Parameter	Description
gname	Name of the object group that you are replicating to the new site.
new_site	The fully qualified database name of the new site to which you are replicating the object group.

Exceptions

Table 21–7 END_INSTANTIATION Procedure Exceptions

Exception	Description
badargument	Null or empty string for object group or new master site name.
dbms_repcat.nonmasterdef	This procedure must be called from the master definition site.
unknownsite	Given site is not recognized by object group.
wrongstate	Status of master definition site must be QUIESCED.
dbms_repcat.missingrepgroup	gname does not exist as a replicated master group.

END_LOAD procedure

This procedure re-enables triggers after importing data to new master site as part of offline instantiation. You must call this procedure from the new master site.

Syntax

```
DBMS_OFFLINE_OG.END_LOAD (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2,
   fname     IN   VARCHAR2);
```

Parameters

Table 21–8 END_LOAD Procedure Parameters

Parameter	Description
gname	Name of the object group whose members you have finished importing.
new_site	The fully qualified database name of the new site at which you have imported the object group members.
fname	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 21–9 END_LOAD Procedure Exceptions

Exception	Description
badargument	NULL or empty string for object group or new master site name.
wrongsite	This procedure must be called from the new master site.
unknownsite	Given site is not recognized by object group.
wrongstate	Status of the new master site must be QUIESCED.
dbms_repcat.missingrepgroup	gname does not exist as a replicated master group.
dbms_repcat.flavor_noobject	If you receive this exception, contact Oracle Worldwide Support.
dbms_repcat.flavor_contains	If you receive this exception, contact Oracle Worldwide Support.

RESUME_SUBSET_OF_MASTERS procedure

This procedure resumes replication activity at all existing sites except the new site during offline instantiation of a replicated master group. You must call this procedure from the master definition site.

Syntax

```
DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS (
   gname     IN  VARCHAR2,
   new_site  IN  VARCHAR2,
   override  IN  BOOLEAN := FALSE);
```

Parameters

Table 21–10 RESUME_SUBSET_OF_MASTERS Procedure Parameters

Parameter	Description
gname	Name of the object group that you are replicating to the new site.
new_site	The fully qualified database name of the new site to which you are replicating the object group.
override	If this is TRUE, then it ignores any pending RepCat administration requests and restores normal replication activity at each master as quickly as possible. This should be considered only in emergency situations. If this is FALSE, then it restores normal replication activity at each master only when there is no pending RepCat administration request for gname at that master.

Exceptions

Table 21–11 RESUME_SUBSET_OF_MASTERS Procedure Exceptions

Exception	Description
badargument	NULL or empty string for object group or new master site name.
dbms_repcat. nonmasterdef	This procedure must be called from the master definition site.
unknownsite	Given site is not recognized by object group.
wrongstate	Status of master definition site must be QUIESCED.
dbms_repcat. missingrepgroup	gname does not exist as a replicated master group.

22

DBMS_OFFLINE_SNAPSHOT

The DBMS_OFFLINE_SNAPSHOT package contains public APIs for offline instantiation of snapshots.

Note: These procedure are used in performing an offline instantiation of a snapshot.

These procedures should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Summary of Subprograms

Table 22–1 DBMS_OFFLINE_SNAPSHOT Package Subprograms

Subprogram	Description
BEGIN_LOAD procedure on page 22-2	Prepares a snapshot site for import of a new snapshot as part of offline instantiation.
END_LOAD procedure on page 22-3	Completes offline instantiation of a snapshot.

BEGIN_LOAD procedure

This procedure prepares a snapshot site for import of a new snapshot as part of offline instantiation. You must call this procedure from the snapshot site for the new snapshot.

Syntax

```
DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
   gname               IN   VARCHAR2,
   sname               IN   VARCHAR2,
   master_site         IN   VARCHAR2,
   snapshot_oname      IN   VARCHAR2,
   storage_c           IN   VARCHAR2 := '',
   comment             IN   VARCHAR2 := '',
   min_communication   IN   BOOLEAN  := TRUE);
```

Parameters

Table 22–2 BEGIN_LOAD Procedure Parameters

Parameter	Description
gname	Name of the object group for the snapshot that you are creating using offline instantiation.
sname	Name of the schema for the new snapshot.
master_site	Fully qualified database name of the snapshot's master site.
snapshot_oname	Name of the temporary snapshot created at the master site.
storage_c	Storage options to use when creating the new snapshot at the snapshot site.
comment	User comment.

Table 22–2 BEGIN_LOAD Procedure Parameters

Parameter	Description
min_communication	If TRUE, then the update trigger sends the new value of a column only if the update statement modifies the column. The update trigger sends the old value of the column only if it is a key column or a column in a modified column group.

Exceptions

Table 22–3 BEGIN_LOAD Procedure Exceptions

Exception	Description
badargument	Null or empty string for object group, schema, master site, or snapshot name.
dbms_repcat. missingrepgroup	gname does not exist as a replicated master group.
missingremotesnap	Could not locate given snapshot at given master site.
dbms_repcat. missingschema	Given schema does not exist.
snaptabmismatch	Base table name of the snapshot at the master and snapshot do not match.

END_LOAD procedure

This procedure completes offline instantiation of a snapshot. You must call this procedure from the snapshot site for the new snapshot.

Syntax

```
DBMS_OFFLINE_SNAPSHOT.END_LOAD (
   gname           IN  VARCHAR2,
   sname           IN  VARCHAR2,
   snapshot_oname  IN  VARCHAR2);
```

Parameters

Table 22–4 END_LOAD Procedure Parameters

Parameter	Description
gname	Name of the object group for the snapshot that you are creating using offline instantiation.
sname	Name of the schema for the new snapshot.
snapshot_oname	Name of the snapshot.

Exceptions

Table 22–5 END_LOAD Procedure Exceptions

Exception	Description
badargument	NULL or empty string for object group, schema, or snapshot name.
dbms_repcat.missingrepgroup	gname does not exist as a replicated master group.
dbms_repcat.nonsnapshot	This procedure must be called from the snapshot site.

23

DBMS_OLAP

The DBMS_OLAP package provides a collection of materialized view analysis and advisory functions that are callable from any PL/SQL program.

> **See Also:** For guidelines on using these functions, please see the *Oracle8i Tuning* manual.

Requirements

DBMS_OLAP utilizes structural statistics (cardinalities of fact tables, dimension tables, and distinct cardinalities of dimension level columns, join key columns, and fact table key columns) and optionally workload statistics on materialized view management events collected by Oracle Trace. The workload tables generated by Oracle Trace, if any, must be stored in the current default schema.

Error Messages

Table 23–1 DBMS_OLAP Error Messages

Error Code	Description
ORA-30476	PLAN_TABLE does not exist in the user's schema
ORA-30477	The input select_clause is incorrectly specified
ORA-30478	Specified dimension does not exist
ORA-30479	Summary Advisor error\n <QSM message with more details
QSM-00501	Unable to initialize Summary Advisor environment
QSM-00502	OCI error
QSM-00503	Out of memory
QSM-00504	Internal error
QSM-00505	Syntax error in <parse_entity_name> - <error_description>
QSM-00506	No fact-tables could be found
QSM-00507	No dimensions could be found
QSM-00508	Statistics missing on tables/columns
QSM-00509	Invalid parameter - <parameter_name>
QSM-00510	Statistics missing on summaries
QSM-00511	Invalid fact-tables specified in fact-filter
QSM-00512	Invalid summaries specified in the retention-list
QSM-00513	One or more of the workload tables is missing

Summary of Subprograms

Table 23–2 DBMS_OLAP Package Subprograms

Subprogram	Description
ESTIMATE_SUMMARY_SIZE procedure on page 23-3	Estimates the size of a materialized view that you might create, in bytes and rows.
EVALUATE_UTILIZATION procedure on page 23-4	Measures the utilization of each existing materialized view.
EVALUATE_UTILIZATION_W procedure on page 23-4	Measures the utilization of each existing materialized view.
RECOMMEND_MV procedure on page 23-5	Generates a set of recommendations about which materialized views should be created, retained, or dropped.
RECOMMEND_MV_W procedure on page 23-7	Generates a set of recommendations about which materialized views should be created, retained, or dropped.
VALIDATE_DIMENSION procedure on page 23-9	Verifies that the relationships specified in a `dimension` are correct.

ESTIMATE_SUMMARY_SIZE procedure

This estimates the size of a materialized view that you might create, in bytes and rows.

Syntax

```
DBMS_OLAP.ESTIMATE_SUMMARY_SIZE (
   stmt_id       IN  VARCHAR2,
   select_clause IN  VARCHAR2,
   num_rows      OUT NUMBER,
   num_bytes     OUT NUMBER);
```

Parameters

Table 23–3 ESTIMATE_SIZE Procedure Parameters

Parameter	Description
stmt_id	Arbitrary string used to identify the statement in an EXPLAIN PLAN.

Table 23–3 ESTIMATE_SIZE Procedure Parameters

Parameter	Description
select_clause	The SELECT statement to be analyzed.
num_rows	Estimated cardinality.
num_bytes	Estimated number of bytes.

EVALUATE_UTILIZATION procedure

This procedure measures the utilization of each existing materialized view based on the materialized view usage statistics from a hypothetical workload. It requires parameters that are *not* explicit.

The output is contained in the MVIEW$_EVALUATIONS table, which is initially truncated if it already contains rows.

Syntax

```
DBMS_OLAP.EVALUATE_UTILIZATION;
```

Parameters

Table 23–4 EVALUATE_UTILIZATION Procedure Parameters

Parameter	Description
MVIEW$_EVALUATIONS	Describes an evaluation of the utilization of each existing materialized view. It is initially truncated if it already contains rows. This is implicit, because it is supplied to the procedure when the procedure is called.

EVALUATE_UTILIZATION_W procedure

This procedure measures the utilization of each existing materialized view based on the materialized view usage statistics collected from the workload. The workload must be contained in tables located in the default schema, as described below.

This procedure also creates the WORK$_IDEAL_MVIEW and WORK$_MVIEW_USAGE views.

See Also: "DBMS_OLAP Interface Tables" on page 23-10

Syntax

```
DBMS_OLAP.EVALUATE_UTILIZATION_W;
```

Parameters

Table 23–5 EVALUATE_UTILIZATION_W Procedure Parameters

Parameter	I/O	Description
MVIEW$_EVALUATIONS	OUT	Returns an evaluation of the utilization of each existing materialized view. It is initially truncated if it already contains rows. This is implicit, because it is supplied to the procedure when the procedure is called.
V_192216243_F_5_E_14_8_1	IN	Table of workload requests logged by Oracle Trace. This is implicit, because it is supplied to the procedure when the procedure is called.
V_192216243_F_5_E_15_8_1	IN	Table of materialized view usages logged by Oracle Trace. This is implicit, because it is supplied to the procedure when the procedure is called.

RECOMMEND_MV procedure

This procedure generates a set of recommendations about which materialized views should be created, retained, or dropped, based on an analysis of table and column cardinality statistics gathered by ANALYZE.

The recommendations are based on a hypothetical workload in which all possible queries in the data warehouse are weighted equally. This procedure does not require or use the workload statistics tables collected by Oracle Trace, but it works even if those tables are present.

Dimensions must have been created, and there must be foreign key constraints that link the dimensions to fact tables.

Recommending materialized views with a hypothetical workload is appropriate in a DBA-less environment where ease of use is the primary consideration; however, if a workload is available in the default schema, it should be used.

See Also: For workload-driven analysis, see "RECOMMEND_MV_W procedure" on page 23-7

Syntax

```
DBMS_OLAP.RECOMMEND_MV (
   fact_table_filter IN VARCHAR2,
   storage_in_bytes  IN NUMBER,
   retention_list    IN VARCHAR2,
   retention_pct     IN NUMBER := 50);
```

Parameters

Table 23–6 RECOMMEND_MV Procedure Parameters

Parameter	Description
fact_table_filter	Comma-separated list of fact table names to analyze, or NULL to analyze all fact tables.
storage_in_bytes	Maximum storage, in bytes, that can be used for storing materialized views. This number must be non-negative.
retention_list	Comma-separated list of materialized view table names. A drop recommendation is not made for any materialized view that appears in this list.
retention_pct	Number between 0 and 100 that specifies the percent of existing materialized view storage that must be retained, based on utilization on the actual or hypothetical workload. A materialized view is retained if the cumulative space, ranked by utilization, is within the retention threshold specified (or if it is explicitly listed in retention_list). Materialized views that have a NULL utilization (e.g., non-dimensional materialized views) are always retained.

Parameters

Table 23–7 RECOMMEND_MV Procedure Parameters

Parameter	Description
MVIEW$_RECOMMENDATION	Returns the recommendations made, including a size estimate and the SQL required to build the materialized view. This is implicit, because it is supplied to the procedure when the procedure is called.

RECOMMEND_MV_W procedure

This procedure generates a set of recommendations about which materialized views should be created, retained, or dropped, based on information in the workload (gathered by Oracle Trace), and an analysis of table and column cardinality statistics gathered by ANALYZE.

RECOMMEND_MV_W requires that you have run ANALYZE to gather table and column cardinality statistics, you have collected and formatted the workload statistics, and you have created dimensions.

The workload is aggregated to determine the count of each request in the workload, and this count is used as a weighting factor during the optimization process.

The space of all dimensional materialized views that include the specified fact tables identifies the set of materialized views that optimize performance across the workload.

This procedure also creates the WORK$_IDEAL_MVIEW and WORK$_MVIEW_USAGE views.

See Also: "DBMS_OLAP Interface Tables" on page 23-10

Syntax

```
DBMS_OLAP.RECOMMEND_MV_W (
   fact_table_filter IN  VARCHAR2,
   storage_in_bytes  IN  NUMBER,
   retention_list    IN  VARCHAR2,
   retention_pct     IN  NUMBER := 80);
```

Parameters

Table 23–8 RECOMMEND_MV_W Procedure Parameters

Parameter	Description
fact_table_filter	Comma-separated list fact table names to analyze, or NULL to analyze all fact tables.
storage_in_bytes	Maximum storage, in bytes, that can be used for storing materialized views. This number must be non-negative.
retention_list	Comma-separated list of materialized view table names. A drop recommendation is not made for any materialized view that appears in this list.
retention_pct	Number between 0 and 100 that specifies the percent of existing materialized view storage that must be retained, based on utilization on the actual or hypothetical workload. A materialized view is retained if the cumulative space, ranked by utilization, is within the retention threshold specified (or if it is explicitly listed in retention_list). Materialized views that have a NULL utilization (e.g. non-dimensional materialized views) are always retained.

Parameters

Table 23–9 RECOMMEND_MV_W Procedure Parameters

Parameter	I/O	Description
MVIEW$_RECOMMENDATION	OUT	Returns the recommendations made, including a size estimate and the SQL required to build the materialized view. This is implicit, because it is supplied to the procedure when the procedure is called.
V_192216243_F_5_E_14_8_1 (required)	IN	Table of workload requests logged by Oracle Trace. This is implicit, because it is supplied to the procedure when the procedure is called.
V_192216243_F_5_E_15_8_1 (required)	IN	Table of materialized view usages logged by Oracle Trace. This is implicit, because it is supplied to the procedure when the procedure is called.

VALIDATE_DIMENSION procedure

This procedure verifies that the hierarchical and attribute relationships, and join relationships, specified in an existing dimension object are correct. This provides a fast way to ensure that referential integrity is maintained.

Syntax

```
DBMS_OLAP.VALIDATE_DIMENSION (
   dimension_name  VARCHAR2,
   incremental     BOOLEAN := TRUE,
   check_nulls     BOOLEAN := FALSE);
```

Parameters

Table 23–10 VALIDATE_DIMENSION Procedure Parameters

Parameter	Description
dimension_name	Name of the dimension to analyze.
incremental	If TRUE, then tests are performed only for the rows specified in the sumdelta$ table for tables of this dimension; otherwise, check all rows.
check_nulls	If TRUE, then all level columns are verified to be non-NULL; otherwise, this check is omitted. Specify FALSE when non-nullness is guaranteed by other means, such as NOT NULL constraints.

Parameters

Table 23–11 VALIDATE_DIMENSION Procedure Parameters

Parameter	I/O	Description
MVIEW$_EXCEPTIONS	OUT	Returns the rows of the tables referenced in the named dimension that violate dimensional integrity. Each row identifies a table and a ROWID for an exception. This is implicit, because it is supplied to the procedure when the procedure is called.

DBMS_OLAP Interface Tables

MVIEW$_RECOMMENDATION

This table represents the recommendations made by the RECOMMEND_MV procedure or the RECOMMEND_MV_W procedure.

Table 23–12 MVIEW$_RECOMMENDATION

Column Name	Type	Constraints	Description
recommendation_number	INTEGER	Primary key > 0	Unique identifier for this recommendation.
recommended_action	VARCHAR2(6)	CREATE, RETAIN, or DROP	What to do with the materialized view described by this row.
summary_owner	VARCHAR(30)	Undefined when action is CREATE	Owner of an existing materialized view to DROP or RETAIN.
summary_name	VARCHAR2(30)	Undefined when action is CREATE	Name of an existing materialized view to DROP or RETAIN.
group_by_columns	VARCHAR2(2000)	NULL unless action is CREATE	Comma-separated list of column references in the GROUP BY clause of the materialized view. These columns references also appear in the SELECT list
where_clause	VARCHAR2(2000)	NULL unless action is CREATE	AND-separated list of inner equijoins, as they would appear in the WHERE clause of this materialized view.
from_clause	VARCHAR2(2000)	NULL unless action is CREATE	Comma-separated list of relation names.

Table 23–12 MVIEW$_RECOMMENDATION

Column Name	Type	Constraints	Description
measures_list	VARCHAR2(2000)	NULL if action is DROP, non-NULL if action is CREATE	Comma separated list of <groupingFunction> (<expression>), as it would appear in the SELECT list of the materialized view. If recommended_action is CREATE, this field is the list of measures which must be present in the created materialized view. If recommended_action is RETAIN and this field is non-NULL, this field is the list of measures which must be added to the materialized view.
storage_in_bytes	NUMBER	>= 0	Actual or estimated storage in bytes.
pct_performance_gain	NUMBER		The expected incremental improvement in performance obtained by accepting this recommendation relative to the initial condition, assuming that all previous recommendations have been accepted, or NULL if unknown.
benefit_to_cost_ratio	NUMBER		Ratio of the incremental improvement in performance to the size of the materialized view in bytes, or NULL if unknown.

Each row contains a recommended action (CREATE, RETAIN, or DROP), and a description of the materialized view in one of two forms:

- The materialized view name, if the materialized view already exists (when the recommended action is RETAIN or DROP), or
- The SQL SELECT expression for creating the materialized view (when the recommended action is CREATE). The SQL SELECT expression is supplied in four parts:

1. A comma-separated list of column names in the GROUP BY clause of the materialized view. These columns also appear in the SELECT list.
2. The contents of the WHERE clause, which is an AND-separated list of inner equijoins
3. The contents of the FROM clause, which is a comma-separated list of relation names
4. The list of aggregated measures, which is a comma separated list of <grouping function>(<expression>), as it appears in the SELECT list of the materialized view.

This is an example of how you can assemble the SQL SELECT expression:

```
SELECT <group by columns>, <measures list>
   FROM <from clause>
   WHERE <where clause>
   GROUP BY <group by columns>;
```

Each row also contains the storage_in_bytes field, which is the storage in bytes that an existing materialized view occupies; in the case of newly recommended materialized views, this is an estimate of the size that the materialized view would occupy.

Two performance metrics are provided for each materialized view:

pct_performance_gain	The expected incremental improvement in performance obtained by accepting this recommendation, assuming that all previous recommendations have been accepted.
benefit_to_cost_ratio	The ratio of the incremental improvement in performance to the size of the materialized view.

The recommendations are ordered from most beneficial to least beneficial by recommendation_number, and the incremental benefit is calculated under the assumption that all previous recommendations in the list have been accepted.

MVIEW$_EVALUATIONS

This table represents the evaluations made by the EVALUATE_UTILIZATION procedure. The number of rows in this table is equal to the number of materialized views in the current database.

Table 23–13 MVIEW$_EVALUATIONS

Column Name	Type	Constraints	Description
summary_owner	VARCHAR2(30)	Primary key	Owner of an existing materialized view in this database.
summary_name	VARCHAR2(30)	Primary key	Name of an exiting materialized view in this database.
rank	INTEGER	>= 1	Rank of this materialized view in descending order of benefit_to_cost_ratio.
storage_in_bytes	NUMBER	>= 0	Size of the materialized view in bytes.
frequency	INTEGER	>= 0, or NULL	Number of times this materialized view appears in the workload.
cumulative_benefit	NUMBER	>= 0, or NULL	Each time a materialized view is used by query rewrite, the materialized view and its benefit is logged to the workload. This field sums up the benefit (or net reduction factor) for each materialized view
benefit_to_cost_ratio	NUMBER		If storage_in_bytes > 0, then the ratio of cumulative_benefit to storage_in_bytes, else NULL.

Note: The benefit_to_cost_ratio in this table uses a similar, but not identical, method as what is used to compute the benefit_to_cost_ratio of the MVIEW$_RECOMMENDATION table.

WORK$_IDEAL_MVIEW

This view is based on the table V_192216243_F_5_E_14_8_1 which corresponds to an actual or potential query rewrite.

Table 23–14 WORK$_IDEAL_MVIEW

Column Name	Type	Constraints	Description
sql_text_hash	NUMBER	Not NULL	SQL statement signature.
lib_cache_addr	VARCHAR2(16)	Not NULL	Associates this materialized view usage with a specific cursor.
group_by_columns	VARCHAR2(2000)	Not NULL	Comma-separated list of qualified column references in the GROUP BY clause of the ideal materialized view. These columns references also appear in the SELECT list.
where_clause	VARCHAR2(2000)		'AND'-separated list of inner equijoins, as they would appear in the WHERE clause of the ideal materialized view.
from_clause	VARCHAR2(2000)	Not NULL	Comma-separated list of owner-qualified relation names and aliases.
measure	VARCHAR2(2000)		Comma-separated list of <groupingFunction> (<expression>), as it would appear in the SELECT list of the ideal materialized view.
idl_sum_flags	NUMBER		Flag vector (4 bytes). Currently all bits are undefined.
summary_owner	VARCHAR2(30)		Owner of an existing materialized view used by query rewrite, or NULL if none.
summary_name	VARCHAR2(30)		Name of an existing materialized view used by query rewrite, or NULL if none.
actual_benefit	NUMBER	>0	Actual performance benefit of using this materialized view.

WORK$_MVIEW_USAGE

This view is based on the table V_1992216243_F_5_E_15_8_1 which is generated by the Oracle Trace format operation. Each row of this table corresponds to an actual query rewrite.

Column Name	Type	Constraints	Description
sql_text_hash	NUMBER	Not NULL	SQL statement signature.
lib_cache_addr	VARCHAR2(16)	Not NULL	Associates this materialized view usage with a specific cursor.

24

DBMS_ORACLE_TRACE_AGENT

The DBMS_ORACLE_TRACE_AGENT package provides some system level utilities.

Security

This package is only accessible to user SYS by default. You can control access to these routines by only granting execute to privileged users.

> **Note:** This package should only be granted to DBA or the Oracle TRACE collection agent.

Summary of Subprograms

This package contains only one subprogram: SET_ORACLE_TRACE_IN_SESSION.

SET_ORACLE_TRACE_IN_SESSION procedure

This procedure collects Oracle Trace data for a database session other than your own. It enables Oracle TRACE in the session identified by (sid, serial#). These value are taken from v$session.

Syntax

```
DBMS_ORACLE_TRACE_AGENT.SET_ORACLE_TRACE_IN_SESSION (
   sid                NUMBER   DEFAULT 0,
   serial#            NUMBER   DEFAULT 0,
   on_off IN          BOOLEAN  DEFAULT false,
   collection_name IN VARCHAR2 DEFAULT '',
   facility_name   IN VARCHAR2 DEFAULT '');
```

Parameters

Table 24–1 SET_ORACLE_TRACE_IN_SESSION Procedure Parameters

Parameter	Description
sid	Session ID.
serial#	Session serial number.
on_off	TRUE or FALSE. Turns tracing on or off.
collection_name	The Oracle TRACE collection name to be used.
facility_name	The Oracle TRACE facility name to be used.

Usage Notes

If the collection does not occur, then check the following:

- Be sure that the server event set file identified by <facility_name> exists. If there is no full file specification on this field, then the file should be located in the directory identified by ORACLE_TRACE_FACILITY_PATH in the initialization file.
- The following files should exist in your Oracle Trace admin directory: REGID.DAT, PROCESS.DAT, and COLLECT.DAT. If they do not, then you must run the OTRCCREF executable to create them.

  Note: PROCESS.DAT was changed to FACILITY.DAT with Oracle8.

- The stored procedure packages should exist in the database. If the packages do not exist, then run the OTRCSVR.SQL file (in your Oracle Trace or RDBMS admin directories) to create the packages.
- The user has the EXECUTE privilege on the stored procedure.

Example

```
EXECUTE DBMS_ORACLE_TRACE_AGENT.SET_ORACLE_TRACE_IN_SESSION
(8,12,TRUE,'NEWCOLL','oracled');
```

25

DBMS_ORACLE_TRACE_USER

DBMS_ORACLE_TRACE_USER provides public access to the Oracle TRACE instrumentation for the calling user. Using the Oracle Trace stored procedures, you can invoke an Oracle Trace collection for your own session or for another session.

Summary of Subprograms

This package contains only one subprogram: SET_ORACLE_TRACE.

SET_ORACLE_TRACE procedure

This procedure collects Oracle Trace data for your own database session.

Syntax

```
DBMS_ORACLE_TRACE_USER.SET_ORACLE_TRACE (
   on_off          IN BOOLEAN  DEFAULT false,
   collection_name IN VARCHAR2 DEFAULT '',
   facility_name   IN VARCHAR2 DEFAULT '');
```

Parameters

Table 25–1 SET_ORACLE_TRACE Procedure Parameters

Parameter	Description
on_off	TRUE or FALSE: Turns tracing on or off.
collection_name	Oracle TRACE collection name to be used.
facility_name	Oracle TRACE facility name to be used.

Example

```
EXECUTE DBMS_ORACLE_TRACE_USER.SET_ORACLE_TRACE
(TRUE,'MYCOLL','oracle');
```

26

DBMS_OUTPUT

The DBMS_OUTPUT package enables you to send messages from stored procedures, packages, and triggers.

The PUT and PUT_LINE procedures in this package enable you to place information in a buffer that can be read by another trigger, procedure, or package. In a separate PL/SQL procedure or anonymous block, you can display the buffered information by calling the GET_LINE procedure.

If you do not call GET_LINE, or if you do not display the messages on your screen in SQL*Plus or Enterprise Manager, then the buffered messages are ignored. The DBMS_OUTPUT package is especially useful for displaying PL/SQL debugging information.

Note: Messages sent using DBMS_OUTPUT are not actually sent until the sending subprogram or trigger completes. There is no mechanism to flush output during the execution of a procedure.

Security

At the end of this script, a public synonym (DBMS_OUTPUT) is created and EXECUTE permission on this package is granted to public.

Errors

DBMS_OUTPUT subprograms raise the application error ORA-20000, and the output procedures can return the following errors:

Table 26–1 DBMS_OUTPUT Errors

Error	Description
ORU-10027:	Buffer overflow
ORU-10028:	Line length overflow

Types

Type CHARARR is a table type.

Using DBMS_OUTPUT

A trigger might want to print out some debugging information. To do this, the trigger would do:

```
DBMS_OUTPUT.PUT_LINE('I got here:'||:new.col||' is the new value');
```

If you have enabled the DBMS_OUTPUT package, then this PUT_LINE would be buffered, and you could, after executing the statement (presumably some INSERT, DELETE, or UPDATE that caused the trigger to fire), get the line of information back. For example:

```
BEGIN
   DBMS_OUTPUT.GET_LINE(:buffer, :status);
END;
```

It could then display the buffer on the screen. You repeat calls to GET_LINE until status comes back as non-zero. For better performance, you should use calls to GET_LINES which can return an array of lines.

Enterprise Manager and SQL*Plus implement a SET SERVEROUTPUT ON command to know whether to make calls to GET_LINE(S) after issuing INSERT, UPDATE,

DELETE or anonymous PL/SQL calls (these are the only ones that can cause triggers or stored procedures to be executed).

Summary of Subprograms

Table 26–2 DBMS_OUTPUT Package Subprograms

Subprogram	Description
ENABLE procedure on page 26-3	Enables message output.
DISABLE procedure on page 26-4	Disables message output.
PUT and PUT_LINE procedures on page 26-4	Places a line in the buffer.
PUT and PUT_LINE procedures on page 26-4	Places partial line in buffer.
NEW_LINE procedure on page 26-6	Terminates a line created with PUT.
GET_LINE and GET_LINES procedures on page 26-7	Retrieves one line, or an array of lines, from buffer.

ENABLE procedure

This procedure enables calls to PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES. Calls to these procedures are ignored if the DBMS_OUTPUT package is not enabled.

Note: It is not necessary to call this procedure when you use the SERVEROUTPUT option of Enterprise Manager or SQL*Plus.

If there are multiple calls to ENABLE, then buffer_size is the largest of the values specified. The maximum size is 1,000,000, and the minimum is 2,000.

Syntax

```
DBMS_OUTPUT.ENABLE (
   buffer_size IN INTEGER DEFAULT 20000);
```

Parameters

Table 26–3 ENABLE Procedure Parameters

Parameter	Description
buffer_size	Amount of information, in bytes, to buffer.

Pragmas

```
pragma restrict_references(enable,WNDS,RNDS);
```

Errors

Table 26–4 ENABLE Procedure Errors

Error	Description
ORA-20000:, ORU-10027:	Buffer overflow, limit of <buffer_limit> bytes.

DISABLE procedure

This procedure disables calls to PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES, and purges the buffer of any remaining information.

As with ENABLE, you do not need to call this procedure if you are using the SERVEROUTPUT option of Enterprise Manager or SQL*Plus.

Syntax

```
DBMS_OUTPUT.DISABLE;
```

Parameters

None.

Pragmas

```
pragma restrict_references(disable,WNDS,RNDS);
```

PUT and PUT_LINE procedures

You can either place an entire line of information into the buffer by calling PUT_LINE, or you can build a line of information piece by piece by making multiple calls

to PUT. Both of these procedures are overloaded to accept items of type VARCHAR2, NUMBER, or DATE to place in the buffer.

All items are converted to VARCHAR2 as they are retrieved. If you pass an item of type NUMBER or DATE, then when that item is retrieved, it is formatted with TO_CHAR using the default format. If you want to use a different format, then you should pass in the item as VARCHAR2 and format it explicitly.

When you call PUT_LINE, the item that you specify is automatically followed by an end-of-line marker. If you make calls to PUT to build a line, then you must add your own end-of-line marker by calling NEW_LINE. GET_LINE and GET_LINES do not return lines that have not been terminated with a newline character.

If your line exceeds the buffer limit, then you receive an error message.

Note: Output that you create using PUT or PUT_LINE is buffered. The output cannot be retrieved until the PL/SQL program unit from which it was buffered returns to its caller.

For example, Enterprise Manager or SQL*Plus do not display DBMS_OUTPUT messages until the PL/SQL program completes. There is no mechanism for flushing the DBMS_OUTPUT buffers within the PL/SQL program. For example:

```
SQL> SET SERVER OUTPUT ON
SQL> BEGIN
     2 DBMS_OUTPUT.PUT_LINE ('hello');
     3 DBMS_LOCK.SLEEP (10);
     4 END;
```

Syntax

```
DBMS_OUTPUT.PUT      (item IN NUMBER);
DBMS_OUTPUT.PUT      (item IN VARCHAR2);
DBMS_OUTPUT.PUT      (item IN DATE);
DBMS_OUTPUT.PUT_LINE (item IN NUMBER);
DBMS_OUTPUT.PUT_LINE (item IN VARCHAR2);
DBMS_OUTPUT.PUT_LINE (item IN DATE);
DBMS_OUTPUT.NEW_LINE;
```

Parameters

Table 26–5 PUT and PUT_LINE Procedure Parameters

Parameter	Description
a	Item to buffer.

Errors

Table 26–6 PUT and PUT_LINE Procedure Errors

Error	Description
ORA-20000, ORU-10027:	Buffer overflow, limit of <buf_limit> bytes.
ORA-20000, ORU-10028:	Line length overflow, limit of 255 bytes per line.

NEW_LINE procedure

This procedure puts an end-of-line marker. GET_LINE(S) returns "lines" as delimited by "newlines". Every call to PUT_LINE or NEW_LINE generates a line that is returned by GET_LINE(S).

Syntax

```
DBMS_OUTPUT.NEW_LINE;
```

Parameters

None.

Errors

Table 26–7 NEW_LINE Procedure Errors

Error	Description
ORA-20000, ORU-10027:	Buffer overflow, limit of <buf_limit> bytes.
ORA-20000, ORU-10028:	Line length overflow, limit of 255 bytes per line.

GET_LINE and GET_LINES procedures

You can choose to retrieve from the buffer a single line or an array of lines. Call the GET_LINE procedure to retrieve a single line of buffered information. To reduce the number of calls to the server, call the GET_LINES procedure to retrieve an array of lines from the buffer.

You can choose to automatically display this information if you are using Enterprise Manager or SQL*Plus by using the special SET SERVEROUTPUT ON command.

After calling GET_LINE or GET_LINES, any lines not retrieved before the next call to PUT, PUT_LINE, or NEW_LINE are discarded to avoid confusing them with the next message.

Syntax

```
DBMS_OUTPUT.GET_LINE (
   line   OUT VARCHAR2,
   status OUT INTEGER);
```

Parameters

Table 26–8 GET_LINE Procedure Parameters

Parameter	Description
line	Returns a single line of buffered information, excluding a final newline character: The maximum length is 255 bytes.
status	If the call completes successfully, then the status returns as 0. If there are no more lines in the buffer, then the status is 1.

Syntax

```
DBMS_OUTPUT.GET_LINES (
   lines       OUT  CHARARR,
   numlines IN OUT  INTEGER);
```

CHARARR is a table of VARCHAR2(255).

Parameters

Table 26–9 GET_LINES Procedure Parameters

Parameter	Description
lines	Returns an array of lines of buffered information. The maximum length of each line in the array is 255 bytes.
numlines	Number of lines you want to retrieve from the buffer. After retrieving the specified number of lines, the procedure returns the number of lines actually retrieved. If this number is less than the number of lines requested, then there are no more lines in the buffer.

Examples

The DBMS_OUTPUT package is commonly used to debug stored procedures and triggers, as shown in Example 1. This package can also be used to enable you to retrieve information about an object and format this output, as shown in Example 2 on page 26-9.

Example 1 This is an example of a function that queries the employee table and returns the total salary for a specified department. The function includes several calls to the PUT_LINE procedure:

```
CREATE FUNCTION dept_salary (dnum NUMBER) RETURN NUMBER IS
   CURSOR emp_cursor IS
      SELECT sal, comm FROM emp WHERE deptno = dnum;
   total_wages    NUMBER(11, 2) := 0;
   counter        NUMBER(10) := 1;
BEGIN

   FOR emp_record IN emp_cursor LOOP
      emp_record.comm := NVL(emp_record.comm, 0);
      total_wages := total_wages + emp_record.sal
         + emp_record.comm;
      DBMS_OUTPUT.PUT_LINE('Loop number = ' || counter ||
         '; Wages = '|| TO_CHAR(total_wages));    /* Debug line */
      counter := counter + 1; /* Increment debug counter */
   END LOOP;
   /* Debug line */
   DBMS_OUTPUT.PUT_LINE('Total wages = ' ||
     TO_CHAR(total_wages));
   RETURN total_wages;
```

```
END dept_salary;
```

Assume the EMP table contains the following rows:

```
EMPNO          SAL     COMM     DEPT
-----        ------- -------- -------
1002           1500      500       20
1203           1000                30
1289           1000                10
1347           1000      250       20
```

Assume the user executes the following statements in the Enterprise Manager SQL Worksheet input pane:

```
SET SERVEROUTPUT ON
VARIABLE salary NUMBER;
EXECUTE :salary := dept_salary(20);
```

The user would then see the following information displayed in the output pane:

```
Loop number = 1; Wages = 2000
Loop number = 2; Wages = 3250
Total wages = 3250


PL/SQL procedure successfully executed.
```

Example 2 In this example, the user has used the EXPLAIN PLAN command to retrieve information about the execution plan for a statement and has stored it in PLAN_TABLE. The user has also assigned a statement ID to this statement. The example EXPLAIN_OUT procedure retrieves the information from this table and formats the output in a nested manner that more closely depicts the order of steps undergone in processing the SQL statement.

```
 /************************************************************/
/* Create EXPLAIN_OUT procedure. User must pass STATEMENT_ID to */
/* to procedure, to uniquely identify statement.               */
/************************************************************/
CREATE OR REPLACE PROCEDURE explain_out
   (statement_id IN VARCHAR2) AS

   -- Retrieve information from PLAN_TABLE into cursor EXPLAIN_ROWS.

   CURSOR explain_rows IS
```

```
      SELECT level, id, position, operation, options,
         object_name
      FROM plan_table
      WHERE statement_id = explain_out.statement_id
      CONNECT BY PRIOR id = parent_id
         AND statement_id = explain_out.statement_id
      START WITH id = 0
       ORDER BY id;

BEGIN

   -- Loop through information retrieved from PLAN_TABLE:

   FOR line IN explain_rows LOOP

      -- At start of output, include heading with estimated cost.

      IF line.id = 0 THEN
         DBMS_OUTPUT.PUT_LINE ('Plan for statement '
            || statement_id
            || ', estimated cost = ' || line.position);
      END IF;

      -- Output formatted information. LEVEL determines indention level.

      DBMS_OUTPUT.PUT_LINE (lpad(' ',2*(line.level-1)) ||
         line.operation || ' ' || line.options || ' ' ||
         line.object_name);
   END LOOP;

END;
```

See Also: Chapter 28, "UTL_FILE"

27

DBMS_PCLXUTIL

The DBMS_PCLXUTIL package provides intra-partition parallelism for creating partition-wise local indexes.

> **See Also:** There are several rules concerning partitions and indexes. For more information, see *Oracle8i Concepts* and *Oracle8i Administrator's Guide*.

DBMS_PCLXUTIL circumvents the limitation that, for local index creation, the degree of parallelism is restricted to the number of partitions as only one slave process per partition is utilized.

DBMS_PCLXUTIL uses the DBMS_JOB package to provide a greater degree of parallelism for creating a local index for a partitioned table. This is achieved by asynchronous inter-partition parallelism using the background processes (with DBMS_JOB), in combination with intra-partition parallelism using the parallel query slave processes.

DBMS_PCLXUTIL works with both range and range-hash composite partitioning.

Note: For range partitioning, the minimum compatibility mode is 8.0; for range-hash composite partitioning, the minimum compatibility mode is 8*i*.

Using DBMS_PCLXUTIL

The DBMS_PCLXUTIL package can be used during the following DBA tasks:

1. Local index creation

   The procedure BUILD_PART_INDEX assumes that the dictionary information for the local index already exists. This can be done by issuing the create index SQL command with the UNUSABLE option.

   ```
   CREATE INDEX <idx_name> on <tab_name>(...) local(...) unusable;
   ```

   This causes the dictionary entries to be created without "building" the index itself, the time consuming part of creating an index. Now, invoking the procedure BUILD_PART_INDEX causes a concurrent build of local indexes with the specified degree of parallelism.

   ```
   EXECUTE dbms_pclxutil.build_part_index(4,4,<idx_name>,<tab_name>,FALSE);
   ```

   For composite partitions, the procedure automatically builds local indices for all subpartitions of the composite table.

2. Local index maintenance

   By marking desired partitions usable or unusable, the BUILD_PART_INDEX procedure also enables selective rebuilding of local indexes. The force_opt parameter provides a way to override this and build local indexes for all partitions.

   ```
   ALTER INDEX <idx_name> local(...) unusable;
   ```

   Rebuild only the desired (sub)partitions (that are marked unusable):

   ```
   EXECUTE dbms_pclxutil.build_part_index(4,4,<idx_name>,<tab_name>,FALSE);
   ```

   Rebuild all (sub)partitions using force_opt = TRUE:

   ```
   EXECUTE dbms_pclxutil.build_part_index(4,4,<idx_name>,<tab_name>,TRUE);
   ```

   A progress report is produced, and the output appears on screen when the program is ended (because the DBMS_OUTPUT package writes messages to a buffer first, and flushes the buffer to the screen only upon termination of the program).

Limitations

Because DBMS_PCLXUTIL uses the DBMS_JOB package, you must be aware of the following limitations pertaining to DBMS_JOB:

- You must decide appropriate values for job_queue_processes and job_queue_interval initalization parameters. Clearly, if the job processes are not started before calling BUILD_PART_INDEX(), then the package will not function properly. The background processes are specified by the following init.ora parameters:

```
job_queue_processes=n   #the number of background processes = n
job_queue_interval=m    #the processes wake-up every m seconds
```

- There is an upper limit to the number of simultaneous jobs in the queue, dictated by the upper limit on the number of background processes marked SNP[0..9] and SNP[A..Z], which is 36.

  See Also: *Oracle8i Administrator's Guide*

  Therefore, the upper limit on jobs_per_batch is MIN(#partitions, #job_queue_processes). The default value for jobs_per_batch is 1; i.e., indexes will be built one partition at a time.

- Failure conditions are reported only in the trace files (a DBMS_JOB limitation), making it impossible to give interactive feedback to the user. This package simply prints a failure message, removes unfinished jobs from the queue, and requests the user to take a look at the snp*.trc trace files.
- The primary ramification of the above point is that you are expected to know how to tune oracle (especially to set various storage parameters) in order to build large indexes. This package is not intended to assist in that tuning process.

Summary of Subprograms

DBMS_PCLXUTIL contains just one procedure: BUILD_PART_INDEX.

BUILD_PART_INDEX procedure

Syntax

```
DBMS_PCLXUTIL.build_part_index (
   jobs_per_batch  IN NUMBER   DEFAULT 1,
   procs_per_job   IN NUMBER   DEFAULT 1,
   tab_name        IN VARCHAR2 DEFAULT NULL,
   idx_name        IN VARCHAR2 DEFAULT NULL,
   force_opt       IN BOOLEAN  DEFAULT FALSE);
```

Parameters

Table 27–1 BUILD_PART_INDEX Procedure Parameters

Parameter	Description
jobs_per_batch	Number of local indexes to be built concurrently (1 <= jobs_per_batch <= number of partitions).
procs_per_job	Number of parallel query slaves to be utilized per local index build (1 <= procs_per_job <= max_slaves).
tab_name	Name of the partitioned table (an exception is raised if the table does not exist or not partitioned).
idx_name	Name given to the local index (an exception is raised if a local index is not created on the table tab_name).
force_opt	If TRUE, then force rebuild of all partitioned indices; otherwise, rebuild only the partitions marked 'UNUSABLE'.

Example

Suppose a table PROJECT is created with two partitions PROJ001 and PROJ002, along with a local index IDX.

A call to the procedure BUILD_PART_INDEX(2,4,'PROJECT','IDX',TRUE) produces the following output:

```
SVRMGR> EXECUTE dbms_pclxutil.build_part_index(2,4,'PROJECT','IDX',TRUE);
Statement processed.
INFO: Job #21 created for partition PROJ002 with 4 slaves
INFO: Job #22 created for partition PROJ001 with 4 slaves
SVRMGR>
```

28

DBMS_PIPE

The DBMS_PIPE package lets two or more sessions in the same instance communicate. Oracle pipes are similar in concept to the pipes used in UNIX, but Oracle pipes are not implemented using the operating system pipe mechanisms.

Information sent through Oracle pipes is buffered in the system global area (SGA). All information in pipes is lost when the instance is shut down.

Depending upon your security requirements, you may choose to use either a *public* or a *private* pipe.

Caution: Pipes are independent of transactions. Be careful using pipes when transaction control can be affected.

Public Pipes

You may create a public pipe either implicitly or explicitly. For *implicit* public pipes, the pipe is automatically created when it is referenced for the first time, and it disappears when it no longer contains data. Because the pipe descriptor is stored in the SGA, there is some space usage overhead until the empty pipe is aged out of the cache.

You create an *explicit* public pipe by calling the CREATE_PIPE function with the `private` flag set to FALSE. You must deallocate explicitly-created pipes by calling the REMOVE_PIPE function.

The domain of a public pipe is the schema in which it was created, either explicitly or implicitly.

Writing and Reading Pipes

Each public pipe works asynchronously. Any number of schema users can write to a public pipe, as long as they have EXECUTE permission on the DBMS_PIPE package, and they know the name of the public pipe. However, once buffered information is read by one user, it is emptied from the buffer, and is not available for other readers of the same pipe.

The sending session builds a message using one or more calls to the PACK_MESSAGE procedure. This procedure adds the message to the session's local message buffer. The information in this buffer is sent by calling the SEND_MESSAGE function, designating the pipe name to be used to send the message. When SEND_MESSAGE is called, all messages that have been stacked in the local buffer are sent.

A process that wants to receive a message calls the RECEIVE_MESSAGE function, designating the pipe name from which to receive the message. The process then calls the UNPACK_MESSAGE procedure to access each of the items in the message.

Private Pipes

You explicitly create a private pipe by calling the CREATE_PIPE function. Once created, the private pipe persists in shared memory until you explicitly deallocate it by calling the REMOVE_PIPE function. A private pipe is also deallocated when the database instance is shut down.

You cannot create a private pipe if an implicit pipe exists in memory and has the same name as the private pipe you are trying to create. In this case, CREATE_PIPE returns an error.

Access to a private pipe is restricted to:

- Sessions running under the same userid as the creator of the pipe
- Stored subprograms executing in the same userid privilege domain as the pipe creator
- Users connected as SYSDBA or INTERNAL

An attempt by any other user to send or receive messages on the pipe, or to remove the pipe, results in an immediate error. Any attempt by another user to create a pipe with the same name also causes an error.

As with public pipes, you must first build your message using calls to PACK_MESSAGE before calling SEND_MESSAGE. Similarly, you must call RECEIVE_MESSAGE to retrieve the message before accessing the items in the message by calling UNPACK_MESSAGE.

Pipe Uses

The pipe functionality has several potential applications:

- External service interface: You can communicate with user-written services that are external to the RDBMS. This can be done in a (effectively) multi-threaded manner, so that several instances of the service are executing simultaneously. Additionally, the services are available asynchronously. The requestor of the service does not need to block a waiting reply. The requestor can check (with or without timeout) at a later time. The service can be written in any of the 3GL languages that Oracle supports.
- Independent transactions: The pipe can communicate to a separate session which can perform an operation in an independent transaction (such as logging an attempted security violation detected by a trigger).
- Alerters (non-transactional): You can post another process without requiring the waiting process to poll. If an "after-row" or "after-statement" trigger were to alert an application, then the application would treat this alert as an indication that the data probably changed. The application would then read the data to get the current value. Because this is an "after" trigger, the application would want to do a "select for update" to make sure it read the correct data.
- Debugging: Triggers and stored procedures can send debugging information to a pipe. Another session can keep reading out of the pipe and display it on the screen or write it to a file.
- Concentrator: This is useful for multiplexing large numbers of users over a fewer number of network connections, or improving performance by concentrating several user-transactions into one DBMS transaction.

Security

Security can be achieved by use of 'grant execute' on the DBMS_PIPE package by creating a pipe using the private parameter in the CREATE_PIPE function and by writing cover packages that only expose particular features or pipenames to particular users or roles.

Constants

```
maxwait   constant integer := 86400000; /* 1000 days */
```

This is the maximum time to wait attempting to send or receive a message.

Errors

DBMS_PIPE package subprograms can return the following errors:

Table 28–1 DBMS_PIPE Errors

Error	Description
ORA-23321:	Pipename may not be null. This can be returned by the CREATE_PIPE function, or any subprogram that takes a pipe name as a parameter.
ORA-23322:	Insufficient privilege to access pipe. This can be returned by any subprogram that references a private pipe in its parameter list.

Summary of Subprograms

Table 28–2 DBMS_PIPE Package Subprograms

Subprogram	Description
CREATE_PIPE function on page 28-5	Explicitly creates a pipe (necessary for private pipes).
PACK_MESSAGE procedure on page 28-7	Builds message in local buffer.
SEND_MESSAGE function on page 28-8	Sends message on named pipe: This implicitly creates a public pipe if the named pipe does not exist.
RECEIVE_MESSAGE function on page 28-10	Copies message from named pipe into local buffer.

Table 28–2 DBMS_PIPE Package Subprograms

Subprogram	Description
NEXT_ITEM_TYPE function on page 28-12	Returns datatype of next item in buffer.
UNPACK_MESSAGE procedure on page 28-13	Accesses next item in buffer.
REMOVE_PIPE function on page 28-14	Removes the named pipe.
PURGE procedure on page 28-15	Purges contents of named pipe.
RESET_BUFFER procedure on page 28-16	Purges contents of local buffer.
UNIQUE_SESSION_NAME function on page 28-17	Returns unique session name.

CREATE_PIPE function

This function explicitly creates a public or private pipe. If the `private` flag is `TRUE`, then the pipe creator is assigned as the owner of the private pipe.

Explicitly-created pipes can only be removed by calling `REMOVE_PIPE`, or by shutting down the instance.

Syntax

```
DBMS_PIPE.CREATE_PIPE (
   pipename    IN VARCHAR2,
   maxpipesize IN INTEGER DEFAULT 8192,
   private     IN BOOLEAN DEFAULT TRUE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(create_pipe,WNDS,RNDS);
```

Parameters

Table 28–3 CREATE_PIPE Function Parameters

Parameter	Description
pipename	Name of the pipe you are creating. You must use this name when you call SEND_MESSAGE and RECEIVE_MESSAGE. This name must be unique across the instance. Caution: Do not use pipe names beginning with ORA$. These are reserved for use by procedures provided by Oracle Corporation. Pipename should not be longer than 128 bytes, and is case_insensitive. At this time, the name cannot contain NLS characters.
maxpipesize	The maximum size allowed for the pipe, in bytes. The total size of all of the messages on the pipe cannot exceed this amount. The message is blocked if it exceeds this maximum. The default maxpipesize is 8192 bytes. The maxpipesize for a pipe becomes a part of the characteristics of the pipe and persists for the life of the pipe. Callers of SEND_MESSAGE with larger values cause the maxpipesize to be increased. Callers with a smaller value use the existing, larger value.
private	Uses the default, TRUE, to create a private pipe. Public pipes can be implicitly created when you call SEND_MESSAGE.

Returns

Table 28–4 CREATE_PIPE Function Returns

Return	Description
0	Successful. If the pipe already exists and the user attempting to create it is authorized to use it, then Oracle returns 0, indicating success, and any data already in the pipe remains. If a user connected as SYSDBA/SYSOPER re-creates a pipe, then Oracle returns status 0, but the ownership of the pipe remains unchanged.

Table 28–4 CREATE_PIPE Function Returns

Return	Description
ORA-23322	Failure due to naming conflict. If a pipe with the same name exists and was created by a different user, then Oracle signals error ORA-23322, indicating the naming conflict.

Exceptions

Table 28–5 CREATE_PIPE Function Exception

Exception	Description
Null pipe name	Permission error: Pipe with the same name already exists, and you are not allowed to use it.

PACK_MESSAGE procedure

This procedure builds your message in the local message buffer.

To send a message, first make one or more calls to PACK_MESSAGE. Then, call SEND_MESSAGE to send the message in the local buffer on the named pipe.

The PACK_MESSAGE procedure is overloaded to accept items of type VARCHAR2, NUMBER, or DATE. In addition to the data bytes, each item in the buffer requires one byte to indicate its type, and two bytes to store its length. One additional byte is needed to terminate the message.The overhead for all types other than VARCHAR is 4 bytes.

In Oracle8, the char-set-id (2 bytes) and the char-set-form (1 byte) are stored with each data item. Therefore, the overhead when using Oracle8 is 7 bytes.

When you call SEND_MESSAGE to send this message, you must indicate the name of the pipe on which you want to send the message. If this pipe already exists, then you must have sufficient privileges to access this pipe. If the pipe does not already exist, then it is created automatically.

Syntax

```
DBMS_PIPE.PACK_MESSAGE       (item  IN  VARCHAR2);
DBMS_PIPE.PACK_MESSAGE       (item  IN  NCHAR);
DBMS_PIPE.PACK_MESSAGE       (item  IN  NUMBER);
DBMS_PIPE.PACK_MESSAGE       (item  IN  DATE);
DBMS_PIPE.PACK_MESSAGE_RAW   (item  IN  RAW);
DBMS_PIPE.PACK_MESSAGE_ROWID (item  IN  ROWID);
```

Note: The PACK_MESSAGE procedure is overloaded to accept items of type VARCHAR2, NCHAR, NUMBER, or DATE. There are two additional procedures to pack RAW and ROWID items.

Pragmas

```
pragma restrict_references(pack_message,WNDS,RNDS);
pragma restrict_references(pack_message_raw,WNDS,RNDS);
pragma restrict_references(pack_message_rowid,WNDS,RNDS);
```

Parameters

Table 28–6 PACK_MESSAGE Procedure Parameters

Parameter	Description
item	Item to pack into the local message buffer.

Exceptions

ORA-06558 is raised if the message buffer overflows (currently 4096 bytes). Each item in the buffer takes one byte for the type, two bytes for the length, plus the actual data. There is also one byte needed to terminate the message.

SEND_MESSAGE function

This function sends a message on the named pipe.

The message is contained in the local message buffer, which was filled with calls to PACK_MESSAGE. A pipe could be explicitly using CREATE_PIPE; otherwise, it is created implicitly.

Syntax

```
DBMS_PIPE.SEND_MESSAGE (
    pipename     IN VARCHAR2,
    timeout      IN INTEGER DEFAULT MAXWAIT,
    maxpipesize  IN INTEGER DEFAULT 8192)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(send_message,WNDS,RNDS);
```

Parameters

Table 28–7 SEND_MESSAGE Function Parameters

Parameter	Description
pipename	Name of the pipe on which you want to place the message. If you are using an explicit pipe, then this is the name that you specified when you called CREATE_PIPE. Caution: Do not use pipe names beginning with 'ORA$'. These names are reserved for use by procedures provided by Oracle Corporation. Pipename should not be longer than 128 bytes, and is case-insensitive. At this time, the name cannot contain NLS characters.
timeout	Time to wait while attempting to place a message on a pipe, in seconds. The default value is the constant MAXWAIT, which is defined as 86400000 (1000 days).
maxpipesize	Maximum size allowed for the pipe, in bytes. The total size of all the messages on the pipe cannot exceed this amount. The message is blocked if it exceeds this maximum. The default is 8192 bytes. The maxpipesize for a pipe becomes a part of the characteristics of the pipe and persists for the life of the pipe. Callers of SEND_MESSAGE with larger values cause the maxpipesize to be increased. Callers with a smaller value simply use the existing, larger value. Specifying maxpipesize as part of the SEND_MESSAGE procedure eliminates the need for a separate call to open the pipe. If you created the pipe explicitly, then you can use the optional maxpipesize parameter to override the creation pipe size specifications.

Returns

Table 28–8 SEND_MESSAGE Function Returns

Return	Description
0	Success. If the pipe already exists and the user attempting to create it is authorized to use it, then Oracle returns 0, indicating success, and any data already in the pipe remains. If a user connected as SYSDBS/SYSOPER re-creates a pipe, then Oracle returns status 0, but the ownership of the pipe remains unchanged.
1	Timed out. This procedure can timeout either because it cannot get a lock on the pipe, or because the pipe remains too full to be used. If the pipe was implicitly-created and is empty, then it is removed.
3	An interrupt occurred. If the pipe was implicitly created and is empty, then it is removed.
ORA-23322	Insufficient privileges. If a pipe with the same name exists and was created by a different user, then Oracle signals error ORA-23322, indicating the naming conflict.

Exceptions

Table 28–9 SEND_MESSAGE Function Exception

Exception	Description
Null pipe name	Permission error. Insufficient privilege to write to the pipe. The pipe is private and owned by someone else.

RECEIVE_MESSAGE function

This function copies the message into the local message buffer.

To receive a message from a pipe, first call RECEIVE_MESSAGE. When you receive a message, it is removed from the pipe; hence, a message can only be received once. For implicitly-created pipes, the pipe is removed after the last record is removed from the pipe.

If the pipe that you specify when you call RECEIVE_MESSAGE does not already exist, then Oracle implicitly creates the pipe and waits to receive the message. If the message does not arrive within a designated timeout interval, then the call returns and the pipe is removed.

After receiving the message, you must make one or more calls to UNPACK_MESSAGE to access the individual items in the message. The UNPACK_MESSAGE procedure is overloaded to unpack items of type DATE, NUMBER, VARCHAR2, and there are two additional procedures to unpack RAW and ROWID items. If you do not know the type of data that you are attempting to unpack, then call NEXT_ITEM_TYPE to determine the type of the next item in the buffer.

Syntax

```
DBMS_PIPE.RECEIVE_MESSAGE (
   pipename     IN VARCHAR2,
   timeout      IN INTEGER      DEFAULT maxwait)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(receive_message,WNDS,RNDS);
```

Parameters

Table 28–10 RECEIVE_MESSAGE Function Parameters

Parameter	Description
pipename	Name of the pipe on which you want to receive a message. Names beginning with ORA$ are reserved for use by Oracle
timeout	Time to wait for a message, in seconds. The default value is the constant MAXWAIT, which is defined as 86400000 (1000 days). A timeout of 0 allows you to read without blocking.

Returns

Table 28–11 RECEIVE_MESSAGE Function Returns

Return	Description
0	Success

Table 28–11 RECEIVE_MESSAGE Function Returns

Return	Description
1	Timed out. If the pipe was implicitly-created and is empty, then it is removed.
2	Record in the pipe is too large for the buffer. (This should not happen.)
3	An interrupt occurred.
ORA-23322	User has insufficient privileges to read from the pipe.

Exceptions

Table 28–12 RECEIVE_MESSAGE Function Exceptions

Exception	Description
Null pipe name	Permission error. Insufficient privilege to remove the record from the pipe. The pipe is owned by someone else.

NEXT_ITEM_TYPE function

This function determines the datatype of the next item in the local message buffer.

After you have called RECEIVE_MESSAGE to place pipe information in a local buffer, call NEXT_ITEM_TYPE.

Syntax

```
DBMS_PIPE.NEXT_ITEM_TYPE
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(next_item_type,WNDS,RNDS);
```

Returns

Table 28–13 NEXT_ITEM_TYPE Function Returns

Return	Description
0	No more items
6	NUMBER

Table 28–13 NEXT_ITEM_TYPE Function Returns

Return	Description
9	VARCHAR2
11	ROWID
12	DATE
23	RAW

UNPACK_MESSAGE procedure

This procedure retrieves items from the buffer.

After you have called RECEIVE_MESSAGE to place pipe information in a local buffer, call UNPACK_MESSAGE.

Syntax

```
DBMS_PIPE.UNPACK_MESSAGE       (item  OUT VARCHAR2);
DBMS_PIPE.UNPACK_MESSAGE       (item  OUT NCHAR);
DBMS_PIPE.UNPACK_MESSAGE       (item  OUT NUMBER);
DBMS_PIPE.UNPACK_MESSAGE       (item  OUT DATE);
DBMS_PIPE.UNPACK_MESSAGE_RAW   (item  OUT RAW);
DBMS_PIPE.UNPACK_MESSAGE_ROWID (item  OUT ROWID);
```

Note: The UNPACK_MESSAGE procedure is overloaded to return items of type VARCHAR2, NCHAR, NUMBER, or DATE. There are two additional procedures to unpack RAW and ROWID items.

Pragmas

```
pragma restrict_references(unpack_message,WNDS,RNDS);
pragma restrict_references(unpack_message_raw,WNDS,RNDS);
pragma restrict_references(unpack_message_rowid,WNDS,RNDS);
```

Parameters

Table 28–14 UNPACK_MESSAGE Procedure Parameters

Parameter	Description
item	Argument to receive the next unpacked item from the local message buffer.

Exceptions

ORA-06556 or 06559 are generated if the buffer contains no more items, or if the item is not of the same type as that requested.

REMOVE_PIPE function

This function removes explicitly-created pipes.

Pipes created implicitly by SEND_MESSAGE are automatically removed when empty. However, pipes created explicitly by CREATE_PIPE are removed only by calling REMOVE_PIPE, or by shutting down the instance. All unconsumed records in the pipe are removed before the pipe is deleted.

This is similar to calling PURGE on an implicitly-created pipe.

Syntax

```
DBMS_PIPE.REMOVE_PIPE (
   pipename  IN  VARCHAR2)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(remove_pipe,WNDS,RNDS);
```

Parameters

Table 28–15 REMOVE_PIPE Function Parameters

Parameter	Description
pipename	Name of pipe that you want to remove.

Returns

Table 28–16 REMOVE_PIPE Function Returns

Return	Description
0	Success If the pipe does not exist, or if the pipe already exists and the user attempting to remove it is authorized to do so, then Oracle returns 0, indicating success, and any data remaining in the pipe is removed.

Table 28–16 REMOVE_PIPE Function Returns

Return	Description
ORA-23322	Insufficient privileges. If the pipe exists, but the user is not authorized to access the pipe, then Oracle signals error ORA-23322, indicating insufficient privileges.

Exceptions

Table 28–17 REMOVE_PIPE Function Exception

Exception	Description
Null pipe name	Permission error: Insufficient privilege to remove pipe. The pipe was created and is owned by someone else.

PURGE procedure

This procedure empties the contents of the named pipe.

An empty implicitly-created pipe is aged out of the shared global area according to the least-recently-used algorithm. Thus, calling PURGE lets you free the memory associated with an implicitly-created pipe.

Because PURGE calls RECEIVE_MESSAGE, the local buffer might be overwritten with messages as they are purged from the pipe. Also, you can receive an ORA-23322 (insufficient privileges) error if you attempt to purge a pipe with which you have insufficient access rights.

Syntax

```
DBMS_PIPE.PURGE (
   pipename  IN  VARCHAR2);
```

Pragmas

```
pragma restrict_references(purge,WNDS,RNDS);
```

Parameters

Table 28–18 Purge Procedure Parameters

Parameter	Description
pipename	Name of pipe from which to remove all messages. The local buffer may be overwritten with messages as they are discarded. Pipename should not be longer than 128 bytes, and is case-insensitive.

Exceptions

Permission error if pipe belongs to another user.

RESET_BUFFER procedure

This procedure resets the PACK_MESSAGE and UNPACK_MESSAGE positioning indicators to 0.

Because all pipes share a single buffer, you may find it useful to reset the buffer before using a new pipe. This ensures that the first time you attempt to send a message to your pipe, you do not inadvertently send an expired message remaining in the buffer.

Syntax

```
DBMS_PIPE.RESET_BUFFER;
```

Parameters

None.

Pragmas

```
pragma restrict_references(reset_buffer,WNDS,RNDS);
```

UNIQUE_SESSION_NAME function

This function receives a name that is unique among all of the sessions that are currently connected to a database.

Multiple calls to this function from the same session always return the same value. You might find it useful to use this function to supply the PIPENAME parameter for your SEND_MESSAGE and RECEIVE_MESSAGE calls.

Syntax

```
DBMS_PIPE.UNIQUE_SESSION_NAME
  RETURN VARCHAR2;
```

Parameters

None.

Pragmas

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

Returns

This function returns a unique name. The returned name can be up to 30 bytes.

Examples

Example 1: Debugging

This example shows a procedure a PL/SQL program can call to place debugging information in a pipe.

```
CREATE OR REPLACE PROCEDURE debug (msg VARCHAR2) AS
    status  NUMBER;
BEGIN
  DBMS_PIPE.PACK_MESSAGE(LENGTH(msg));
  DBMS_PIPE.PACK_MESSAGE(msg);
  status := DBMS_PIPE.SEND_MESSAGE('plsql_debug');
  IF status != 0 THEN
    raise_application_error(-20099, 'Debug error');
  END IF;
END debug;
```

This example shows the Pro*C code that receives messages from the PLSQL_DEBUG pipe in the PL/SQL example above and displays the messages. If the Pro*C session is run in a separate window, then it can be used to display any messages that are sent to the debug procedure from a PL/SQL program executing in a separate session.

```
#include <stdio.h>
#include <string.h>

EXEC SQL BEGIN DECLARE SECTION;
   VARCHAR username[20];
   int     status;
   int     msg_length;
   char    retval[2000];
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

void sql_error();

main()
{

-- Prepare username:
   strcpy(username.arr, "SCOTT/TIGER");
   username.len = strlen(username.arr);

   EXEC SQL WHENEVER SQLERROR DO sql_error();
   EXEC SQL CONNECT :username;

   printf("connected\n");

-- Start an endless loop to look for and print messages on the pipe:
   FOR (;;)
   {
      EXEC SQL EXECUTE
         DECLARE
            len INTEGER;
            typ INTEGER;
            sta INTEGER;
            chr VARCHAR2(2000);
         BEGIN
            chr := '';
            sta := dbms_pipe.receive_message('plsql_debug');
            IF sta = 0 THEN
```

```
            DBMS_PIPE.UNPACK_MESSAGE(len);
            DBMS_PIPE.UNPACK_MESSAGE(chr);
         END IF;
         :status := sta;
         :retval := chr;
         IF len IS NOT NULL THEN
            :msg_length := len;
         ELSE
            :msg_length := 2000;
         END IF;
      END;
   END-EXEC;
   IF (status == 0)
      printf("\n%.*s\n", msg_length, retval);
   ELSE
      printf("abnormal status, value is %d\n", status);
}
}

void sql_error()
{
   char msg[1024];
   int rlen, len;
   len = sizeof(msg);
   sqlglm(msg, &len, &rlen);
   printf("ORACLE ERROR\n");
   printf("%.*s\n", rlen, msg);
   exit(1);
}
```

Example 2: Execute System Commands

This example shows PL/SQL and Pro*C code let a PL/SQL stored procedure (or anonymous block) call PL/SQL procedures to send commands over a pipe to a Pro*C program that is listening for them.

The Pro*C program sleeps and waits for a message to arrive on the named pipe. When a message arrives, the C program processes it, carrying out the required action, such as executing a UNIX command through the *system*() call or executing a SQL command using embedded SQL.

DAEMON.SQL is the source code for the PL/SQL package. This package contains procedures that use the DBMS_PIPE package to send and receive message to and from the Pro*C daemon. Note that full handshaking is used. The daemon always sends a message back to the package (except in the case of the STOP command).

This is valuable, because it allows the PL/SQL procedures to be sure that the Pro*C daemon is running.

You can call the DAEMON packaged procedures from an anonymous PL/SQL block using SQL*Plus or Enterprise Manager. For example:

```
SVRMGR> variable rv number
SVRMGR> execute :rv := DAEMON.EXECUTE_SYSTEM('ls -la');
```

This would, on a UNIX system, cause the Pro*C daemon to execute the command *system*("*ls -la*").

Remember that the daemon needs to be running first. You might want to run it in the background, or in another window beside the SQL*Plus or Enterprise Manager session from which you call it.

The DAEMON.SQL also uses the DBMS_OUTPUT package to display the results. For this example to work, you must have execute privileges on this package.

DAEMON.SQL example This is the code for the PL/SQL DAEMON package:

```
CREATE OR REPLACE PACKAGE daemon AS
  FUNCTION execute_sql(command VARCHAR2,
                       timeout NUMBER DEFAULT 10)
    RETURN NUMBER;

  FUNCTION execute_system(command VARCHAR2,
                          timeout NUMBER DEFAULT 10)
    RETURN NUMBER;

  PROCEDURE stop(timeout NUMBER DEFAULT 10);
END daemon;
/
CREATE OR REPLACE PACKAGE BODY daemon AS

  FUNCTION execute_system(command VARCHAR2,
                          timeout NUMBER DEFAULT 10)
  RETURN NUMBER IS

    status       NUMBER;
    result       VARCHAR2(20);
    command_code NUMBER;
    pipe_name    VARCHAR2(30);
  BEGIN
    pipe_name := DBMS_PIPE.UNIQUE_SESSION_NAME;
```

```
  DBMS_PIPE.PACK_MESSAGE('SYSTEM');
  DBMS_PIPE.PACK_MESSAGE(pipe_name);
  DBMS_PIPE.PACK_MESSAGE(command);
  status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
  IF status <> 0 THEN
    RAISE_APPLICATION_ERROR(-20010,
      'Execute_system: Error while sending.  Status = ' ||
       status);
  END IF;

  status := DBMS_PIPE.RECEIVE_MESSAGE(pipe_name, timeout);
  IF status <> 0 THEN
    RAISE_APPLICATION_ERROR(-20011,
      'Execute_system: Error while receiving.
       Status = ' || status);
  END IF;

  DBMS_PIPE.UNPACK_MESSAGE(result);
  IF result <> 'done' THEN
    RAISE_APPLICATION_ERROR(-20012,
      'Execute_system: Done not received.');
  END IF;

  DBMS_PIPE.UNPACK_MESSAGE(command_code);
  DBMS_OUTPUT.PUT_LINE('System command executed.  result = ' ||
                       command_code);
  RETURN command_code;
END execute_system;

FUNCTION execute_sql(command VARCHAR2,
                     timeout NUMBER DEFAULT 10)
RETURN NUMBER IS

  status       NUMBER;
  result       VARCHAR2(20);
  command_code NUMBER;
  pipe_name    VARCHAR2(30);

BEGIN
  pipe_name := DBMS_PIPE.UNIQUE_SESSION_NAME;

  DBMS_PIPE.PACK_MESSAGE('SQL');
  DBMS_PIPE.PACK_MESSAGE(pipe_name);
  DBMS_PIPE.PACK_MESSAGE(command);
  status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
```

```
    IF status <> 0 THEN
      RAISE_APPLICATION_ERROR(-20020,
        'Execute_sql: Error while sending.  Status = ' || status);
    END IF;

    status := DBMS_PIPE.RECEIVE_MESSAGE(pipe_name, timeout);

    IF status <> 0 THEN
      RAISE_APPLICATION_ERROR(-20021,
        'execute_sql: Error while receiving.
         Status = ' || status);
    END IF;

    DBMS_PIPE.UNPACK_MESSAGE(result);
    IF result <> 'done' THEN
      RAISE_APPLICATION_ERROR(-20022,
        'execute_sql: done not received.');
    END IF;

    DBMS_PIPE.UNPACK_MESSAGE(command_code);
    DBMS_OUTPUT.PUT_LINE
        ('SQL command executed.  sqlcode = ' || command_code);
    RETURN command_code;
  END execute_sql;

  PROCEDURE stop(timeout NUMBER DEFAULT 10) IS
    status NUMBER;
  BEGIN
    DBMS_PIPE.PACK_MESSAGE('STOP');
    status := DBMS_PIPE.SEND_MESSAGE('daemon', timeout);
    IF status <> 0 THEN
      RAISE_APPLICATION_ERROR(-20030,
        'stop: error while sending.  status = ' || status);
    END IF;
  END stop;
END daemon;
```

daemon.pc example This is the code for the Pro*C daemon. You must precompile this using the Pro*C Precompiler, Version 1.5.x or later. You must also specify the USERID and SQLCHECK options, as the example contains embedded PL/SQL code.

For example:

```
proc iname=daemon userid=scott/tiger sqlcheck=semantics
```

Then C-compile and link in the normal way.

```
#include <stdio.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

EXEC SQL BEGIN DECLARE SECTION;
  char *uid = "scott/tiger";
  int status;
  VARCHAR command[20];
  VARCHAR value[2000];
  VARCHAR return_name[30];
EXEC SQL END DECLARE SECTION;

void
connect_error()
{
  char msg_buffer[512];
  int msg_length;
  int buffer_size = 512;

  EXEC SQL WHENEVER SQLERROR CONTINUE;
  sqlglm(msg_buffer, &buffer_size, &msg_length);
  printf("Daemon error while connecting:\n");
  printf("%.*s\n", msg_length, msg_buffer);
  printf("Daemon quitting.\n");
  exit(1);
}

void
sql_error()
{
  char msg_buffer[512];
  int msg_length;
  int buffer_size = 512;

  EXEC SQL WHENEVER SQLERROR CONTINUE;
  sqlglm(msg_buffer, &buffer_size, &msg_length);
  printf("Daemon error while executing:\n");
  printf("%.*s\n", msg_length, msg_buffer);
  printf("Daemon continuing.\n");
}
main()
{
```

```
EXEC SQL WHENEVER SQLERROR DO connect_error();
EXEC SQL CONNECT :uid;
printf("Daemon connected.\n");

EXEC SQL WHENEVER SQLERROR DO sql_error();
printf("Daemon waiting...\n");
while (1) {
  EXEC SQL EXECUTE
    BEGIN
      :status := DBMS_PIPE.RECEIVE_MESSAGE('daemon');
      IF :status = 0 THEN
        DBMS_PIPE.UNPACK_MESSAGE(:command);
      END IF;
    END;
  END-EXEC;
  IF (status == 0)
  {
    command.arr[command.len] = '\0';
    IF (!strcmp((char *) command.arr, "STOP"))
    {
      printf("Daemon exiting.\n");
      break;
    }

    ELSE IF (!strcmp((char *) command.arr, "SYSTEM"))
    {
      EXEC SQL EXECUTE
        BEGIN
          DBMS_PIPE.UNPACK_MESSAGE(:return_name);
          DBMS_PIPE.UNPACK_MESSAGE(:value);
        END;
      END-EXEC;
      value.arr[value.len] = '\0';
      printf("Will execute system command '%s'\n", value.arr);

      status = system(value.arr);
      EXEC SQL EXECUTE
        BEGIN
          DBMS_PIPE.PACK_MESSAGE('done');
          DBMS_PIPE.PACK_MESSAGE(:status);
          :status := DBMS_PIPE.SEND_MESSAGE(:return_name);
        END;
      END-EXEC;

      IF (status)
```

```
    {
      printf
       ("Daemon error while responding to system command.");
      printf("  status: %d\n", status);
    }
  }
  ELSE IF (!strcmp((char *) command.arr, "SQL")) {
    EXEC SQL EXECUTE
      BEGIN
        DBMS_PIPE.UNPACK_MESSAGE(:return_name);
        DBMS_PIPE.UNPACK_MESSAGE(:value);
      END;
    END-EXEC;
    value.arr[value.len] = '\0';
    printf("Will execute sql command '%s'\n", value.arr);

    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL EXECUTE IMMEDIATE :value;
    status = sqlca.sqlcode;

    EXEC SQL WHENEVER SQLERROR DO sql_error();
    EXEC SQL EXECUTE
      BEGIN
        DBMS_PIPE.PACK_MESSAGE('done');
        DBMS_PIPE.PACK_MESSAGE(:status);
        :status := DBMS_PIPE.SEND_MESSAGE(:return_name);
      END;
    END-EXEC;

    IF (status)
    {
      printf("Daemon error while responding to sql command.");
      printf("  status: %d\n", status);
    }
  }
  ELSE
  {
    printf
      ("Daemon error: invalid command '%s' received.\n",
        command.arr);
  }
}
ELSE
{
  printf("Daemon error while waiting for signal.");
```

```
    printf("  status = %d\n", status);
  }
}
EXEC SQL COMMIT WORK RELEASE;
exit(0);
```

Example 3: External Service Interface

Put the user-written 3GL code into an OCI or Precompiler program. The program connects to the database and executes PL/SQL code to read its request from the pipe, computes the result, and then executes PL/SQL code to send the result on a pipe back to the requestor.

Below is an example of a stock service request. The recommended sequence for the arguments to pass on the pipe for all service requests is:

```
protocol_version      VARCHAR2        - '1', 10 bytes or less
returnpipe            VARCHAR2        - 30 bytes or less
service               VARCHAR2        - 30 bytes or less
arg1                  VARCHAR2/NUMBER/DATE
   ...
argn                  VARCHAR2/NUMBER/DATE
```

The recommended format for returning the result is:

```
success               VARCHAR2        - 'SUCCESS' if OK,
                                        otherwise error message
arg1                  VARCHAR2/NUMBER/DATE
   ...
argn                  VARCHAR2/NUMBER/DATE
```

The "stock price request server" would do, using OCI or PRO* (in pseudo-code):

```
<loop forever>
  BEGIN dbms_stock_server.get_request(:stocksymbol); END;
  <figure out price based on stocksymbol (probably from some radio
        signal), set error if can't find such a stock>
  BEGIN dbms_stock_server.return_price(:error, :price); END;
```

A client would do:

```
BEGIN :price := stock_request('YOURCOMPANY'); end;
```

The stored procedure, `dbms_stock_server`, which is called by the "stock price request server" above is:

```
CREATE OR REPLACE PACKAGE dbms_stock_server IS
```

```
  PROCEDURE get_request(symbol OUT VARCHAR2);
  PROCEDURE return_price(errormsg IN VARCHAR2, price IN VARCHAR2);
END;

CREATE OR REPLACE PACKAGE BODY dbms_stock_server IS
  returnpipe    VARCHAR2(30);

  PROCEDURE returnerror(reason VARCHAR2) IS
    s INTEGER;
  BEGIN
    dbms_pipe.pack_message(reason);
    s := dbms_pipe.send_message(returnpipe);
    IF s <> 0 THEN
      raise_application_error(-20000, 'Error:' || to_char(s) ||
        ' sending on pipe');
    END IF;
  END;

  PROCEDURE get_request(symbol OUT VARCHAR2) IS
    protocol_version VARCHAR2(10);
    s                INTEGER;
    service          VARCHAR2(30);
  BEGIN
    s := dbms_pipe.receive_message('stock_service');
    IF s <> 0 THEN
      raise_application_error(-20000, 'Error:' || to_char(s) ||
        'reading pipe');
    END IF;
    dbms_pipe.unpack_message(protocol_version);
    IF protocol_version <> '1' THEN
      raise_application_error(-20000, 'Bad protocol: ' ||
        protocol_version);
    END IF;
    dbms_pipe.unpack_message(returnpipe);
    dbms_pipe.unpack_message(service);
    IF service != 'getprice' THEN
      returnerror('Service ' || service || ' not supported');
    END IF;
    dbms_pipe.unpack_message(symbol);
  END;

  PROCEDURE return_price(errormsg in VARCHAR2, price in VARCHAR2) IS
    s INTEGER;
  BEGIN
    IF errormsg is NULL THEN
```

```
      dbms_pipe.pack_message('SUCCESS');
      dbms_pipe.pack_message(price);
    ELSE
      dbms_pipe.pack_message(errormsg);
    END IF;
    s := dbms_pipe.send_message(returnpipe);
    IF s <> 0 THEN
      raise_application_error(-20000, 'Error:'||to_char(s)||
        ' sending on pipe');
    END IF;
  END;
END;
```

The procedure called by the client is:

```
CREATE OR REPLACE FUNCTION stock_request (symbol VARCHAR2)
    RETURN VARCHAR2 IS
  s        INTEGER;
  price    VARCHAR2(20);
  errormsg VARCHAR2(512);
BEGIN
  dbms_pipe.pack_message('1');  -- protocol version
  dbms_pipe.pack_message(dbms_pipe.unique_session_name); -- return pipe
  dbms_pipe.pack_message('getprice');
  dbms_pipe.pack_message(symbol);
  s := dbms_pipe.send_message('stock_service');
  IF s <> 0 THEN
    raise_application_error(-20000, 'Error:'||to_char(s)||
      ' sending on pipe');
  END IF;
  s := dbms_pipe.receive_message(dbms_pipe.unique_session_name);
  IF s <> 0 THEN
    raise_application_error(-20000, 'Error:'||to_char(s)||
      ' receiving on pipe');
  END IF;
  dbms_pipe.unpack_message(errormsg);
  IF errormsg <> 'SUCCESS' THEN
    raise_application_error(-20000, errormsg);
  END IF;
  dbms_pipe.unpack_message(price);
  RETURN price;
END;
```

You would typically only grant execute on `dbms_stock_service` to the stock service application server, and would only grant execute on `stock_request` to those users allowed to use the service.

See Also: Chapter 2, "DBMS_ALERT"

29

DBMS_PROFILER

Oracle8*i* provides a Probe Profiler API to profile existing PL/SQL applications and to identify performance bottlenecks. The collected profiler (performance) data can be used for performance improvement efforts or for determining code coverage for PL/SQL applications. Code coverage data can be used by application developers to focus their incremental testing efforts.

The profiler API is implemented as a PL/SQL package, `DBMS_PROFILER`, that provides services for collecting and persistently storing PL/SQL profiler data.

Using DBMS_PROFILER

Improving application performance is an iterative process. Each iteration involves the following:

1. Exercising the application with one or more benchmark tests, with profiler data collection enabled.
2. Analyzing the profiler data, and identifying performance problems.
3. Fixing the problems.

To support this process, the PL/SQL profiler supports the notion of a *run*. A run involves running the application through benchmark tests with profiler data collection enabled. You can control the beginning and the end of the run by calling the `START_PROFILER` and `STOP_PROFILER` functions.

A typical session involves:

- Starting profiler data collection in session.
- Executing PL/SQL code for which profiler/code coverage data is required.
- Stopping profiler data collection.

> **Note:** Stopping data collection flushes out the collected data.

As the application executes, profiler data gets collected in memory data structures which last for the duration of the session. You can call the `FLUSH_DATA` function at intermediate points during the session to get incremental data and to free memory for allocated profiler data structures.

Flushing the collected data involves storing collected data to database tables. The tables should already exist in the profiler user's schema. The `PROFTAB.SQL` script is provided for creating the tables and other data structures required for persistently storing the profiler data.

See Also: "FLUSH_DATA function" on page 29-5.

Alternately, the tables can be created in one centrally administered schema, and profiler users can get access to the tables by creating public synonyms and granting `INSERT/SELECT` privileges on the tables and sequence.

Note: The collected profiler data is not automatically flushed at the end of the session. You must issue an explicit call to the FLUSH_DATA or the STOP_PROFILER function to flush the data at the end of the session.

Some PL/SQL operations, such as the very first execution of a PL/SQL unit, may involve I/O to catalog tables to load the byte code for the PL/SQL unit being executed. Also, it may take some time executing package initialization code the first time a package procedure or function is called. To avoid timing this overhead, you should *warm up* the database before collecting profile data. Warming up involves running the application once without gathering profiler data.

Collected Data

With the Probe Profiler API, you can generate profiling information for all named library units that are executed in a session. The profiler gathers information at the PL/SQL virtual machine level that includes the total number of times each line has been executed, the total amount of time that has been spent executing that line, and the minimum and maximum times that have been spent on a particular execution of that line.

Note: It is possible to infer the code coverage figures for PL/SQL units for which data has been collected.

The profiling information is stored in database tables. This enables the ad-hoc querying on the data: It lets you build customizable reports (summary reports, hottest lines, code coverage data, etc.) and analysis capabilities.

With Oracle8*i*, a sample textual report writer is provided with the PL/SQL demo scripts.

Requirements

DBMS_PROFILER must be installed as SYS.

Use the PROFLOAD.SQL script to load the PL/SQL Profiler packages.

Error Codes

A 0 return value from any function denotes successful completion; a non-zero return value denotes an error condition. The possible error returns are listed below:

- A subprogram was called with an incorrect parameter.

  ```
  error_param constant binary_integer := 1;
  ```

- Data flush operation failed. Check to see if the profiler tables have been created, are accessible, and that there is adequate space.

  ```
  error_io    constant binary_integer := 2;
  ```

- There is a mismatch between package and database implementation. You get this error if an incorrect version of the DBMS_PROFILER package is installed, and this version of the profiler package cannot work with this database version. The only possible recovery is to install the correct version of the package.

  ```
  error_version constant binary_integer := -1;
  ```

Summary of Subprograms

Table 29–1 DBMS_PROFILER Package Subprograms

Subprogram	Description
START_PROFILER function on page 29-4	Starts profiler data collection in session.
STOP_PROFILER function on page 29-5	Stops profiler data collection in session.
FLUSH_DATA function on page 29-5	Flushes profiler data collected in session.
GET_VERSION procedure on page 29-6	Gets the version of this API.
INTERNAL_VERSION_CHECK function on page 29-6	Verifies that this version of the DBMS_PROFILER package can work with the implementation in the database.

START_PROFILER function

This function starts profiler data collection in session.

Syntax

```
DBMS_PROFILER.START_PROFILER (
   run_comment IN VARCHAR2 := sysdate)
  RETURN BINARY_INTEGER;
```

Parameters

Table 29–2 START_PROFILER Function Parameters

Parameter	Description
run_comment	Each profiler run can be associated with a comment. For example, the comment could provide the name and version of the benchmark test that was used to collect data.

STOP_PROFILER function

This function stops profiler data collection in session.

This function has the side effect of flushing data collected so far in the session, and it signals the end of a run.

Syntax

```
DBMS_PROFILER.STOP_PROFILER
  RETURN BINARY_INTEGER;
```

Parameters

None.

FLUSH_DATA function

This function flushes profiler data collected in session. The data is flushed to database tables, which are expected to pre-exist.

> **Note:** Use the PROFTAB.SQL script to create the tables and other data structures required for persistently storing the profiler data.

Syntax

```
DBMS_PROFILER.FLUSH_DATA
  RETURN BINARY_INTEGER;
```

Parameters

None.

GET_VERSION procedure

This procedure gets the version of this API.

Syntax

```
DBMS_PROFILER.GET_VERSION (
   major OUT BINARY_INTEGER,
   minor OUT BINARY_INTEGER);
```

Parameters

Table 29–3 GET_VERSION Procedure Parameters

Parameter	Description
major	Major version of DBMS_PROFILER.
minor	Minor version of DBMS_PROFILER.

INTERNAL_VERSION_CHECK function

This function verifies that this version of the DBMS_PROFILER package can work with the implementation in the database.

Syntax

```
DBMS_PROFILER.INTERNAL_VERSION_CHECK
  RETURN BINARY_INTEGER;
```

Parameters

None.

30

DBMS_RANDOM

The DBMS_RANDOM package provides a built-in random number generator. It is faster than generators written in PL/SQL because it calls Oracle's internal random number generator.

Requirements

DBMS_RANDOM must be initialized prior to calling the random number generator. The generator produces 8 digit integers. If the initialization subprogram is not called, then the package raises an exception.

Summary of Subprograms

Table 30–1 DBMS_RANDOM Package Subprograms

Subprogram	Description
INITIALIZE procedure on page 30-2	Initializes the package with a seed value.
SEED procedure on page 30-3	Resets the seed.
RANDOM function on page 30-3	Gets the random number.
TERMINATE procedure on page 30-3	Closes the package.

INITIALIZE procedure

To use the package, first call the initialize subprogram with the seed to use.

Syntax

```
DBMS_RANDOM.INITIALIZE (
   seed  IN  BINARY_INTEGER);
```

Note: Use a seed that is sufficiently large, more than 5 digits. A single digit might not return sufficiently random numbers.

Parameters

Table 30–2 INITIALIZE Procedure Parameters

Parameter	Description
seed	Seed number used to generate a random number.

SEED procedure

This procedure resets the seed.

Syntax

```
DBMS_RANDOM.SEED (
   seed  IN  BINARY_INTEGER);
```

Parameters

Table 30–3 INITIALIZE Procedure Parameters

Parameter	Description
seed	Seed number used to generate a random number.

RANDOM function

This function gets the random number.

Syntax

```
DBMS_RANDOM.RANDOM
   RETURN BINARY_INTEGER;
```

Parameters

None.

Example

```
my_random_number := Random;
```

TERMINATE procedure

When you are finished with the package, call the `TERMINATE` procedure.

Syntax

```
DBMS_RANDOM.TERMINATE;
```

Parameters

None.

31

DBMS_RECTIFIER_DIFF

The DBMS_RECTIFIER_DIFF package contains APIs used to detect and resolve data inconsistencies between two replicated sites.

Summary of Subprograms

Table 31–1 DBMS_RECTIFIER_DIFF Package Subprograms

Subprogram	Description
DIFFERENCES procedure on page 31-2	Determines the differences between two tables.
RECTIFY procedure on page 31-5	Resolves the differences between two tables.

DIFFERENCES procedure

This procedure determines the differences between two tables.

Syntax

```
DBMS_RECTIFIER_DIFF.DIFFERENCES (
   sname1               IN  VARCHAR2,
   oname1               IN  VARCHAR2,
   reference_site       IN  VARCHAR2 := '',
   sname2               IN  VARCHAR2,
   oname2               IN  VARCHAR2,
   comparison_site      IN  VARCHAR2 := '',
   where_clause         IN  VARCHAR2 := '',
   { column_list        IN  VARCHAR2 := ''
   | array_columns      IN  dbms_utility.name_array, }
   missing_rows_sname   IN  VARCHAR2,
   missing_rows_oname1  IN  VARCHAR2,
   missing_rows_oname2  IN  VARCHAR2,
   missing_rows_site    IN  VARCHAR2 := '',
   max_missing          IN  INTEGER,
   commit_rows          IN  INTEGER := 500);
```

Note: This procedure is overloaded. The `column_list` and `array_columns` parameters are mutually exclusive.

Parameters

Table 31–2 DIFFERENCES Procedure Parameters

Parameter	Description
sname1	Name of the schema at REFERENCE_SITE.

Table 31–2 DIFFERENCES Procedure Parameters

Parameter	Description
oname1	Name of the table at REFERENCE_SITE.
reference_site	Name of the reference database site. The default, NULL, indicates the current site.
sname2	Name of the schema at COMPARISON_SITE.
oname2	Name of the table at COMPARISON_SITE.
comparison_site	Name of the comparison database site. The default, NULL, indicates the current site.
where_clause	Only rows satisfying this restriction are selected for comparison. The default, NULL, indicates the current site.
column_list	A comma-separated list of one or more column names being compared for the two tables. You must not have any white space before or after the comma. The default, NULL, indicates that all columns be compared.
array_columns	A PL/SQL table of column names being compared for the two tables. Indexing begins at 1, and the final element of the array must be NULL. If position 1 is NULL, then all columns are used.
missing_rows_sname	Name of the schema containing the tables with the missing rows.
missing_rows_oname1	Name of the table at MISSING_ROWS_SITE that stores information about the rows in the table at REFERENCE site missing from the table at COMPARISON site and the rows at COMPARISON site missing from the table at REFERENCE site.
missing_rows_oname2	Name of the table at MISSING_ROWS_SITE that stores about the missing rows. This table has three columns: the rowid of the row in the MISSING_ROWS_ONAME1 table, the name of the site at which the row is present, and the name of the site from which the row is absent.
missing_rows_site	Name of the site where the MISSING_ROWS_ONAME1 and MISSING_ROWS_ONAME2 tables are located. The default, NULL, indicates that the tables are located at the current site.

Table 31–2 DIFFERENCES Procedure Parameters

Parameter	Description
max_missing	Integer that refers to the maximum number of rows that should be inserted into the missing_rows_oname table. If more than max_missing number of rows is missing, then that many rows are inserted into missing_rows_oname, and the routine then returns normally without determining whether more rows are missing; this argument is useful in the cases that the fragments are so different that the missing rows table will have too many entries and there's no point in continuing. Raises exception badnumber if max_missing is less than 1 or NULL.
commit_rows	Maximum number of rows to insert to or delete from the reference or comparison table before a COMMIT occurs. By default, a COMMIT occurs after 500 inserts or 500 deletes. An empty string (' ') or NULL indicates that a COMMIT should only be issued after all rows for a single table have been inserted or deleted.

Exceptions

Table 31–3 DIFFERENCES Procedure Exceptions

Exception	Description
nosuchsite	Database site could not be found.
badnumber	COMMIT_ROWS parameter less than 1.
missingprimarykey	Column list must include primary key (or SET_COLUMNS equivalent).
badname	NULL or empty string for table or schema name.
cannotbenull	Parameter cannot be NULL.
notshapeequivalent	Tables being compared are not shape equivalent. Shape refers to the number of columns, their column names, and the column datatypes.
unknowncolumn	Column does not exist.
unsupportedtype	Type not supported.
dbms_repcat.commfailure	Remote site is inaccessible.
dbms_repcat.missingobject	Table does not exist.

Restrictions

The error ORA-00001 (Unique constraint violated) is issued when there are any unique or primary key constraints on the MISSING_ROWS_DATA table.

RECTIFY procedure

This procedure resolves the differences between two tables.

Syntax

```
DBMS_RECTIFIER_DIFF.RECTIFY (
   sname1               IN  VARCHAR2,
   oname1               IN  VARCHAR2,
   reference_site       IN  VARCHAR2 := '',
   sname2               IN  VARCHAR2,
   oname2               IN  VARCHAR2,
   comparison_site      IN  VARCHAR2 := '',
   { column_list        IN  VARCHAR2 := ''
   | array_columns      IN  dbms_utility.name_array, }
   missing_rows_sname   IN  VARCHAR2,
   missing_rows_oname1  IN  VARCHAR2,
   missing_rows_oname2  IN  VARCHAR2,
   missing_rows_site    IN  VARCHAR2 := '',
   commit_rows          IN  INTEGER := 500);
```

Note: This procedure is overloaded. The column_list and array_columns parameters are mutually exclusive.

Parameters

Table 31–4 RECTIFY Procedure Parameters

Parameter	Description
sname1	Name of the schema at REFERENCE_SITE.
oname1	Name of the table at REFERENCE_SITE.
reference_site	Name of the reference database site. The default, NULL, indicates the current site.
sname2	Name of the schema at COMPARISON_SITE.
oname2	Name of the table at COMPARISON_SITE.

Table 31–4 RECTIFY Procedure Parameters

Parameter	Description
comparison_site	Name of the comparison database site. The default, NULL, indicates the current site.
column_list	A comma-separated list of one or more column names being compared for the two tables. You must not have any white space before or after the comma. The default, NULL, indicates that all columns be compared.
array_columns	A PL/SQL table of column names being compared for the two tables. Indexing begins at 1, and the final element of the array must be NULL. If position 1 is NULL, then all columns are used.
missing_rows_sname	Name of the schema containing the tables with the missing rows.
missing_rows_oname1	Name of the table at MISSING_ROWS_SITE that stores information about the rows in the table at REFERENCE site missing from the table at COMPARISON site and the rows at COMPARISON site missing from the table at REFERENCE site.
missing_rows_oname2	Name of the table at MISSING_ROWS_SITE that stores about the missing rows. This table has three columns: the rowid of the row in the MISSING_ROWS_ONAME1 table, the name of the site at which the row is present, and the name of the site from which the row is absent.
missing_rows_site	Name of the site where the MISSING_ROWS_ONAME1 and MISSING_ROWS_ONAME2 tables are located. The default, NULL, indicates that the tables are located at the current site.
commit_rows	Maximum number of rows to insert to or delete from the reference or comparison table before a COMMIT occurs. By default, a COMMIT occurs after 500 inserts or 500 deletes. An empty string (' ') or NULL indicates that a COMMIT should only be issued after all rows for a single table have been inserted or deleted.

Exceptions

Table 31–5 RECTIFY Procedure Exceptions

Exception	Description
nosuchsite	Database site could not be found.
badnumber	COMMIT_ROWS parameter less than 1.
badname	NULL or empty string for table or schema name.
dbms_repcat.commfailure	Remote site is inaccessible.
dbms_repcat.missingobject	Table does not exist.

32

DBMS_REFRESH

DBMS_REFRESH enables you to create groups of snapshots that can be refreshed together to a transactionally consistent point in time.

Summary of Subprograms

Table 32–1 DBMS_REFRESH Package Subprograms

Subprogram	Description
ADD procedure on page 32-2	Adds snapshots to a refresh group.
CHANGE procedure on page 32-3	Changes the refresh interval for a snapshot group.
DESTROY procedure on page 32-5	Removes all of the snapshots from a refresh group and deletes the refresh group.
MAKE procedure on page 32-5	Specifies the members of a refresh group and the time interval used to determine when the members of this group should be refreshed.
REFRESH procedure on page 32-8	Manually refreshes a refresh group.
SUBTRACT procedure on page 32-8	Removes snapshots from a refresh group.

ADD procedure

This procedure adds snapshots to a refresh group.

> **See Also:** For additional information, see "Add Objects To Refresh Group" in the *Oracle8i Replication API Reference*. Also see "Snapshot Concepts & see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REFRESH.ADD (
   name     IN VARCHAR2,
   { list   IN VARCHAR2
   | tab    IN DBMS_UTILITY.UNCL_ARRAY, }
   lax      IN BOOLEAN := FALSE);
```

Note: This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

Table 32–2 ADD Procedures Parameters

Parameter	Description
`name`	Name of the refresh group to which you want to add members.
`list`	Comma-separated list of snapshots that you want to add to the refresh group. (Synonyms are not supported.)
`tab`	Instead of a comma-separated list, you can supply a PL/SQL table of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a snapshot. The first snapshot should be in position 1. The last position must be `NULL`.
`lax`	A snapshot can belong to only one refresh group at a time. If you are moving a snapshot from one group to another, then you must set the `lax` flag to `TRUE` to succeed. Oracle then automatically removes the snapshot from the other refresh group and updates its refresh interval to be that of its new group. Otherwise, the call to `ADD` generates an error message.

CHANGE procedure

This procedure changes the refresh interval for a snapshot group.

See Also: For additional information, see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REFRESH.CHANGE (
   name                 IN VARCHAR2,
   next_date            IN DATE           := NULL,
   interval             IN VARCHAR2       := NULL,
   implicit_destroy     IN BOOLEAN        := NULL,
   rollback_seg         IN VARCHAR2       := NULL,
   push_deferred_rpc    IN BOOLEAN        := NULL,
   refresh_after_errors IN BOOLEAN        := NULL,
   purge_option         IN BINARY_INTEGER := NULL,
   parallelism          IN BINARY_INTEGER := NULL,
   heap_size            IN BINARY_INTEGER := NULL);
```

Parameters

Table 32–3 CHANGE Procedures Parameters

Parameter	Description
name	Name of the refresh group for which you want to alter the refresh interval.
next_date	Next date that you want a refresh to occur. By default, this date remains unchanged.
interval	Function used to calculate the next time to refresh the snapshots in the group. This interval is evaluated immediately before the refresh. Thus, you should select an interval that is greater than the time it takes to perform a refresh. By default, the interval remains unchanged.
implicit_destroy	Allows you to reset the value of the implicit_destroy flag. If this flag is set, then Oracle automatically deletes the group if it no longer contains any members. By default, this flag remains unchanged.
rollback_seg	Allows you to change the rollback segment used. By default, the rollback segment remains unchanged. To reset this parameter to use the default rollback segment, specify NULL, including the quotes. Specifying NULL without quotes indicates that you do not want to change the rollback segment currently being used.
push_deferred_rpc	Used by updatable snapshots only. Set this parameter to TRUE if you want to push changes from the snapshot to its associated master before refreshing the snapshot. Otherwise, these changes may appear to be temporarily lost. By default, this flag remains unchanged
refresh_after_ errors	Used by updatable snapshots only. Set this parameter to TRUE if you want the refresh to proceed even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master. By default, this flag remains unchanged.
purge_option	If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), then 0 = don't purge; 1 = lazy (default); 2 = aggressive. In most cases, *lazy* purge is the optimal setting. Set purge to *aggressive* to trim back the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, then set purge to *don't purge* and occasionally execute PUSH with purge set to *aggressive* to reduce the queue.

Table 32–3 CHANGE Procedures Parameters

Parameter	Description
parallelism	0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set the parameter unless so directed by Oracle Worldwide Support.

DESTROY procedure

This procedure removes all of the snapshots from a refresh group and delete the refresh group.

> **See Also:** For additional information, see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REFRESH.DESTROY (
   name   IN   VARCHAR2);
```

Parameters

Table 32–4 DESTROY Procedure Parameters

Parameter	Description
name	Name of the refresh group that you want to destroy.

MAKE procedure

This procedure specifies the members of a refresh group and the time interval used to determine when the members of this group should be refreshed.

> **See Also:** For additional information, see "Create Refresh Group" in *Oracle8i Replication API Reference*. Also see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REFRESH.MAKE (
   name                 IN     VARCHAR2,
   { list               IN     VARCHAR2
   | tab                IN     DBMS_UTILITY.UNCL_ARRAY,}
   next_date            IN     DATE,
   interval             IN     VARCHAR2,
   implicit_destroy     IN     BOOLEAN         := FALSE,
   lax                  IN     BOOLEAN         := FALSE,
   job                  IN     BINARY INTEGER  := 0,
   rollback_seg         IN     VARCHAR2        := NULL,
   push_deferred_rpc    IN     BOOLEAN         := TRUE,
   refresh_after_errors IN     BOOLEAN         := FALSE,
   purge_option         IN     BINARY_INTEGER  := NULL,
   parallelism          IN     BINARY_INTEGER  := NULL,
   heap_size            IN     BINARY_INTEGER  := NULL);
```

Note: This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Table 32–5 MAKE Procedure Parameters

Parameter	Description
`name`	Unique name used to identify the refresh group. Refresh groups must follow the same naming conventions as tables.
`list`	Comma-separated list of snapshots that you want to refresh. (Synonyms are not supported.) These snapshots can be located in different schemas and have different master tables; however, all of the listed snapshots must be in your current database.
`tab`	Instead of a comma separated list, you can supply a PL/SQL table of names of snapshots that you want to refresh using the datatype `DBMS_UTILITY.UNCL_ARRAY`. If the table contains the names of N snapshots, then the first snapshot should be in position 1 and the N + 1 position should be set to `NULL`.
`next_date`	Next date that you want a refresh to occur.

Table 32–5 MAKE Procedure Parameters

Parameter	Description
interval	Function used to calculate the next time to refresh the snapshots in the group. This field is used with the NEXT_DATE value. For example, if you specify NEXT_DAY(SYSDATE+1, "MONDAY") as your interval, and if your NEXT_DATE evaluates to Monday, then Oracle refreshes the snapshots every Monday. This interval is evaluated immediately before the refresh. Thus, you should select an interval that is greater than the time it takes to perform a refresh.
implicit_destroy	Set this to TRUE if you want to delete the refresh group automatically when it no longer contains any members. Oracle checks this flag only when you call the SUBTRACT procedure. That is, setting this flag still allows you to create an empty refresh group.
lax	A snapshot can belong to only one refresh group at a time. If you are moving a snapshot from an existing group to a new refresh group, then you must set this to TRUE to succeed. Oracle then automatically removes the snapshot from the other refresh group and updates its refresh interval to be that of its new group. Otherwise, the call to MAKE generates an error message.
job	Needed by the Import utility. Use the default value, 0.
rollback_seg	Name of the rollback segment to use while refreshing snapshots. The default, NULL, uses the default rollback segment.
push_deferred_rpc	Used by updatable snapshots only. Use the default value, TRUE, if you want to push changes from the snapshot to its associated master before refreshing the snapshot. Otherwise, these changes may appear to be temporarily lost.
refresh_after_errors	Used by updatable snapshots only. Set this to 0 if you want the refresh to proceed even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master.
purge_option	If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), then 0 = don't purge; 1 = lazy (default); 2 = aggressive. In most cases, *lazy* purge is the optimal setting. Set purge to *aggressive* to trim back the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, then set purge to *don't purge* and occasionally execute PUSH with purge set to *aggressive* to reduce the queue.

Table 32–5 MAKE Procedure Parameters

Parameter	Description
parallelism	0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set this unless so directed by Oracle Worldwide Support.

REFRESH procedure

This procedure manually refreshes a refresh group.

> **See Also:** For additional information, see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REFRESH.REFRESH (
   name    IN    VARCHAR2);
```

Table 32–6 REFRESH Procedure Parameters

Parameter	Description
name	Name of the refresh group that you want to refresh manually.

SUBTRACT procedure

This procedure removes snapshots from a refresh group.

> **See Also:** For additional information, see "Snapshot Concepts & Architecture" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REFRESH.SUBTRACT (
   name       IN    VARCHAR2,
   { list     IN    VARCHAR2
   | tab      IN     DBMS_UTILITY.UNCL_ARRAY, }
   lax        IN     BOOLEAN := FALSE);
```

Note: This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

Table 32–7 SUBTRACT Procedure Parameters

Parameter	Description
`name`	Name of the refresh group from which you want to remove members.
`list`	Comma-separated list of snapshots that you want to remove from the refresh group. (Synonyms are not supported.) These snapshots can be located in different schemas and have different master tables; however, all of the listed snapshots must be in your current database.
`tab`	Instead of a comma-separated list, you can supply a PL/SQL table of names of snapshots that you want to refresh using the datatype `DBMS_UTILITY.UNCL_ARRAY`. If the table contains the names of N snapshots, then the first snapshot should be in position 1 and the N+1 position should be set to `NULL`.
`lax`	Set this to `FALSE` if you want Oracle to generate an error message if the snapshot you are attempting to remove is not a member of the refresh group.

33

DBMS_REPAIR

DBMS_REPAIR contains data corruption repair procedures that enable you to detect and repair corrupt blocks in tables and indexes. You can address corruptions where possible and continue to use objects while you attempt to rebuild or repair them.

Note: The DBMS_REPAIR package is intended for use by database administrators only. It is not intended for use by application developers.

See Also: For detailed information about using the DBMS_REPAIR package, see *Oracle8i Administrator's Guide.*

Security

The package is owned by SYS. Execution privilege is not granted to other users.

Enumeration Types

The DBMS_REPAIR package defines several enumerated constants that should be used for specifying parameter values. Enumerated constants must be prefixed with the package name. For example, DBMS_REPAIR.TABLE_OBJECT.

Table 33–1 lists the parameters and the enumerated constants.

Table 33–1 DBMS_REPAIR Enumeration Types

Parameter	Constant
object_type	TABLE_OBJECT, INDEX_OBJECT, CLUSTER_OBJECT
action	CREATE_ACTION, DROP_ACTION, PURGE_ACTION
table_type	REPAIR_TABLE, ORPHAN_TABLE
flags	SKIP_FLAG, NOSKIP_FLAG

Note: The default table_name will be REPAIR_TABLE when table_type is REPAIR_TABLE, and will be ORPHAN_KEY_TABLE when table_type is ORPHAN_TABLE.

Exceptions

Table 33–2 DBMS_REPAIR Exceptions

Exception	Description	Action
942	Reported by DBMS_REPAIR.ADMIN_TABLES during a DROP_ACTION when the specified table doesn't exist.	
955	Reported by DBMS_REPAIR.CREATE_ACTION when the specified table already exists.	

Table 33–2 DBMS_REPAIR Exceptions

Exception	Description	Action
24120	An invalid parameter was passed to the specified DBMS_REPAIR procedure.	Specify a valid parameter value or use the parameter's default.
24122	An incorrect block range was specified.	Specify correct values for the BLOCK_START and BLOCK_END parameters.
24123	An attempt was made to use the specified feature, but the feature is not yet implemented.	Do not attempt to use the feature.
24124	An invalid ACTION parameter was specified.	Specify CREATE_ACTION, PURGE_ACTION or DROP_ACTION for the ACTION parameter.
24125	An attempt was made to fix corrupt blocks on an object that has been dropped or truncated since DBMS_REPAIR.CHECK_OBJECT was run.	Use DBMS_REPAIR.ADMIN_TABLES to purge the repair table and run DBMS_REPAIR.CHECK_OBJECT to determine whether there are any corrupt blocks to be fixed.
24127	TABLESPACE parameter specified with an ACTION other than CREATE_ACTION.	Do not specify TABLESPACE when performing actions other than CREATE_ACTION.
24128	A partition name was specified for an object that is not partitioned.	Specify a partition name only if the object is partitioned.
24129	An attempt was made to pass a table name parameter without the specified prefix.	Pass a valid table name parameter.
24130	An attempt was made to specify a repair or orphan table that does not exist.	Specify a valid table name parameter.
24131	An attempt was made to specify a repair or orphan table that does not have a correct definition.	Specify a table name that refers to a properly created table.
24132	An attempt was made to specify a table name is greater than 30 characters long.	Specify a valid table name parameter.

Summary of Subprograms

Table 33–3 DBMS_REPAIR Package Subprograms

Subprogram	Description
ADMIN_TABLES procedure on page 33-4	Provides administrative functions for the DBMS_REPAIR package repair and orphan key tables, including create, purge, and drop functions.
CHECK_OBJECT procedure on page 33-5	Detects and reports corruptions in a table or index.
DUMP_ORPHAN_KEYS procedure on page 33-7	Reports on index entries that point to rows in corrupt data blocks.
FIX_CORRUPT_BLOCKS procedure on page 33-9	Marks blocks software corrupt that have been previously detected as corrupt by CHECK_OBJECT.
REBUILD_FREELISTS procedure on page 33-10	Rebuilds an object's freelists.
SKIP_CORRUPT_BLOCKS procedure on page 33-11	Sets whether to ignore blocks marked corrupt during table and index scans or to report ORA-1578 when blocks marked corrupt are encountered.

ADMIN_TABLES procedure

This procedure provides administrative functions for the DBMS_REPAIR package repair and orphan key tables.

Syntax

```
DBMS_REPAIR.ADMIN_TABLES (
   table_name IN   VARCHAR2,
   table_type IN   BINARY_INTEGER,
   action     IN   BINARY_INTEGER,
   tablespace IN   VARCHAR2        DEFAULT NULL);
```

Parameters

Table 33–4 ADMIN_TABLES Procedure Parameters

Parameter	Description
table_name	Name of the table to be processed. Defaults to ORPHAN_KEY_TABLE or REPAIR_TABLE based on the specified table_type. When specified, the table name must have the appropriate prefix: ORPHAN_ or REPAIR_.
table_type	Type of table; must be either ORPHAN_TABLE or REPAIR_TABLE. See "Enumeration Types" on page 33-2.
action	Indicates what administrative action to perform. Must be either CREATE_ACTION, PURGE_ACTION, or DROP_ACTION. If the table already exists, and if CREATE_ACTION is specified, then an error is returned. PURGE_ACTION indicates to delete all rows in the table that are associated with non-existent objects. If the table does not exist, and if DROP_ACTION is specified, then an error is returned. When CREATE_ACTION and DROP_ACTION are specified, an associated view named DBA_<table_name> is created and dropped respectively. The view is defined so that rows associated with non-existent objects are eliminated. Created in the SYS schema. See "Enumeration Types" on page 33-2.
tablespace	Indicates the tablespace to use when creating a table. By default, the SYS default tablespace is used. An error is returned if the tablespace is specified and if the action is not CREATE_ACTION.

CHECK_OBJECT procedure

This procedure checks the specified objects and populates the repair table with information about corruptions and repair directives.

Validation consists of block checking all blocks in the object. You may optionally specify a DBA range, partition name, or subpartition name when you want to check a portion of an object.

Syntax

```
DBMS_REPAIR.CHECK_OBJECT (
   schema_name       IN  VARCHAR2,
   object_name       IN  VARCHAR2,
   partition_name    IN  VARCHAR2       DEFAULT NULL,
   object_type       IN  BINARY_INTEGER DEFAULT TABLE_OBJECT,
   repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
   flags             IN  BINARY_INTEGER DEFAULT NULL,
   relative_fno      IN  BINARY_INTEGER DEFAULT NULL,
   block_start       IN  BINARY_INTEGER DEFAULT NULL,
   block_end         IN  BINARY_INTEGER DEFAULT NULL,
   corrupt_count     OUT BINARY_INTEGER);
```

Parameters

Table 33–5 CHECK_OBJECT Procedure Parameters

Parameter	Description
schema_name	Schema name of the object to be checked.
object_name	Name of the table or index to be checked.
partition_name	Partition or subpartition name to be checked. If this is a partitioned object, and if partition_name is not specified, then all partitions and subpartitions are checked. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are checked.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See "Enumeration Types" on page 33-2.
repair_table_name	Name of the repair table to be populated. The table must exist in the SYS schema. Use the admin_tables procedure to create a repair table. The default name is REPAIR_TABLE.
flags	Reserved for future use.
relative_fno	Relative file number: Used when specifying a block range.
block_start	First block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition.

Table 33–5 CHECK_OBJECT Procedure Parameters

Parameter	Description
block_end	Last block to process if specifying a block range. May be specified only if the object is a single table, partition, or subpartition. If only one of block_start or block_end is specified, then the other defaults to the first or last block in the file respectively.
corrupt_count	Number of corruptions reported.

DUMP_ORPHAN_KEYS procedure

This procedure reports on index entries that point to rows in corrupt data blocks. For each such index entry encountered, a row is inserted into the specified orphan table.

If the repair table is specified, then any corrupt blocks associated with the base table are handled in addition to all data blocks that are marked software corrupt. Otherwise, only blocks that are marked corrupt are handled.

This information may be useful for rebuilding lost rows in the table and for diagnostic purposes.

Syntax

```
DBMS_REPAIR.DUMP_ORPHAN_KEYS (
   schema_name       IN  VARCHAR2,
   object_name       IN  VARCHAR2,
   partition_name    IN  VARCHAR2       DEFAULT NULL,
   object_type       IN  BINARY_INTEGER DEFAULT INDEX_OBJECT,
   repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
   orphan_table_name IN  VARCHAR2       DEFAULT 'ORPHAN_KEYS_TABLE',
   flags             IN  BINARY_INTEGER DEFAULT NULL,
   key_count         OUT BINARY_INTEGER);
```

Parameters

Table 33–6 DUMP_ORPHAN_KEYS Procedure Parameters

Parameter	Description
schema_name	Schema name.
object_name	Object name.

Table 33–6 DUMP_ORPHAN_KEYS Procedure Parameters

Parameter	Description
partition_name	Partition or subpartition name to be processed. If this is a partitioned object, and if partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. The default is INDEX_OBJECT See "Enumeration Types" on page 33-2.
repair_table_name	Name of the repair table that has information regarding corrupt blocks in the base table. The specified table must exist in the SYS schema. The admin_tables procedure is used to create the table.
orphan_table_name	Name of the orphan key table to populate with information regarding each index entry that refers to a row in a corrupt data block. The specified table must exist in the SYS schema. The admin_tables procedure is used to create the table.
flags	Reserved for future use.
key_count	Number of index entries processed.

FIX_CORRUPT_BLOCKS procedure

This procedure fixes the corrupt blocks in specified objects based on information in the repair table that was previously generated by the `check_object` procedure.

Prior to effecting any change to a block, the block is checked to ensure the block is still corrupt. Corrupt blocks are repaired by marking the block software corrupt. When a repair is effected, the associated row in the repair table is updated with a fix timestamp.

Syntax

```
DBMS_REPAIR.FIX_CORRUPT_BLOCKS (
   schema_name       IN  VARCHAR2,
   object_name       IN  VARCHAR2,
   partition_name    IN  VARCHAR2       DEFAULT NULL,
   object_type       IN  BINARY_INTEGER DEFAULT TABLE_OBJECT,
   repair_table_name IN  VARCHAR2       DEFAULT 'REPAIR_TABLE',
   flags             IN  BINARY_INTEGER DEFAULT NULL,
   fix_count         OUT BINARY_INTEGER);
```

Parameters

Table 33–7 FIX_CORRUPT_BLOCKS Procedure Parameters

Parameter	Description
schema_name	Schema name.
object_name	Name of the object with corrupt blocks to be fixed.
partition_name	Partition or subpartition name to be processed. If this is a partitioned object, and if `partition_name` is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. This must be either `TABLE_OBJECT` (default) or `INDEX_OBJECT`. See "Enumeration Types" on page 33-2.
repair_table_name	Name of the repair table with the repair directives. Must exist in the SYS schema.
flags	Reserved for future use.
fix_count	Number of blocks fixed.

REBUILD_FREELISTS procedure

This procedure rebuilds the freelists for the specified object. All free blocks are placed on the master freelist. All other freelists are zeroed.

If the object has multiple freelist groups, then the free blocks are distributed among all freelists, allocating to the different groups in round-robin fashion.

Syntax

```
DBMS_REPAIR.REBUILD_FREELISTS (
   schema_name    IN VARCHAR2,
   partition_name IN VARCHAR2       DEFAULT NULL,
   object_type    IN BINARY_INTEGER DEFAULT TABLE_OBJECT);
```

Parameters

Table 33–8 REBUILD_FREELISTS Procedure Parameters

Parameter	Description
schema_name	Schema name.
object_name	Name of the object whose freelists are to be rebuilt.
partition_name	Partition or subpartition name whose freelists are to be rebuilt. If this is a partitioned object, and partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or INDEX_OBJECT. See "Enumeration Types" on page 33-2.

SKIP_CORRUPT_BLOCKS procedure

This procedure enables or disables the skipping of corrupt blocks during index and table scans of the specified object.

When the object is a table, skip applies to the table and its indexes. When the object is a cluster, it applies to all of the tables in the cluster, and their respective indexes.

Syntax

```
DBMS_REPAIR.SKIP_CORRUPT_BLOCKS (
   schema_name IN VARCHAR2,
   object_name IN VARCHAR2,
   object_type IN BINARY_INTEGER DEFAULT TABLE_OBJECT,
   flags       IN BINARY_INTEGER DEFAULT SKIP_FLAG);
```

Parameters

Table 33–9 SKIP_CORRUPT_BLOCKS Procedure Parameters

Parameter	Description
schema_name	Schema name of the object to be processed.
object_name	Name of the object.
partition_name (optional)	Partition or subpartition name to be processed. If this is a partitioned object, and if partition_name is not specified, then all partitions and subpartitions are processed. If this is a partitioned object, and if the specified partition contains subpartitions, then all subpartitions are processed.
object_type	Type of the object to be processed. This must be either TABLE_OBJECT (default) or CLUSTER_OBJECT. See "Enumeration Types" on page 33-2.
flags	If SKIP_FLAG is specified, then it turns on the skip of software corrupt blocks for the object during index and table scans. If NOSKIP_FLAG is specified, then scans that encounter software corrupt blocks return an ORA-1578. See "Enumeration Types" on page 33-2.

34

DBMS_REPCAT

DBMS_REPCAT provides routines to administer and update the replication catalog and environment.

Summary of Subprograms

Table 34–1 DBMS_REPCAT Package Subprograms

Subprogram	Description
ADD_GROUPED_COLUMN procedure on page 34-5	Adds members to an existing column group.
ADD_MASTER_DATABASE procedure on page 34-7	Adds another master site to your replicated environment.
ADD_PRIORITY_datatype procedure on page 34-8	Adds a member to a priority group.
ADD_SITE_PRIORITY_SITE procedure on page 34-9	Adds a new site to a site priority group.
ADD_conflicttype_RESOLUTION procedure on page 34-10	Designates a method for resolving an update, delete, or uniqueness conflict.
ALTER_MASTER_PROPAGATION procedure on page 34-14	Alters the propagation method for a given object group at a given master site.
ALTER_MASTER_REPOBJECT procedure on page 34-16	Alters an object in your replicated environment.
ALTER_PRIORITY procedure on page 34-17	Alters the priority level associated with a given priority group member.
ALTER_PRIORITY_datatype procedure on page 34-18	Alters the value of a member in a priority group.
ALTER_SITE_PRIORITY procedure on page 34-20	Alters the priority level associated with a given site.
ALTER_SITE_PRIORITY_SITE procedure on page 34-21	Alters the site associated with a given priority level.
ALTER_SNAPSHOT_PROPAGATION procedure on page 34-22	Alters the propagation method for a given object group at the current snapshot site.
CANCEL_STATISTICS procedure on page 34-23	Stops collecting statistics about the successful resolution of update, uniqueness, and delete conflicts for a table.
COMMENT_ON_COLUMN_GROUP procedure on page 34-24	Updates the comment field in the RepColumn_Group view for a column group.
COMMENT_ON_PRIORITY_ GROUP/PRIORITY procedure on page 34-25	Updates the comment field in the REPPRIORITY_GROUP view for a (site) priority group.

Table 34–1 DBMS_REPCAT Package Subprograms

Subprogram	Description
COMMENT_ON_REPGROUP procedure on page 34-26	Updates the comment field in the REPGROUP view for a replicated object group.
COMMENT_ON_REPSITES procedure on page 34-27	Updates the comment field in the RepSite view for a replicated site.
COMMENT_ON_REPOBJECT procedure on page 34-28	Updates the comment field in the RepObject view for a replicated object.
COMMENT_ON_conflicttype_ RESOLUTION procedure on page 34-29	Updates the comment field in the RepResolution view for a conflict resolution routine.
CREATE_MASTER_REPGROUP procedure on page 34-33	Creates a new, empty, quiesced master replication object group.
CREATE_MASTER_REPOBJECT procedure on page 34-34	Indicates that an object is a replicated object.
CREATE_SNAPSHOT_REPGROUP procedure on page 34-37	Creates a new, empty snapshot replication object group in your local database.
CREATE_SNAPSHOT_REPOBJECT procedure on page 34-38	Adds a replicated object to your snapshot site.
DEFINE_COLUMN_GROUP procedure on page 34-40	Creates an empty column group
DEFINE_PRIORITY_GROUP procedure on page 34-41	Creates a new priority group for a replicated object group.
DEFINE_SITE_PRIORITY procedure on page 34-43	Creates a new site priority group for a replicated object group.
DO_DEFERRED_REPCAT_ADMIN procedure on page 34-44	Executes the local outstanding deferred administrative procedures for the given replicated object group at the current master site, or for all master sites.
DROP_COLUMN_GROUP procedure on page 34-44	Drops a column group.
DROP_GROUPED_COLUMN procedure on page 34-45	Removes members from a column group.
DROP_MASTER_REPGROUP procedure on page 34-46	Drops a replicated object group from your current site.

Table 34–1 DBMS_REPCAT Package Subprograms

Subprogram	Description
DROP_MASTER_REPOBJECT procedure on page 34-48	Drops a replicated object from a replicated object group.
DROP_PRIORITY procedure on page 34-49	Drops a member of a priority group by priority level.
DROP_PRIORITY_GROUP procedure on page 34-49	Drops a priority group for a given replicated object group.
DROP_PRIORITY_datatype procedure on page 34-50	Drops a member of a priority group by value.
DROP_SITE_PRIORITY procedure on page 34-52	Drops a site priority group for a given replicated object group.
DROP_SITE_PRIORITY_SITE procedure on page 34-52	Drops a given site, by name, from a site priority group.
DROP_SNAPSHOT_REPGROUP procedure on page 34-53	Drops a snapshot site from your replicated environment.
DROP_SNAPSHOT_REPOBJECT procedure on page 34-54	Drops a replicated object from a snapshot site.
DROP_conflicttype_ RESOLUTION procedure on page 34-55	Drops an update, delete, or uniqueness conflict resolution routine.
EXECUTE_DDL procedure on page 34-57	Supplies DDL that you want to have executed at each master site.
GENERATE_REPLICATION_ SUPPORT procedure on page 34-58	Generates the triggers, packages, and procedures needed to support replication.
GENERATE_SNAPSHOT_SUPPORT procedure on page 34-60	Activates triggers and generate packages needed to support the replication of updatable snapshots or procedural replication.
MAKE_COLUMN_GROUP procedure on page 34-61	Creates a new column group with one or more members.
PURGE_MASTER_LOG procedure on page 34-63	Removes local messages in the RepCatLog associated with a given identification number, source, or replicated object group.
PURGE_STATISTICS procedure on page 63	Removes information from the RepResolution_ Statistics view.

Table 34–1 DBMS_REPCAT Package Subprograms

Subprogram	Description
REFRESH_SNAPSHOT_REPGROUP procedure on page 34-64	Refreshes a snapshot site object group with the most recent data from its associated master site.
REGISTER_SNAPSHOT_REPGROUP procedure on page 34-65	Facilitates the administration of snapshots at their respective master sites by inserting / modifying / deleting from repcat_repsite.
REGISTER_STATISTICS procedure on page 34-66	Collects information about the successful resolution of update, delete and uniqueness conflicts for a table.
RELOCATE_MASTERDEF procedure on page 67	Changes your master definition site to another master site in your replicated environment.
REMOVE_MASTER_DATABASES procedure on page 34-69	Removes one or more master databases from a replicated environment.
REPCAT_IMPORT_CHECK procedure on page 34-70	Ensures that the objects in the replicated object group have the appropriate object identifiers and status values after you perform an export / import of a replicated object or an object used by the advanced replication facility.
RESUME_MASTER_ACTIVITY procedure on page 34-71	Resumes normal replication activity after quiescing a replicated environment.
SUSPEND_MASTER_ACTIVITY procedure on page 34-75	Suspends replication activity for an object group
SWITCH_SNAPSHOT_MASTER procedure on page 34-76	Changes the master database of a snapshot replicated object group to another master site.
UNREGISTER_SNAPSHOT_ REPGROUP procedure on page 34-77	Facilitates the administration of snapshots at their respective master sites by inserting / modifying / deleting from repcat$_repsite.
VALIDATE function on page 34-78	Validates the correctness of key conditions of a multiple master replication environment.
WAIT_MASTER_LOG procedure on page 34-80	Determines whether changes that were asynchronously propagated to a master site have been applied.

ADD_GROUPED_COLUMN procedure

This procedure adds members to an existing column group. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.ADD_GROUPED_COLUMN (
   sname                 IN   VARCHAR2,
   oname                 IN   VARCHAR2,
   column_group          IN   VARCHAR2,
   list_of_column_names  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S);
```

Parameters

Table 34–2 ADD_GROUPED_COLUMN Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table with which the column group is associated.
column_group	Name of the column group to which you are adding members.
list_of_column_names	Names of the columns that you are adding to the designated column group. This can either be a comma-separated list or a PL/SQL table of column names. The PL/SQL table must be of type `dbms_repcat.varchar2s`. Use the single value '*' to create a column group that contains all of the columns in your table.

Exceptions

Table 34–3 ADD_GROUPED_COLUMN Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingobject	Given table does not exist.
missinggroup	Given column group does not exist.
missingcolumn	Given column does not exist in the designated table.
duplicatecolumn	Given column is already a member of another column group.
missingschema	Given schema does not exist.
notquiesced	Object group that the given table belongs to is not quiesced.

ADD_MASTER_DATABASE procedure

This procedure adds another master site to your replicated environment. This procedure regenerates all the triggers and their associated packages at existing master sites. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.ADD_MASTER_DATABASE (
   gname                IN   VARCHAR2,
   master               IN   VARCHAR2,
   use_existing_objects IN   BOOLEAN  := TRUE,
   copy_rows            IN   BOOLEAN  := TRUE,
   comment              IN   VARCHAR2 := '',
   propagation_mode     IN   VARCHAR2 := 'ASYNCHRONOUS',
   fname                IN   VARCHAR2 := NULL);
```

Parameters

Table 34–4 ADD_MASTER_DATABASE Procedure Parameters

Parameter	Description
gname	Name of the object group being replicated. This object group must already exist at the master definition site.
master	Fully qualified database name of the new master database.
use_existing_objects	Indicate TRUE if you want to reuse any objects of the same type and shape that already exist in the schema at the new master site. See "Using Multimaster Replication" in the *Oracle8i Replication* manual for more information on how these changes are applied.
copy_rows	Indicate TRUE if you want the initial contents of a table at the new master site to match the contents of the table at the master definition site.
comment	This is added to the MASTER_COMMENT field of the RepSites view.
propagation_mode	Method of forwarding changes to and receiving changes from new master database. Accepted values are SYNCHRONOUS and ASYNCHRONOUS.
fname	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 34–5 ADD_MASTER_DATABASE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Replicated object group has not been suspended.
missingrepgroup	Object group does not exist at the given database site.
commfailure	New master is not accessible.
typefailure	An incorrect propagation mode was specified.
notcompat	Compatibility mode must be 7.3.0.0 or greater.
duplrepgrp	Master site already exists.

ADD_PRIORITY_*datatype* procedure

This procedure adds a member to a priority group. You must call this procedure from the master definition site. The procedure that you must call is determined by the datatype of your `priority` column. You must call this procedure once for each of the possible values of the `priority` column.

> **See Also:** For additional information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ADD_PRIORITY_datatype (
   gname              IN   VARCHAR2,
   pgroup             IN   VARCHAR2,
   value              IN   datatype,
   priority           IN   NUMBER);
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

Parameters

Table 34–6 ADD_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	Replicated object group for which you are creating a priority group.
pgroup	Name of the priority group.
value	Value of the priority group member. This would be one of the possible values of the associated `priority` column of a table using this priority group.
priority	Priority of this value. The higher the number, the higher the priority.

Exceptions

Table 34–7 ADD_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
duplicatevalue	Given value already exists in the priority group.
duplicatepriority	Given priority already exists in the priority group.
missingrepgroup	Given replicated object group does not exist.
missingprioritygroup	Given priority group does not exist.
typefailure	Given value has the incorrect datatype for the priority group.
notquiesced	Given replicated object group is not quiesced.

ADD_SITE_PRIORITY_SITE procedure

This procedure adds a new site to a site priority group. You must call this procedure from the master definition site.

> **See Also:** For additional information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
   gname          IN   VARCHAR2,
   name           IN   VARCHAR2,
   site           IN   VARCHAR2,
   priority       IN   NUMBER);
```

Parameters

Table 34–8 ADD_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	Replicated object group for which you are adding a site to a group.
name	Name of the site priority group to which you are adding a member.
site	Global database name of the site that you are adding.
priority	Priority level of the site that you are adding. A higher number indicates a higher priority level.

Exceptions

Table 34–9 ADD_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated object group does not exist.
missingpriority	Given site priority group does not exist.
duplicatepriority	Given priority level already exists for another site in the group.
duplicatevalue	Given site already exists in the site priority group.
notquiesced	Replicated object group is not quiesced.

ADD_*conflicttype*_RESOLUTION procedure

This procedure designates a method for resolving an update, delete, or uniqueness conflict. You must call these procedures from the master definition site. The procedure that you need to call is determined by the type of conflict that the routine resolves.

Conflict Type	Procedure Name
update	ADD_UPDATE_RESOLUTION
uniqueness	ADD_UNIQUE_RESOLUTION
delete	ADD_DELETE_RESOLUTION

See Also: For more information about designating methods to resolve update conflicts, selecting uniqueness conflict resolution methods, and, assigning delete conflict resolution methods see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
   sname                  IN   VARCHAR2,
   oname                  IN   VARCHAR2,
   column_group           IN   VARCHAR2,
   sequence_no            IN   NUMBER,
   method                 IN   VARCHAR2,
   parameter_column_name  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S,
   priority_group         IN   VARCHAR2     := NULL,
   function_name          IN   VARCHAR2     := NULL,
   comment                IN   VARCHAR2     := NULL);

DBMS_REPCAT.ADD_DELETE_RESOLUTION (
   sname                  IN   VARCHAR2,
   oname                  IN   VARCHAR2,
   sequence_no            IN   NUMBER,
   parameter_column_name  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S,
   function_name          IN   VARCHAR2,
   comment                IN   VARCHAR2     := NULL,
   method                 IN   VARCHAR2     := 'USER FUNCTION');
```

```
DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
   sname                 IN   VARCHAR2,
   oname                 IN   VARCHAR2,
   constraint_name       IN   VARCHAR2,
   sequence_no           IN   NUMBER,
   method                IN   VARCHAR2,
   parameter_column_name IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S,
   function_name         IN   VARCHAR2     := NULL,
   comment               IN   VARCHAR2     := NULL);
```

Parameters

Table 34–10 ADD_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
sname	Name of the schema containing the table to be replicated.
oname	Name of the table for which you are adding a conflict resolution routine.
column_group	Name of the column group for which you are adding a conflict resolution routine. Column groups are required for update conflict resolution routines only.
constraint_name	Name of the unique constraint or unique index for which you are adding a conflict resolution routine. Use the name of the unique index if it differs from the name of the associated unique constraint. Constraint names are required for uniqueness conflict resolution routines only.
sequence_no	Order in which the designated conflict resolution methods should be applied.
method	Type of conflict resolution routine that you want to create. This can be the name of one of the standard routines provided with advanced replication, or, if you have written your own routine, you should choose USER FUNCTION, and provide the name of your routine as the FUNCTION_NAME argument. The methods supported in this release are: MINIMUM, MAXIMUM, LATEST TIMESTAMP, EARLIEST TIMESTAMP, ADDITIVE, AVERAGE, PRIORITY GROUP, SITE PRIORITY, OVERWRITE, and DISCARD (for update conflicts) and APPEND SITE NAME, APPEND SEQUENCE NUMBER, and DISCARD (for uniqueness conflicts). There are no standard methods for delete conflicts.

Table 34–10 ADD_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
parameter_column_name	Name of the columns used to resolve the conflict. The standard methods operate on a single column. For example, if you are using the LATEST TIMESTAMP method for a column group, then you should pass the name of the column containing the timestamp value as this argument. If your are using a USER FUNCTION, then you can resolve the conflict using any number of columns. This argument accepts either a comma separated list of column names, or a PL/SQL table of type dbms_repcat.varchar2s. The single value '*' indicates that you want to use all of the columns in the table (or column group, for update conflicts) to resolve the conflict. If you specify '*', then the columns are passed to your function in alphabetical order.
priority_group	If you are using the PRIORITY GROUP or SITE PRIORITY update conflict resolution method, then you must supply the name of the priority group that you have created. See "Conflict Resolution" in the *Oracle8i Replication* manual. If you are using a different method, then you can use the default value for this argument, NULL. This argument is applicable to update conflicts only.
function_name	If you selected the USER FUNCTION method, or if you are adding a delete conflict resolution routine, then you must supply the name of the conflict resolution routine that you have written. If you are using one of the standard methods, then you can use the default value for this argument, NULL.
comment	This user comment is added to the RepResolution view.

Exceptions

Table 34–11 ADD_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingobject	Given object does not exist as a table in the given schema using row-level replication.
missingschema	Given schema does not exist.
missingcolumn	Column that you specified as part of the PARAMETER_COLUMN_NAME argument does not exist.
missinggroup	Given column group does not exist.
missingprioritygroup	priority group that you specified does not exist for the table.
invalidmethod	Resolution method that you specified is not recognized.
invalidparameter	Number of columns that you specified for the PARAMETER_COLUMN_NAME argument is invalid. (The standard routines take only one column name.)
missingfunction	User function that you specified does not exist.
missingconstraint	Constraint that you specified for a uniqueness conflict does not exist.
notquiesced	Object group that the given table belongs to is not quiesced.
duplicateresolution	Given conflict resolution method is already registered.
paramtype	Type is different from the type assigned to the priority group.

ALTER_MASTER_PROPAGATION procedure

This procedure alters the propagation method for a given object group at a given master site. This object group must be quiesced. You must call this procedure from the master definition site. If the master appears in the dblink_list or dblink_table, then ALTER_MASTER_PROPAGATION ignores that database link. You cannot change the propagation mode from a master to itself.

Syntax

```
DBMS_REPCAT.ALTER_MASTER_PROPAGATION (
   gname                IN   VARCHAR2,
   master               IN   VARCHAR2,
   { dblink_list        IN   VARCHAR2
   | dblink_table       IN   dbms_utility.dblink_array,}
   propagation_mode     IN   VARCHAR2 : ='asynchronous',
   comment              IN   VARCHAR2 := '');
```

> **Note:** This procedure is overloaded. The `dblink_list` and `dblink_table` parameters are mutually exclusive.

Parameters

Table 34–12 ALTER_MASTER_PROPAGATION Procedure Parameters

Parameter	Description
gname	Name of the object group to which to alter the propagation mode.
master	Name of the master site at which to alter the propagation mode.
dblink_list	A comma-separated list of database links for which to alter propagation. If NULL, then all masters except the master site being altered are used by default.
dblink_table	A PL/SQL table, indexed from position 1, of database links for which to alter propagation.
propagation_mode	Determines the manner in which changes from the given master site are propagated to the sites identified by the list of database links. Appropriate values are SYNCHRONOUS and ASYNCHRONOUS.
comment	This comment is added to the RepProp view.

Exceptions

Table 34–13 ALTER_MASTER_PROPAGATION Procedure Exceptions

Exception	Description
nonmasterdef	Local site is not the master definition site.
notquiesced	Local site is not quiesced.
typefailure	Propagation mode specified was not recognized.
nonmaster	List of database links includes a site that is not a master site.

ALTER_MASTER_REPOBJECT procedure

This procedure alters an object in your replicated environment. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
   sname        IN   VARCHAR2,
   oname        IN   VARCHAR2,
   type         IN   VARCHAR2,
   ddl_text     IN   VARCHAR2,
   comment      IN   VARCHAR2     := '',
   retry        IN   BOOLEAN      := FALSE);
```

Parameters

Table 34–14 ALTER_MASTER_REPOBJECT Procedure Parameters

Parameter	Description
sname	Schema containing the object that you want to alter.
oname	Name of the object that you want to alter.
type	Type of the object that you are altering. The types supported are: TABLE, INDEX, SYNONYM, TRIGGER, VIEW, PROCEDURE, FUNCTION, PACKAGE, and PACKAGE BODY.
ddl_text	The DDL text that you want used to alter the object. Oracle does not parse this DDL before applying it; therefore, you must ensure that your DDL text provides the appropriate schema and object name for the object being altered.
comment	If not NULL, then this comment is added to the COMMENT field of the RepObject view.
retry	If retry is TRUE, then ALTER_MASTER_REPOBJECT alters the object only at masters whose object status is not VALID.

Exceptions

Table 34–15 ALTER_MASTER_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Associated object group has not been suspended.
missingobject	Object identified by SNAME and ONAME does not exist.
typefailure	Given type parameter is not supported.
ddlfailure	DDL at the master definition site did not succeed.
commfailure	At least one master site is not accessible.

Usage Notes

If the DDL is supplied without specifying a schema, then the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema.

ALTER_PRIORITY procedure

This procedure alters the priority level associated with a given priority group member. You must call this procedure from the master definition site.

> **See Also:** See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ALTER_PRIORITY (
   gname            IN   VARCHAR2,
   pgroup           IN   VARCHAR2,
   old_priority     IN   NUMBER,
   new_priority     IN   NUMBER);
```

Parameters

Table 34–16 ALTER_PRIORITY Procedure Parameters

Parameter	Description
gname	Replicated object group with which the priority group is associated.
pgroup	Name of the priority group containing the priority that you want to alter.
old_priority	Current priority level of the priority group member.
new_priority	New priority level that you want assigned to the priority group member.

Exceptions

Table 34–17 ALTER_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
duplicatepriority	New priority level already exists in the priority group.
missingrepgroup	Given replicated object group does not exist.
missingvalue	Value was not registered by a call to DBMS_REPCAT.ADD_PRIORITY_datatype.
missingprioritygroup	Given priority group does not exist.
notquiesced	Given replicated object group is not quiesced.

ALTER_PRIORITY_*datatype* procedure

This procedure alters the value of a member in a priority group. You must call this procedure from the master definition site. The procedure that you must call is determined by the datatype of your priority column.

> **See Also:** For additional information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ALTER_PRIORITY_datatype (
   gname        IN   VARCHAR2,
   pgroup       IN   VARCHAR2,
   old_value    IN   datatype,
   new_value    IN   datatype);
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

Parameters

Table 34–18 ALTER_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	Replicated object group with which the priority group is associated.
pgroup	Name of the priority group containing the value that you want to alter.
old_value	Current value of the priority group member.
new_value	New value that you want assigned to the priority group member.

Exceptions

Table 34–19 ALTER_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
duplicatevalue	New value already exists in the priority group.
missingrepgroup	Given replicated object group does not exist.
missingprioritygroup	Given priority group does not exist.
missingvalue	Old value does not exist.
paramtype	New value has the incorrect datatype for the priority group.
typefailure	Given value has the incorrect datatype for the priority group.
notquiesced	Given replicated object group is not quiesced.

ALTER_SITE_PRIORITY procedure

This procedure alters the priority level associated with a given site. You must call this procedure from the master definition site.

> **See Also:** See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ALTER_SITE_PRIORITY (
   gname         IN   VARCHAR2,
   name          IN   VARCHAR2,
   old_priority  IN   NUMBER,
   new_priority  IN   NUMBER);
```

Parameters

Table 34–20 ALTER_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	Replicated object group with which the site priority group is associated.
name	Name of the site priority group whose member you are altering.
old_priority	Current priority level of the site whose priority level you want to change.
new_priority	New priority level for the site. A higher number indicates a higher priority level.

Exceptions

Table 34–21 ALTER_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated object group does not exist.
missingpriority	Old priority level is not associated with any group members.
duplicatepriority	New priority level already exists for another site in the group.
missingvalue	Old value does not already exist.
paramtype	New value has the incorrect datatype for the priority group.
notquiesced	Replicated object group is not quiesced.

ALTER_SITE_PRIORITY_SITE procedure

This procedure alters the site associated with a given priority level. You must call this procedure from the master definition site.

> **See Also:** See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.ALTER_SITE_PRIORITY_SITE (
   gname     IN   VARCHAR2,
   name      IN   VARCHAR2,
   old_site  IN   VARCHAR2,
   new_site  IN   VARCHAR2);
```

Parameters

Table 34–22 ALTER_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	Replicated object group with which the site priority group is associated.
name	Name of the site priority group whose member you are altering.
old_site	Current global database name of the site to dissociate from the priority level.
new_site	New global database name that you want to associate with the current priority level.

Exceptions

Table 34–23 ALTER_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated object group does not exist.
missingpriority	Given site priority group does not exist.
missingvalue	Old site is not a group member.
notquiesced	Replicated object group is not quiesced

ALTER_SNAPSHOT_PROPAGATION procedure

This procedure alters the propagation method for a given object group at the current snapshot site. This procedure pushes the deferred transaction queue at the snapshot site, locks the snapshot base tables, and regenerates any triggers and their associated packages. You must call this procedure from the snapshot site.

Syntax

```
DBMS_REPCAT.ALTER_SNAPSHOT_PROPAGATION (
   gname                IN  VARCHAR2,
   propagation_mode     IN  VARCHAR2,
   comment              IN  VARCHAR2   := '');
```

Parameters

Table 34–24 ALTER_SNAPSHOT_PROPAGATION Procedure Parameters

Parameter	Description
gname	Name of the object group for which to alter propagation mode.
propagation_mode	Manner in which changes from the current snapshot site are propagated to its associated master site. Appropriate values are SYNCHRONOUS and ASYNCHRONOUS.
comment	This comment is added to the RepProp view.

Exceptions

Table 34–25 ALTER_SNAPSHOT_PROPAGATION Procedure Exceptions

Exception	Description
missingrepgroup	Given replicated object group does not exist.
typefailure	Propagation mode was specified incorrectly.
nonsnapshot	Current site is not a snapshot site for the given object group.
commfailure	Cannot contact master.

CANCEL_STATISTICS procedure

This procedure stops collecting statistics about the successful resolution of update, uniqueness, and delete conflicts for a table.

Syntax

```
DBMS_REPCAT.CANCEL_STATISTICS (
   sname    IN   VARCHAR2,
   oname    IN   VARCHAR2);
```

Parameters

Table 34–26 CANCEL_STATISTICS Procedure Parameters

Parameter	Description
sname	Name of the schema in which the table is located.
oname	Name of the table for which you do not want to gather conflict resolution statistics.

Exceptions

Table 34–27 CANCEL_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Given schema does not exist.
missingobject	Given table does not exist.
statnotreg	Given table is not currently registered to collect statistics.

COMMENT_ON_COLUMN_GROUP procedure

This procedure updates the comment field in the RepColumn_Group view for a column group. This comment is not added at all master sites until the next call to DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT.

Syntax

```
DBMS_REPCAT.COMMENT_ON_COLUMN_GROUP (
   sname           IN   VARCHAR2,
   oname           IN   VARCHAR2,
   column_group    IN   VARCHAR2,
   comment         IN   VARCHAR2);
```

Parameters

Table 34–28 COMMENT_ON_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the replicated table with which the column group is associated.
column_group	Name of the column group.
comment	Text of the updated comment that you want included in the GROUP_COMMENT field of the RepColumn_Group view.

Exceptions

Table 34–29 COMMENT_ON_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missinggroup	Given column group does not exist.
missingobj	Object is missing.

COMMENT_ON_PRIORITY_GROUP/PRIORITY procedure

COMMENT_ON_PRIORITY_GROUP updates the comment field in the REPPRIORITY_GROUP view for a priority group. This comment is not added at all master sites until the next call to GENERATE_REPLICATION_SUPPORT.

COMMENT_ON_SITE_PRIORITY updates the comment field in the REPPRIORITY_GROUP view for a site priority group. This procedure is a wrapper for the COMMENT_ON_COLUMN_GROUP procedure and is provided as a convenience only. This procedure must be issued at the master definition site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_PRIORITY_GROUP (
   gname       IN   VARCHAR2,
   pgroup      IN   VARCHAR2,
   comment     IN   VARCHAR2);
```

```
DBMS_REPCAT.COMMENT_ON_SITE_PRIORITY (
   gname       IN   VARCHAR2,
   name        IN   VARCHAR2,
   comment     IN   VARCHAR2);
```

Parameters

Table 34–30 COMMENT_ON_PRIORITY_GROUP/COMMENT_ON_SITE_PRIORITY Parameters

Parameter	Description
gname	Name of the replicated object group.
pgroup/name	Name of the priority or site priority group.
comment	Text of the updated comment that you want included in the PRIORITY_COMMENT field of the RepPriority_Group view.

Exceptions

Table 34–31 COMMENT_ON_PRIORITY_GROUP/COMMENT_ON_SITE_PRIORITY Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Given replicated object group does not exist.
missingprioritygroup	Given priority group does not exist.

COMMENT_ON_REPGROUP procedure

This procedure updates the comment field in the REPGROUP view for a replicated object group. This procedure must be issued at the master definition site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_REPGROUP (
   gname     IN   VARCHAR2,
   comment   IN   VARCHAR2);
```

Parameters

Table 34–32 COMMENT_ON_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the object group that you want to comment on.
comment	Updated comment to include in the SCHEMA_COMMENT field of the RepGroup view.

Exceptions

Table 34–33 COMMENT_ON_REPGROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
commfailure	At least one master site is not accessible.

COMMENT_ON_REPSITES procedure

This procedure updates the comment field in the RepSite view for a replicated site. This procedure must be issued at the master definition site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_REPSITES (
   gname       IN   VARCHAR2,
   [ master    IN   VARCHAR,]
   comment     IN   VARCHAR2);
```

Parameters

Table 34–34 COMMENT_ON_REPSITES Procedure Parameters

Parameter	Description
gname	Name of the object group. This avoids confusion if a database is a master site in more than one replicated environment.
master	(Optional) The fully qualified database name of the master site that you want to comment on. To update comments at a snapshot site, omit this parameter.
comment	Text of the updated comment that you want to include in the MASTER_COMMENT field of the RepSites view.

Exceptions

Table 34–35 COMMENT_ON_REPSITES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
nonmaster	Invocation site is not a master site.
commfailure	At least one master site is not accessible.

COMMENT_ON_REPOBJECT procedure

This procedure updates the comment field in the RepObject view for a replicated object. This procedure must be issued at the master definition site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_REPOBJECT (
   sname    IN   VARCHAR2,
   oname    IN   VARCHAR2,
   type     IN   VARCHAR2,
   comment  IN   VARCHAR2);
```

Parameters

Table 34–36 COMMENT_ON_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the object that you want to comment on.
type	Type of the object.
comment	Text of the updated comment that you want to include in the OBJECT_COMMENT field of the RepObject view.

Exceptions

Table 34–37 COMMENT_ON_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist.
typefailure	Given type parameter is not supported.
commfailure	At least one master site is not accessible.

COMMENT_ON_*conflicttype*_RESOLUTION procedure

This procedure updates the comment field in the RepResolution view for a conflict resolution routine. The procedure that you need to call is determined by the type of conflict that the routine resolves. These procedures must be issued at the master definition site.

Conflict Type	Procedure Name
update	COMMENT_ON_UPDATE_RESOLUTION
uniqueness	COMMENT_ON_UNIQUE_RESOLUTION
delete	COMMENT_ON_DELETE_RESOLUTION

The comment is not added at all master sites until the next call to GENERATE_REPLICATION_SUPPORT.

Syntax

```
DBMS_REPCAT.COMMENT_ON_UPDATE_RESOLUTION (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   column_group         IN   VARCHAR2,
   sequence_no          IN   NUMBER,
   comment              IN   VARCHAR2);

DBMS_REPCAT.COMMENT_ON_UNIQUE_RESOLUTION (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   constraint_name      IN   VARCHAR2,
   sequence_no          IN   NUMBER,
   comment              IN   VARCHAR2);

DBMS_REPCAT.COMMENT_ON_DELETE_RESOLUTION (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   sequence_no          IN   NUMBER,
   comment              IN   VARCHAR2);
```

Parameters

Table 34–38 COMMENT_ON_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
sname	Name of the schema.
oname	Name of the replicated table with which the conflict resolution routine is associated.
column_group	Name of the column group with which the update conflict resolution routine is associated.
constraint_name	Name of the unique constraint with which the uniqueness conflict resolution routine is associated.
sequence_no	Sequence number of the conflict resolution procedure.
comment	The text of the updated comment that you want included in the RESOLUTION_COMMENT field of the RepResolution view.

Exceptions

Table 34–39 COMMENT_ON_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist.
missingresolution	Specified conflict resolution routine is not registered.

COMPARE_OLD_VALUES procedure

You have the option of comparing old column values for each non-key column of a replicated table for updates and deletes. The default is to compare old values for all columns. You can change this behavior at all master and snapshot sites by invoking DBMS_REPCAT.COMPARE_OLD_VALUES at the master definition site.

Syntax

```
DBMS_REPCAT.COMPARE_OLD_VALUES(
   sname           IN  VARCHAR2,
   oname           IN  VARCHAR2,
   { column_list   IN  VARCHAR2
   | column_table  IN  DBMS_REPCAT.VARCHAR2s,}
   operation       IN  VARCHAR2 := 'UPDATE',
   compare         IN  BOOLEAN := TRUE );
```

Note: This procedure is overloaded. The column_list and column_table parameters are mutually exclusive.

Parameters

Table 34–40 COMPARE_OLD_VALUES Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the replicated table.
column_list	A comma-separated list of the columns in the table. There must be no white space between entries.
column_table	Instead of a list, you can use a PL/SQL table of type DBMS_REPCAT.VARCHAR2S to contain the column names. The first column name should be at offset 1, the second at offset 2, and so on.
operation	Possible values are: UPDATE, DELETE, or the asterisk wildcard '*', which means update and delete.
compare	If compare is TRUE, then the old values of the specified columns are compared when sent. If compare is FALSE, then the old values of the specified columns are not compared when sent. Unspecified columns and unspecified operations are not affected. The specified change takes effect at the master definition site as soon as min_communication is TRUE for the table. The change takes effect at a master site or at a snapshot site the next time replication support is generated at that site with min_communication TRUE.

Note: The operation parameter allows you to decide whether to transmit old values for non-key columns when rows are deleted or when non-key columns are updated. If you do not send the old value, then Oracle sends a NULL in place of the old value and assumes the old value is equal to the current value of the column at the target side when the update or delete is applied.

Read "Minimizing Data Propagation for Update Conflict Resolution" in the *Oracle8i Replication* manual before changing the default behavior of Oracle.

Exceptions

Table 34–41 COMPARE_OLD_VALUES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist as a table in the given schema awaiting row-level replication information.
missingcolumn	At least one column is not in the table.
notquiesced	Replicated object group has not been suspended.
typefailure	An illegal operation is given.

CREATE_MASTER_REPGROUP procedure

This procedure creates a new, empty, quiesced master replication object group.

Syntax

```
DBMS_REPCAT.CREATE_MASTER_REPGROUP (
   gname           IN   VARCHAR2,
   group_comment   IN   VARCHAR2     := '',
   master_comment  IN   VARCHAR2     := ''),
   qualifier       IN   VARCHAR2     := '');
```

Parameters

Table 34–42 CREATE_MASTER_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the object group that you want to create.
group_comment	This comment is added to the RepGroup view.
master_comment	This comment is added to the RepSites view.
qualifier	Connection qualifier for object group. Be sure to use the @ sign, as shown in the example: See "Managing Master Groups" in the *Oracle8i Replication* manual.

Exceptions

Table 34–43 CREATE_MASTER_REPGROUP Procedure Exceptions

Exception	Description
duplicaterepgroup	Object group already exists.
norepopt	Advanced replication option is not installed.
missingrepgroup	Object group name was not specified.
qualifiertoolong	Connection qualifier is too long.

CREATE_MASTER_REPOBJECT procedure

This procedure indicates that an object is a replicated object.

Syntax

```
DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
   sname                IN   VARCHAR2,
   oname                IN   VARCHAR2,
   type                 IN   VARCHAR2,
   use_existing_object  IN   BOOLEAN     := TRUE,
   ddl_text             IN   VARCHAR2    := NULL,
   comment              IN   VARCHAR2    := '',
   retry                IN   BOOLEAN     := FALSE,
   copy_rows            IN   BOOLEAN     := TRUE,
   gname                IN   VARCHAR2    := '');
```

Parameters

Table 34–44 CREATE_MASTER_REPOBJECT Procedure Parameters

Parameters	Description
sname	Name of the schema in which the object that you want to replicate is located.
oname	Name of the object you are replicating. If DDL_TEXT is NULL, then this object must already exist in the given schema. To ensure uniqueness, table names should be a maximum of 27 bytes long, and package names should be no more than 24 bytes.
type	Type of the object that you are replicating. The types supported are: TABLE, INDEX, SYNONYM, TRIGGER, VIEW, PROCEDURE, FUNCTION, PACKAGE, and PACKAGE BODY.
use_existing_object	Indicate TRUE if you want to reuse any objects of the same type and shape at the current master sites. See Table 34–46 for more information.
ddl_text	If the object does not already exist at the master definition site, then you must supply the DDL text necessary to create this object. PL/SQL packages, package bodies, procedures, and functions must have a trailing semicolon. SQL statements do not end with trailing semicolon. Oracle does not parse this DDL before applying it; therefore, you must ensure that your DDL text provides the appropriate schema and object name for the object being created.
comment	This comment is added to the OBJECT_COMMENT field of the RepObject view.
retry	Indicate TRUE if you want Oracle to reattempt to create an object that it was previously unable to create. Use this if the error was transient or has since been rectified; for example, if you previously had insufficient resources. If this is TRUE, then Oracle creates the object only at master sites whose object status is not VALID.
copy_rows	Indicate TRUE if you want the initial contents of a newly replicated object to match the contents of the object at the master definition site. See Table 34–46 for more information.
gname	Name of the object group in which you want to create the replicated object. The schema name is used as the default object group name if none is specified.

Table 34–45 CREATE_MASTER_REPOBJECT Procedure Exceptions

Exceptions	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Replicated object group has not been suspended.
duplicateobject	Given object already exists in the replicated object group and retry is FALSE, or if a name conflict occurs.
missingobject	Object identified by SNAME and ONAME does not exist and appropriate DDL has not been provided.
typefailure	Objects of the given type cannot be replicated.
ddlfailure	DDL at the master definition site did not succeed.
commfailure	At least one master site is not accessible.
notcompat	Not all remote masters in 7.3 compatibility mode.

Object Creations

Table 34–46 Object Creation at Master Sites

Object Already Exists?	COPY_ROWS	USE_EXISTING_OBJECTS	Result
yes	TRUE	TRUE	duplicatedobject message if objects do not match. For tables, use data from master definition site.
yes	FALSE	TRUE	duplicatedobject message if objects do not match. For tables, DBA must ensure contents are identical.
yes	TRUE/FALSE	FALSE	duplicatedobject message.
no	TRUE	TRUE/FALSE	Object is created. Tables populated using data from master definition site.
no	FALSE	TRUE/FALSE	Object is created. DBA must populate tables and ensure consistency of tables at all sites.

Usage Notes

If the DDL is supplied without specifying a schema, then the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema.

CREATE_SNAPSHOT_REPGROUP procedure

This procedure creates a new, empty snapshot replication object group in your local database.

Syntax

```
DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP (
   gname              IN   VARCHAR2,
   master             IN   VARCHAR2,
   comment            IN   VARCHAR2     := '',
   propagation_mode   IN   VARCHAR2     := 'ASYNCHRONOUS',
   fname              IN   VARCHAR2     := NULL);
```

Parameters

Table 34–47 CREATE_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated object group. This object group must exist at the given master site.
master	Fully qualified database name of the database in the replicated environment to use as the master.
comment	This comment is added to the RepGroup view.
propagation_mode	Method of propagation for all updatable snapshots in the object group. Acceptable values are SYNCHRONOUS and ASYNCHRONOUS.
fname	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 34–48 CREATE_SNAPSHOT_REPGROUP Procedure Exceptions

Exception	Description
duplicaterepgroup	Object group already exists at the invocation site.
nonmaster	Given database is not a master site.
commfailure	Given database is not accessible.
norepopt	Advanced replication option is not installed.
typefailure	Propagation mode was specified incorrectly.
missingrepgroup	If replicated object group not at master site.

Usage Notes

CREATE_SNAPSHOT_REPGROUP automatically calls REGISTER_SNAPSHOT_REPGROUP, but ignores any errors that may have happened during registration.

CREATE_SNAPSHOT_REPOBJECT procedure

This procedure adds a replicated object to your snapshot site.

Syntax

```
DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
   sname                  IN   VARCHAR2,
   oname                  IN   VARCHAR2,
   type                   IN   VARCHAR2,
   ddl_text               IN   VARCHAR2  := '',
   comment                IN   VARCHAR2  := '',
   gname                  IN   VARCHAR2  := '',
   gen_objs_owner         IN   VARCHAR2 := '',
   min_communication      IN   BOOLEAN  := TRUE ,
   generate_80_compatible IN   BOOLEAN  := TRUE);
```

Parameters

Table 34–49 CREATE_SNAPSHOT_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the object that you want to add to the replicated snapshot object group. ONAME must exist at the associated master site.
type	Type of the object that you are replicating. The types supported for snapshot sites are: PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, SNAPSHOT, SYNONYM, TRIGGER, and VIEW.
ddl_text	For objects of type SNAPSHOT, the DDL text needed to create the object; for other types, use the default, (" (an empty string). If a snapshot with the same name already exists, then Oracle ignores the DDL and registers the existing snapshot as a replicated object. If the master table for a snapshot does not exist in the replicated object group of the master site designated for this schema, then Oracle raises a missingobject error.
comment	This comment is added to the OBJECT_COMMENT field of the RepObject view.
gname	Name of the replicated object group to which you are adding an object. The schema name is used as the default group name if none is specified.
gen_objs_owner	Name of the user you want to assign as owner of the transaction.
min_communication	Set to FALSE if any master site is running Oracle7 release 7.3. Set to TRUE to minimize new and old values of propagation. The default is TRUE. For more information, see "Conflict Resolution" in the *Oracle8i Replication* manual.
generate_80_ compatible	Set to TRUE if any master site is running a version of Oracle Server prior to Oracle8*i* release 8.1.5. Set to FALSE if replicated environment is a pure Oracle8*i* release 8.1.5 or greater environment.

Exceptions

Table 34–50 CREATE_SNAPSHOT_REPOBJECT Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
nonmaster	Master is no longer a master site.
missingobject	Given object does not exist in the master's replicated object group.
duplicateobject	Given object already exists with a different shape.
typefailure	Type is not an allowable type.
ddlfailure	DDL did not succeed.
commfailure	Master site is not accessible.
missingschema	Schema does not exist as a database schema.
badsnapddl	DDL was executed but snapshot does not exist.
onlyonesnap	Only one snapshot for master table can be created.
badsnapname	Snapshot base table differs from master table.
missingrepgroup	Replicated object group does not exist.

Usage Notes

If the DDL is supplied without specifying a schema, then the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema.

DEFINE_COLUMN_GROUP procedure

This procedure creates an empty column group. You must call this procedure from the master definition site.

> **See Also:** For more information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DEFINE_COLUMN_GROUP (
   sname            IN   VARCHAR2,
   oname            IN   VARCHAR2,
   column_group     IN   VARCHAR2,
   comment          IN   VARCHAR2 := NULL);
```

Parameters

Table 34–51 DEFINE_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table for which you are creating a column group.
column_group	Name of the column group that you want to create.
comment	This user text is displayed in the RepColumn_Group view.

Exceptions

Table 34–52 DEFINE_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingobject	Given table does not exist.
duplicategroup	Given column group already exists for the table.
notquiesced	Object group that the given table belongs to is not quiesced.

DEFINE_PRIORITY_GROUP procedure

This procedure creates a new priority group for a replicated object group. You must call this procedure from the master definition site.

> **See Also:** See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DEFINE_PRIORITY_GROUP (
   gname         IN   VARCHAR2,
   pgroup        IN   VARCHAR2,
   datatype      IN   VARCHAR2,
   fixed_length  IN   INTEGER := NULL,
   comment       IN   VARCHAR2 := NULL);
```

Parameters

Table 34–53 DEFINE_PRIORITY_GROUP Procedure Parameters

Parameter	Description
gname	Replicated object group for which you are creating a priority group.
pgroup	Name of the priority group that you are creating.
datatype	Datatype of the priority group members. The datatypes supported are: CHAR, VARCHAR2, NUMBER, DATE, RAW, NCHAR, and NVARCHAR2.
fixed_length	You must provide a column length for the CHAR datatype. All other types can use the default, NULL.
comment	This user comment is added to the RepPriority view.

Exceptions

Table 34–54 DEFINE_PRIORITY_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated object group does not exist.
duplicatepriority group	Given priority group already exists in the replicated object group.
typefailure	Given datatype is not supported.
notquiesced	Replicated object group is not quiesced.

DEFINE_SITE_PRIORITY procedure

This procedure creates a new site priority group for a replicated object group. You must call this procedure from the master definition site.

> **See Also:** See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DEFINE_SITE_PRIORITY (
   gname        IN   VARCHAR2,
   name         IN   VARCHAR2,
   comment      IN   VARCHAR2 := NULL);
```

Parameters

Table 34–55 DEFINE_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	The replicated object group for which you are creating a site priority group.
name	Name of the site priority group that you are creating.
comment	This user comment is added to the `RepPriority` view.

Exceptions

Table 34–56 DEFINE_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated object group does not exist.
duplicate prioritygroup	Given site priority group already exists in the replicated object group.
notquiesced	Replicated object group is not quiesced.

DO_DEFERRED_REPCAT_ADMIN procedure

This procedure executes the local outstanding deferred administrative procedures for the given replicated object group at the current master site, or (with assistance from job queues) for all master sites.

Syntax

```
DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN (
   gname          IN   VARCHAR2,
   all_sites      IN   BOOLEAN := FALSE);
```

Parameters

Table 34–57 DO_DEFERRED_REPCAT_ADMIN Procedure Parameters

Parameter	Description
gname	Name of the replicated object group.
all_sites	If this is TRUE, then use a job to execute the local administrative procedures at each master.

Exceptions

Table 34–58 DO_DEFERRED_REPCAT_ADMIN Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.
commfailure	At least one master site is not accessible and all_sites is TRUE.

Usage Notes

DO_DEFERRED_REPCAT_ADMIN executes only those administrative requests submitted by the connected user that called DO_DEFERRED_REPCAT_ADMIN. Requests submitted by other users are ignored.

DROP_COLUMN_GROUP procedure

This procedure drops a column group. You must call this procedure from the master definition site.

See Also: See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_COLUMN_GROUP (
   sname        IN   VARCHAR2,
   oname        IN   VARCHAR2,
   column_group IN   VARCHAR2);
```

Parameters

Table 34–59 DROP_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table whose column group you are dropping.
column_group	Name of the column group that you want to drop.

Exceptions

Table 34–60 DROP_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
referenced	Given column group is being used in conflict detection and resolution.
missingobject	Given table does not exist.
missinggroup	Given column group does not exist.
notquiesced	Replicated object group that the table belongs to is not quiesced.

DROP_GROUPED_COLUMN procedure

This procedure removes members from a column group. You must call this procedure from the master definition site.

See Also: For more information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_GROUPED_COLUMN (
   sname                 IN   VARCHAR2,
   oname                 IN   VARCHAR2,
   column_group          IN   VARCHAR2,
   list_of_column_names  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S);
```

Parameters

Table 34–61 DROP_GROUPED_COLUMN Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table in which the column group is located.
column_group	Name of the column group from which you are removing members.
list_of_column_names	Names of the columns that you are removing from the designated column group. This can either be a comma-separated list or a PL/SQL table of column names. The PL/SQL table must be of type dbms_repcat.varchar2s.

Exceptions

Table 34–62 DROP_GROUPED_COLUMN Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingobject	Given table does not exist.
notquiesced	Replicated object group that the table belongs to is not quiesced.

DROP_MASTER_REPGROUP procedure

This procedure drops a replicated object group from your current site. To drop the replicated object group from all master sites, including the master definition site, you can call this procedure at the master definition site, and set the final argument to TRUE.

Syntax

```
DBMS_REPCAT.DROP_MASTER_REPGROUP (
   gname             IN VARCHAR2,
   drop_contents     IN BOOLEAN    := FALSE,
   all_sites         IN BOOLEAN    := FALSE);
```

Parameters

Table 34–63 DROP_MASTER_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated object group that you want to drop from the current master site.
drop_contents	By default, when you drop the object group at a master site, all of the objects remain in the database. They simply are no longer replicated; that is, the replicated objects in the object group no longer send changes to, or receive changes from, other master sites. If you set this to TRUE, then any replicated objects in the replicated object group are dropped from their associated schemas.
all_sites	If this is TRUE and if the invocation site is the master definition site, then the procedure synchronously multicasts the request to all masters. In this case, execution is immediate at the master definition site and may be deferred at all other master sites.

Exceptions

Table 34–64 DROP_MASTER_REPGROUP Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.
nonmasterdef	Invocation site is not the master definition site and ALL_SITES is TRUE.
commfailure	At least one master site is not accessible and ALL_SITES is TRUE.
fullqueue	Deferred RPC queue has entries for the replicated object group.
masternotremoved	Master does not recognize the masterdef and ALL_SITES is TRUE.

DROP_MASTER_REPOBJECT procedure

This procedure drops a replicated object from a replicated object group. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.DROP_MASTER_REPOBJECT (
   sname          IN   VARCHAR2,
   oname          IN   VARCHAR2,
   type           IN   VARCHAR2,
   drop_objects   IN   BOOLEAN     := FALSE);
```

Parameters

Table 34–65 DROP_MASTER_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the object that you want to remove from the replicated object group.
type	Type of object that you want to drop.
drop_objects	By default, the object remains in the schema, but is dropped from the replicated object group; that is, any changes to the object are no longer replicated to other master and snapshot sites. To completely remove the object from the replicated environment, set this argument to TRUE.

Exceptions

Table 34–66 DROP_MASTER_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist.
typefailure	Given type parameter is not supported.
commfailure	At least one master site is not accessible.

DROP_PRIORITY procedure

This procedure drops a member of a priority group by priority level. You must call this procedure from the master definition site.

> **See Also:** See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_PRIORITY(
      gname          IN   VARCHAR2,
      pgroup         IN   VARCHAR2,
      priority_num   IN   NUMBER);
```

Parameters

Table 34–67 DROP_PRIORITY Procedure Parameters

Parameter	Description
gname	Replicated object group with which the priority group is associated.
pgroup	Name of the priority group containing the member that you want to drop.
priority_num	Priority level of the priority group member that you want to remove from the group.

Exceptions

Table 34–68 DROP_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated object group does not exist.
missingprioritygroup	Given priority group does not exist.
notquiesced	Replicated object group is not quiesced.

DROP_PRIORITY_GROUP procedure

This procedure drops a priority group for a given replicated object group. You must call this procedure from the master definition site.

See Also: See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_PRIORITY_GROUP (
   gname     IN   VARCHAR2,
   pgroup    IN   VARCHAR2);
```

Parameters

Table 34–69 DROP_PRIORITY_GROUP Procedure Parameters

Parameter	Description
gname	Replicated object group with which the priority group is associated.
pgroup	Name of the priority group that you want to drop.

Exceptions

Table 34–70 DROP_PRIORITY_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated object group does not exist.
referenced	Given priority group is being used in conflict resolution.
notquiesced	Given replicated object group is not quiesced.

DROP_PRIORITY_*datatype* procedure

This procedure drops a member of a priority group by value. You must call this procedure from the master definition site. The procedure that you must call is determined by the datatype of your `priority` column.

See Also: See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_PRIORITY_datatype (
   gname    IN   VARCHAR2,
   pgroup   IN   VARCHAR2,
   value    IN   datatype);
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

Parameters

Table 34–71 DROP_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	Replicated object group with which the priority group is associated.
pgroup	Name of the priority group containing the member that you want to drop.
value	Value of the priority group member that you want to remove from the group.

Exceptions

Table 34–72 DROP_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated object group does not exist.
missingprioritygroup	Given priority group does not exist.
paramtype, typefailure	Value has the incorrect datatype for the priority group.
notquiesced	Given replicated object group is not quiesced

DROP_SITE_PRIORITY procedure

This procedure drops a site priority group for a given replicated object group. You must call this procedure from the master definition site.

> **See Also:** See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_SITE_PRIORITY (
   gname       IN   VARCHAR2,
   name        IN   VARCHAR2);
```

Parameters

Table 34–73 DROP_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	Replicated object group with which the site priority group is associated.
name	Name of the site priority group that you want to drop.

Exceptions

Table 34–74 DROP_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated object group does not exist.
referenced	Given site priority group is being used in conflict resolution.
notquiesced	Given replicated object group is not quiesced

DROP_SITE_PRIORITY_SITE procedure

This procedure drops a given site, by name, from a site priority group. You must call this procedure from the master definition site.

See Also: See "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.DROP_SITE_PRIORITY_SITE (
   gname      IN   VARCHAR2,
   name       IN   VARCHAR2,
   site       IN   VARCHAR2);
```

Parameters

Table 34–75 DROP_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	Replicated object group with which the site priority group is associated.
name	Name of the site priority group whose member you are dropping.
site	Global database name of the site you are removing from the group.

Exceptions

Table 34–76 DROP_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingrepgroup	Given replicated object group does not exist.
missingpriority	Given site priority group does not exist.
notquiesced	Given replicated object group is not quiesced.

DROP_SNAPSHOT_REPGROUP procedure

This procedure drops a snapshot site from your replicated environment.

Syntax

```
DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP (
   gname                IN   VARCHAR2,
   drop_contents        IN   BOOLEAN   := FALSE);
```

Parameters

Table 34–77 DROP_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated object group that you want to drop from the current snapshot site. All objects generated to support replication, such as triggers and packages, are dropped.
drop_contents	By default, when you drop the replicated object group at a snapshot site, all of the objects remain in their associated schemas; they simply are no longer replicated. If you set this to TRUE, then any replicated objects in the replicated object group are dropped from their schemas.

Exceptions

Table 34–78 DROP_SNAPSHOT_REPGROUP Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
missingrepgroup	Specified object group does not exist.

Usage Notes

DROP_SNAPSHOT_REPGROUP automatically calls UNREGISTER_SNAPSHOT_REPGROUP to unregister the snapshot, but ignores any errors that may have occurred during unregistration.

DROP_SNAPSHOT_REPOBJECT procedure

This procedure drops a replicated object from a snapshot site.

Syntax

```
DBMS_REPCAT.DROP_SNAPSHOT_REPOBJECT (
   sname          IN   VARCHAR2,
   oname          IN   VARCHAR2,
   type           IN   VARCHAR2,
   drop_objects   IN   BOOLEAN  := FALSE);
```

Parameters

Table 34–79 DROP_SNAPSHOT_REPOBJECT Procedure Parameters

Parameter	Description
sname	Name of the schema in which the object is located.
oname	Name of the object that you want to drop from the replicated object group.
type	Type of the object that you want to drop.
drop_objects	By default, the object remains in its associated schema, but is dropped from its associated object group. To completely remove the object from its schema at the current snapshot site, set this argument to TRUE.

Exceptions

Table 34–80 DROP_SNAPSHOT_REPOBJECT Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
missingobject	Given object does not exist.
typefailure	Given type parameter is not supported.

DROP_*conflicttype*_RESOLUTION procedure

This procedure drops an update, delete, or uniqueness conflict resolution routine. You must call these procedures from the master definition site. The procedure that you must call is determined by the type of conflict that the routine resolves.

Conflict Type	Procedure Name
update	DROP_UPDATE_RESOLUTION
uniqueness	DROP_UNIQUE_RESOLUTION
delete	DROP_DELETE_RESOLUTION

Syntax

```
DBMS_REPCAT.DROP_UPDATE_RESOLUTION (
   sname               IN   VARCHAR2,
   oname               IN   VARCHAR2,
   column_group        IN   VARCHAR2,
   sequence_no         IN   NUMBER);

DBMS_REPCAT.DROP_DELETE_RESOLUTION (
   sname               IN   VARCHAR2,
   oname               IN   VARCHAR2,
   sequence_no         IN   NUMBER);

DBMS_REPCAT.DROP_UNIQUE_RESOLUTION (
   sname               IN   VARCHAR2,
   oname               IN   VARCHAR2,
   constraint_name     IN   VARCHAR2,
   sequence_no         IN   NUMBER);
```

Parameters

Table 34–81 DROP_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the table for which you want to drop a conflict resolution routine.
column_group	Name of the column group for which you want to drop an update conflict resolution routine.
constraint_name	Name of the Unique constraint for which you want to drop a unique conflict resolution routine.
sequence_no	Sequence number assigned to the conflict resolution method that you want to drop. This number uniquely identifies the routine.

Exceptions

Table 34–82 DROP_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
missingobject	Given object does not exist as a table in the given schema, or a conflict resolution routine with the given sequence number is not registered.
notquiesced	Replicated object group is not quiesced.

EXECUTE_DDL procedure

This procedure supplies DDL that you want to have executed at some or all master sites. You can call this procedure only from the master definition site.

Syntax

```
DBMS_REPCAT.EXECUTE_DDL (
   gname           IN   VARCHAR2,
   { master_list   IN   VARCHAR2     := NULL
   | master_table  IN   DBMS_UTILITY.DBLINK_ARRAY,}
   DDL_TEXT        IN   VARCHAR2);
```

Parameters

Table 34–83 EXECUTE_DDL Procedure Parameters

Parameter	Description
gname	Name of the replicated object group.
master_list	A comma-separated list of master sites at which you want to execute the supplied DDL. There must be no extra white space between site names. The default value, NULL, indicates that the DDL should be executed at all sites, including the master definition site.
master_table	A table of master sites at which you want to execute the supplied DDL. The first master should be at offset 1, the second at offset 2, and so on.
ddl_text	The DDL that you want to have executed at each of the given master sites.

Exceptions

Table 34–84 EXECUTE_DDL Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
nonmaster	At least one site is not a master site.
ddlfailure	DDL at the master definition site did not succeed.
commfailure	At least one master site is not accessible.

Usage Notes

If the DDL is supplied without specifying a schema, then the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema. This procedure is overloaded. The MASTER_LIST and MASTER_TABLE parameters are mutually exclusive.

GENERATE_REPLICATION_SUPPORT procedure

This procedure generates the triggers and packages needed to support replication. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
  sname                     IN    VARCHAR2,
  oname                     IN    VARCHAR2,
  type                      IN    VARCHAR2,
  package_prefix            IN    VARCHAR2   := NULL,
  procedure_prefix          IN    VARCHAR2   := NULL,
  distributed               IN    BOOLEAN    := TRUE,
  gen_objs_owner            IN    VARCHAR2   := NULL,
  min_communication         IN    BOOLEAN    := TRUE,
  generate_80_compatible    IN    BOOLEAN    := TRUE);
```

Parameters

Table 34–85 GENERATE_REPLICATION_SUPPORT Procedure Parameters

Parameter	Description
sname	Schema in which the object is located.
oname	Name of the object for which you are generating replication support.
type	Type of the object. The types supported are: TABLE, PACKAGE, and PACKAGE BODY.
package_prefix	For objects of type PACKAGE or PACKAGE BODY this value is prepended to the generated wrapper package name. The default is DEFER_.
procedure_prefix	For objects of type PACKAGE or PACKAGE BODY, this value is prepended to the generated wrapper procedure names. By default, no prefix is assigned.
distributed	This must be set to TRUE.
gen_objs_owner	For objects of type PACKAGE or PACKAGE BODY, the schema in which the generated object should be created. If NULL, then the objects will be created in sname.
min_communication	Set to FALSE if any master site is running Oracle7 release 7.3. Set to TRUE when you want propagation of new and old values to be minimized. The default is TRUE. For more information, see "Conflict Resolution' in the *Oracle8i Replication* manual.
generate_80_ compatible	Set to TRUE if any master site is running a version of Oracle Server prior to Oracle8*i* release 8.1.5. Set to FALSE if replicated environment is a pure Oracle8*i* release 8.1.5 or greater environment.

Exceptions

Table 34–86 GENERATE_REPLICATION_SUPPORT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist as a table in the given schema awaiting row-level replication information or as a package (body) awaiting wrapper generation.
typefailure	Given type parameter is not supported.
notquiesced	Replicated object group has not been suspended.
commfailure	At least one master site is not accessible.
missingschema	Schema does not exist.
dbnotcompatible	One of the masters is not 7.3 compatible.
duplicateobject	Object already exists.

GENERATE_SNAPSHOT_SUPPORT procedure

This procedure activates triggers and generate packages needed to support the replication of updatable snapshots or procedural replication.You must call this procedure from the snapshot site.

Syntax

```
DBMS_REPCAT.GENERATE_SNAPSHOT_SUPPORT (
   sname                    IN VARCHAR2,
   oname                    IN VARCHAR2,
   type                     IN VARCHAR2,
   gen_objs_owner           IN VARCHAR2 := '',
   min_communication        IN BOOLEAN  := TRUE,
   generate_80_compatible   IN BOOLEAN  := TRUE);
```

Parameters

Table 34–87 GENERATE_SNAPSHOT_SUPPORT Procedure Parameters

Parameter	Description
sname	Schema in which the object is located.
oname	The name of the object that you are generating support for.

Table 34–87 GENERATE_SNAPSHOT_SUPPORT Procedure Parameters

Parameter	Description
type	Type of the object. The types supported are SNAPSHOT, PACKAGE, and PACKAGE BODY.
gen_objs_owner	For objects of type PACKAGE or PACKAGE BODY, the schema in which the generated object should be created. If NULL, then the objects will be created in sname.
min_communication	If TRUE, then the update trigger sends the new value of a column only if the update statement modifies the column. The update trigger sends the old value of the column only if it is a key column or a column in a modified column group.
generate_80_compatible	Set to TRUE if any master site is running a version of Oracle Server prior to Oracle8*i* release 8.1.5. Set to FALSE if replicated environment is a pure Oracle8*i* release 8.1.5 or greater environment.

Exceptions

Table 34–88 GENERATE_SNAPSHOT_SUPPORT Procedure Exceptions

Exceptions	Descriptions
nonsnapshot	Invocation site is not a snapshot site.
missingobject	Given object does not exist as a snapshot in the replicated schema awaiting row / column-level replication information or as a package (body) awaiting wrapper generation.
typefailure	Given type parameter is not supported.
missingschema	Specified owner of generated objects does not exist.
missingremoteobject	Master object has not yet generated replication support.
commfailure	Master is not accessible.

Usage Notes

CREATE_SNAPSHOT_REPOBJECT automatically generates snapshot support for updatable snapshots.

MAKE_COLUMN_GROUP procedure

This procedure creates a new column group with one or more members. You must call this procedure from the master definition site.

See Also: For more information, see "Conflict Resolution" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.MAKE_COLUMN_GROUP (
   sname                 IN   VARCHAR2,
   oname                 IN   VARCHAR2,
   column_group          IN   VARCHAR2,
   list_of_column_names  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S);
```

Parameters

Table 34–89 MAKE_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table for which you are creating a new column group.
column_group	Name that you want assigned to the column group that you are creating.
list_of_column_names	Names of the columns that you are grouping. This can either be a comma-separated list or a PL/SQL table of column names. The PL/SQL table must be of type dbms_repcat.varchar2s. Use the single value '*' to create a column group that contains all of the columns in your table.

Exceptions

Table 34–90 MAKE_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
duplicategroup	Given column group already exists for the table.
missingobject	Given table does not exist.
missingcolumn	Given column does not exist in the designated table.
duplicatecolumn	Given column is already a member of another column group.
notquiesced	Replicated object group is not quiesced.

PURGE_MASTER_LOG procedure

This procedure removes local messages in the RepCatLog associated with a given identification number, source, or replicated object group.

Syntax

```
DBMS_REPCAT.PURGE_MASTER_LOG (
   id      IN   NATURAL,
   source  IN   VARCHAR2,
   gname   IN   VARCHAR2);
```

Parameters

Table 34–91 PURGE_MASTER_LOG Procedure Parameters

Parameter	Description
id	Identification number of the request, as it appears in the RepCatLog view.
source	Master site from which the request originated.
gname	Name of the replicated object group for which the request was made.

Exceptions

Table 34–92 PURGE_MASTER_LOG Procedure Exceptions

Exception	Description
nonmaster	gname is not NULL, and the invocation site is not a master site.

PURGE_STATISTICS procedure

This procedure removes information from the RepResolution_Statistics view.

Syntax

```
DBMS_REPCAT.PURGE_STATISTICS (
   sname      IN   VARCHAR2,
   oname      IN   VARCHAR2,
   start_date IN   DATE,
   end_date   IN   DATE);
```

Parameters

Table 34–93 PURGE_STATISTICS Procedure Parameters

Parameter	Description
sname	Name of the schema in which the replicated table is located.
oname	Name of the table whose conflict resolution statistics you want to purge.
start_date/end_date	Range of dates for which you want to purge statistics. If START_DATE is NULL, then purge all statistics up to the END_DATE. If END_DATE is NULL, then purge all statistics after the START_DATE.

Exceptions

Table 34–94 PURGE_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Given schema does not exist.
missingobject	Given table does not exist.
statnotreg	Table not registered to collect statistics.

REFRESH_SNAPSHOT_REPGROUP procedure

This procedure refreshes a snapshot site object group with the most recent data from its associated master site.

Syntax

```
DBMS_REPCAT.REFRESH_SNAPSHOT_REPGROUP (
   gname                 IN   VARCHAR2,
   drop_missing_contents IN   BOOLEAN    := FALSE,
   refresh_snapshots     IN   BOOLEAN    := FALSE,
   refresh_other_objects IN   BOOLEAN    := FALSE);
```

Parameters

Table 34–95 REFRESH_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated object group.
drop_missing_ contents	If an object was dropped from the replicated object group, then it is not automatically dropped from the schema at the snapshot site. It is simply no longer replicated; that is, changes to this object are no longer sent to its associated master site. Snapshots can continue to be refreshed from their associated master tables; however, any changes to an updatable snapshot are lost. When an object is dropped from the object group, you can choose to have it dropped from the schema entirely by setting this argument to TRUE.
refresh_snapshots	Set this to TRUE to refresh the contents of the snapshots in the replicated object group.
refresh_other_ objects	Set this to TRUE to refresh the contents of the non-snapshot objects in the replicated object group.

Exceptions

Table 34–96 REFRESH_SNAPSHOT_REPGROUP Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
nonmaster	Master is no longer a master site.
commfailure	Master is not accessible.
missingrepgroup	Object group name not specified.

REGISTER_SNAPSHOT_REPGROUP procedure

This procedure facilitates the administration of snapshots at their respective master sites by inserting / modifying / deleting from registered_snapshot_groups.

Syntax

```
DBMS_REPCAT.REGISTER_SNAPSHOT_REPGROUP (
   gname            IN   VARCHAR2,
   snapsite         IN   VARCHAR2,
   comment          IN   VARCHAR2  := NULL,
   rep_type         IN   NUMBER    := reg_unknown,
   fname            IN   VARCHAR2  := NULL);
```

Parameters

Table 34–97 REGISTER_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the snapshot object group to be registered.
snapsite	Global name of the snapshot site.
comment	Comment for the snapshot site or update for an existing comment.
rep_type	Version of the snapshot group. Valid constants that can be assigned include reg_uknown (the default), reg_v7_group, reg_v8_group, and reg_repapi_group.
fname	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 34–98 REGISTER_SNAPSHOT_REPGROUP Procedure Exceptions

Exception	Description
missingrepgroup	Object group name not specified.
nullsitename	A snapshot site was not specified.
nonmaster	Procedure must be executed at the snapshot's master site.
duplicaterepgroup	Object already exists.

REGISTER_STATISTICS procedure

This procedure collects information about the successful resolution of update, delete, and uniqueness conflicts for a table.

Syntax

```
DBMS_REPCAT.REGISTER_STATISTICS (
   sname IN   VARCHAR2,
   oname IN   VARCHAR2);
```

Parameters

Table 34–99 REGISTER_STATISTICS Procedure Parameters

Parameter	Description
sname	Name of the schema in which the table is located.
oname	Name of the table for which you want to gather conflict resolution statistics.

Exceptions

Table 34–100 REGISTER_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Given schema does not exist.
missingobject	Given table does not exist.

RELOCATE_MASTERDEF procedure

This procedure changes your master definition site to another master site in your replicated environment.

Syntax

```
DBMS_REPCAT.RELOCATE_MASTERDEF (
   gname                 IN   VARCHAR2,
   old_masterdef         IN   VARCHAR2,
   new_masterdef         IN   VARCHAR2,
   notify_masters        IN   BOOLEAN    := TRUE,
   include_old_masterdef IN   BOOLEAN    := TRUE,
   require_flavor_change IN   BOOLEAN    := FALSE);
```

Parameters

Table 34–101 RELOCATE_MASTERDEF Procedure Parameters

Parameter	Description
gname	Name of the object group whose master definition you want to relocate.
old_masterdef	Fully qualified database name of the current master definition site.
new_masterdef	Fully qualified database name of the existing master site that you want to make the new master definition site.
notify_masters	If this is TRUE, then the procedure synchronously multicasts the change to all masters (including OLD_MASTERDEF only if INCLUDE_OLD_MASTERDEF is TRUE). If any master does not make the change, then roll back the changes at all masters.
include_old_ masterdef	If NOTIFY_MASTERS is TRUE and if INCLUDE_OLD_MASTERDEF is also TRUE, then the old master definition site is also notified of the change.
require_flavor_ change	This system parameter is for internal use only. Do not set the parameter unless so directed by Oracle Worldwide Support.

Exceptions

Table 34–102 RELOCATE_MASTERDEF Procedure Exceptions

Exception	Description
nonmaster	NEW_MASTERDEF is not a master site or the invocation site is not a master site.
nonmasterdef	OLD_MASTERDEF is not the master definition site.
commfailure	At least one master site is not accessible and NOTIFY_MASTERS is TRUE.

Usage Notes

It is not necessary for either the old or new master definition site to be available when you call RELOCATE_MASTERDEF. In a planned reconfiguration, you should invoke RELOCATE_MASTERDEF with NOTIFY_MASTERS TRUE and INCLUDE_OLD_ MASTERDEF TRUE.

If just the master definition site fails, then you should invoke RELOCATE_ MASTERDEF with NOTIFY_MASTERS TRUE and INCLUDE_OLD_MASTERDEF

FALSE. If several master sites and the master definition site fail, then the administrator should invoke RELOCATE_MASTERDEF at each operational master with NOTIFY_MASTERS FALSE.

REMOVE_MASTER_DATABASES procedure

This procedure removes one or more master databases from a replicated environment. This procedure regenerates the triggers and their associated packages at the remaining master sites. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.REMOVE_MASTER_DATABASES (
   gname           IN   VARCHAR2,
   master_list     IN   VARCHAR2 |
   master_table    IN   DBMS_UTILITY.DBLINK_ARRAY);
```

Parameters

Table 34–103 REMOVE_MASTER_DATABASES Procedure Parameters

Parameter	Description
gname	Name of the object group associated with the replicated environment. This prevents confusion if a master database is involved in more than one replicated environment.
master_list	A comma-separated list of fully qualified master database names that you want to remove from the replicated environment. There must be no white space between names in the list.
master_table	In place of a list, you may also specify the database names in a PL/SQL table of type DBMS_UTILITY.DBLINK_ARRAY.

Exceptions

Table 34–104 REMOVE_MASTER_DATABASES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
nonmaster	At least one of the given databases is not a master site.
reconfigerror	One of the given databases is the master definition site.
commfailure	At least one remaining master site is not accessible.

REPCAT_IMPORT_CHECK procedure

This procedure ensures that the objects in the replicated object group have the appropriate object identifiers and status values after you perform an export/import of a replicated object or an object used by the advanced replication facility.

Syntax

```
DBMS_REPCAT.REPCAT_IMPORT_CHECK (
   gname      IN   VARCHAR2,
   master     IN   BOOLEAN);
```

Parameters

Table 34–105 REPCAT_IMPORT_CHECK Procedure Parameters

Parameter	Description
gname	Name of the replicated object group. If you omit both parameters, then the procedure checks all replicated object groups at your current site.
master	Set this to TRUE if you are checking a master site or FALSE if you are checking a snapshot site.

Exceptions

Table 34–106 REPCAT_IMPORT_CHECK Procedure Exceptions

Exception	Description
nonmaster	MASTER is TRUE and either the database is not a master site for the object group or the database is not the expected database.
nonsnapshot	MASTER is FALSE and the database is not a snapshot site for the object group.
missingobject	A valid replicated object in the object group does not exist.
missingrepgroup	The given group name does not exist.

RESUME_MASTER_ACTIVITY procedure

This procedure resumes normal replication activity after quiescing a replicated environment.

Syntax

```
DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
   gname       IN  VARCHAR2,
   override    IN  BOOLEAN := FALSE);
```

Parameters

Table 34–107 RESUME_MASTER_ACTIVITY Procedure Parameters

Parameter	Description
gname	Name of the replicated object group.
override	If this is TRUE, then it ignores any pending RepCat administration requests and restores normal replication activity at each master as quickly as possible. This should be considered only in emergency situations. If this is FALSE, then it restores normal replication activity at each master only when there is no pending RepCat administration request for gname at that master.

Exceptions

Table 34–108 RESUME_MASTER_ACTIVITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Replicated object group is not quiescing or quiesced.
commfailure	At least one master site is not accessible.

SEND_OLD_VALUES procedure

You have the option of sending old column values for each non-key column of a replicated table for updates and deletes. The default is to send old values for all columns. You can change this behavior at all master and snapshot sites by invoking DBMS_REPCAT.SEND_OLD_VALUES at the master definition site.

Syntax

```
DBMS_REPCAT.SEND_OLD_VALUES(
   sname           IN  VARCHAR2,
   oname           IN  VARCHAR2,
   { column_list   IN  VARCHAR2
   | column_table  IN  DBMS_REPCAT.VARCHAR2s,}
   operation       IN  VARCHAR2 := 'UPDATE',
   send            IN  BOOLEAN  := TRUE );
```

Note: This procedure is overloaded. The column_list and column_table parameters are mutually exclusive.

Parameters

Table 34–109 SEND_OLD_VALUES Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the replicated table.
column_list	A comma-separated list of the columns in the table. There must be no white space between entries.
column_table	Instead of a list, you can use a PL/SQL table of type DBMS_REPCAT.VARCHAR2S to contain the column names. The first column name should be at offset 1, the second at offset 2, and so on.
operation	Possible values are: UPDATE, DELETE, or the asterisk wildcard '*', which means update and delete.
send	If TRUE, then the old values of the specified columns are sent. If FALSE, then the old values of the specified columns are not sent. Unspecified columns and unspecified operations are not affected. The specified change takes effect at the master definition site as soon as min_communication is TRUE for the table. The change takes effect at a master site or at a snapshot site the next time replication support is generated at that site with min_communication TRUE.

Note: The operation parameter allows you to decide whether or not to transmit old values for non-key columns when rows are deleted or when non-key columns are updated. If you do not send the old value, then Oracle sends a NULL in place of the old value and assumes the old value is equal to the current value of the column at the target side when the update or delete is applied.

Read "Minimizing Data Propagation for Update Conflict Resolution" in the *Oracle8i Replication* manual before changing the default behavior of Oracle.

Exceptions

Table 34–110 SEND_OLD_VALUES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist as a table in the given schema awaiting row-level replication information.
missingcolumn	At least one column is not in the table.
notquiesced	Replicated object group has not been suspended.
typefailure	An illegal operation is given.

SET_COLUMNS procedure

To use an alternate column or group of columns, instead of the primary key, to determine which columns of a table to compare when using row-level replication. You must call this procedure from the master definition site.

> **See Also:** See "Using Multimaster Replication" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_REPCAT.SET_COLUMNS (
   sname           IN    VARCHAR2,
   oname           IN    VARCHAR2,
   { column_list   IN    VARCHAR2
   | column_table  IN    DBMS_UTILITY.NAME_ARRAY } );
```

> **Note:** This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

Parameters

Table 34–111 SET_COLUMNS Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the table.
column_list	A comma-separated list of the columns in the table that you want to use as a primary key. There must be no white space between entries.
column_table	Instead of a list, you can use a PL/SQL table of type DBMS_UTILITY.NAME_ARRAY to contain the column names. The first column name should be at offset 1, the second at offset 2, and so on.

Exceptions

Table 34–112 SET_COLUMNS Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Given object does not exist as a table in the given schema awaiting row-level replication information.
missingcolumn	At least one column is not in the table.

SUSPEND_MASTER_ACTIVITY procedure

This procedure suspends replication activity for an object group. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
   gname   IN   VARCHAR2);
```

Parameters

Table 34–113 SUSPEND_MASTER_ACTIVITY Procedure Parameters

Parameter	Description
gname	Name of the object group for which you want to suspend activity.

Exceptions

Table 34–114 SUSPEND_MASTER_ACTIVITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notnormal	Replicated object group is not in normal operation.
commfailure	At least one master site is not accessible.

Usage Notes

The current implementation of SUSPEND_MASTER_ACTIVITY quiesces all replicated object groups at each master site.

SWITCH_SNAPSHOT_MASTER procedure

This procedure changes the master database of a snapshot replicated object group to another master site. This procedure does a full refresh of the affected snapshots and regenerates the triggers and their associated packages as needed. This procedure does not push the queue to the old master site before changing masters.

Syntax

```
DBMS_REPCAT.SWITCH_SNAPSHOT_MASTER (
   gname              IN   VARCHAR2,
   master             IN   VARCHAR2);
```

Parameters

Table 34–115 SWITCH_SNAPSHOT_MASTER Procedure Parameters

Parameter	Description
gname	Name of the snapshot object group for which you want to change master sites.
master	Fully qualified database name of the new master database to use for the snapshot site.

Exceptions

Table 34–116 SWITCH_SNAPSHOT_MASTER Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
nonmaster	Given database is not a master site.
commfailure	Given database is not accessible.

UNREGISTER_SNAPSHOT_REPGROUP procedure

This procedure facilitates the administration of snapshots at their respective master sites by inserting / modifying / deleting from `registered_snapshot_groups`.

Syntax

```
DBMS_REPCAT.UNREGISTER_SNAPSHOT_REPGROUP (
   gname     IN   VARCHAR2,
   snapsite  IN   VARCHAR2);
```

Parameters

Table 34–117 UNREGISTER_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the snapshot object group to be unregistered.
snapsite	Global name of the snapshot site.

VALIDATE function

This function validates the correctness of key conditions of a multiple master replication environment. This is overloaded.

Syntax

```
DBMS_REPCAT.VALIDATE (
   gname                IN  VARCHAR2,
   check_genflags       IN  BOOLEAN := FALSE,
   check_valid_objs     IN  BOOLEAN := FALSE,
   check_links_sched    IN  BOOLEAN := FALSE,
   check_links          IN  BOOLEAN := FALSE,
   error_table          OUT dbms_repcat.validate_err_table )
  RETURN BINARY_INTEGER;

DBMS_REPCAT.VALIDATE (
   gname                IN  VARCHAR2,
   check_genflags       IN  BOOLEAN := FALSE,
   check_valid_objs     IN  BOOLEAN := FALSE,
   check_links_sched    IN  BOOLEAN := FALSE,
   check_links          IN  BOOLEAN := FALSE,
   error_msg_table      OUT DBMS_UTILITY.UNCL_ARRAY,
   error_num_table      OUT DBMS_UTILITY.NUMBER_ARRAY )
  RETURN BINARY_INTEGER;
```

Parameters

Table 34–118 VALIDATE Function Parameters

Parameter	Description
gname	Name of the master group to validate.
check_genflags	Check whether all the objects in the group are generated. This must be done at the masterdef site only.
check_valid_objs	Check that the underlying objects for objects in the group valid. This must be done at the masterdef site only. The masterdef site goes to all other sites and checks that the underlying objects are valid. The validity of the objects is checked within the schema of the connected user.
check_links_sched	Check whether the links are scheduled for execution. This should be invoked at each master site.

Table 34–118 VALIDATE Function Parameters

Parameter	Description
check_links	Check whether the connected user (repadmin), as well as the propagator, have correct links for replication to work properly. Checks that the links exist in the database and are accessible. This should be invoked at each master site.
error_table	Returns the message and numbers of all errors that are found.
error_msg_table	Returns the messages of all errors that are found.
error_num_table	Returns the numbers of all errors that are found.

Exceptions

Table 34–119 VALIDATE Function Exceptions

Exception	Description
missingdblink	Database link does not exist in the schema of the replication propagator or has not been scheduled. Ensure that the database link exists in the database, is accessible, and is scheduled for execution.
dblinkmismatch	Database link name at the local node does not match the global name of the database that the link accesses. Ensure that global names is set to true and the link name matches the global name.
dblinkuidmismatch	User name of the replication administration user at the local node and the user name at the node corresponding to the database link are not the same. Advanced replication expects the two users to be the same. Ensure that the user ID of the replication administration user at the local node and the user ID at the node corresponding to the database link are the same.
objectnotgenerated	Object has not been generated at other master sites or is still being generated. Ensure that the object is generated by calling generate_replication_support and do_deferred_repcat_admin for the object at the masterdef site.
opnotsupported	Operation is not supported if the object group is replicated at a pre-V8 node. Ensure that all nodes of the replicated object group are V8.

Usage Notes

The return value of VALIDATE is the number of errors found. The function's OUT parameter(s) returns any errors that are found. In the first interface function, the

ERROR_TABLE consists of an array of records. Each record has a VARCHAR2 and a NUMBER in it. The string field contains the error message and the number field contains the Oracle error number.

The second interface is similar except that there are two OUT arrays. A VARCHAR2 array with the error messages and a NUMBER array with the error numbers.

WAIT_MASTER_LOG procedure

This procedure determines whether changes that were asynchronously propagated to a master site have been applied.

Syntax

```
DBMS_REPCAT.WAIT_MASTER_LOG (
   gname          IN    VARCHAR2,
   record_count   IN    NATURAL,
   timeout        IN    NATURAL,
   true_count     OUT   NATURAL);
```

Parameters

Table 34–120 WAIT_MASTER_LOG Procedure Parameters

Parameter	Description
gname	Name of the replicated object group.
record_count	Procedure returns whenever the number of incomplete activities is at or below this threshold.
timeout	Maximum number of seconds to wait before the procedure returns.
true_count (out parameter)	Returns the number of incomplete activities.

Exceptions

Table 34–121 WAIT_MASTER_LOG Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.

35

DBMS_REPCAT_ADMIN

DBMS_REPCAT_ADMIN enables you to create users with the privileges needed by the symmetric replication facility.

Summary of Subprograms

Table 35–1 DBMS_REPCAT_ADMIN Package Subprograms

Subprogram	Description
GRANT_ADMIN_ANY_SCHEMA procedure on page 35-2	Grants the necessary privileges to the replication administrator to administer any replicated object group at the current site.
GRANT_ADMIN_SCHEMA procedure on page 35-3	Grants the necessary privileges to the replication administrator to administer a schema at the current site.
REGISTER_USER_REPGROUP procedure on page 3	Assigns proxy snapshot administrator or receiver privileges at the master site for use with remote sites.
REVOKE_ADMIN_ANY_SCHEMA procedure on page 35-5	Revokes the privileges and roles from the replication administrator that would be granted by GRANT_ADMIN_ANY_SCHEMA.
REVOKE_ADMIN_SCHEMA procedure on page 35-6	Revokes the privileges and roles from the replication administrator that would be granted by GRANT_ADMIN_SCHEMA.
UNREGISTER_USER_REPGROUP procedure on page 35-6	Revokes the privileges and roles from the proxy snapshot administrator or receiver that would be granted by the REGISTER_USER_REPGROUP procedure.

GRANT_ADMIN_ANY_SCHEMA procedure

This procedure grants the necessary privileges to the replication administrator to administer any replicated object group at the current site.

Syntax

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
   username  IN   VARCHAR2);
```

Parameters

Table 35–2 GRANT_ADMIN_ANY_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator to whom you want to grant the necessary privileges and roles to administer any replicated object groups at the current site.

Exceptions

Table 35–3 GRANT_ADMIN_ANY_REPGROUP Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

GRANT_ADMIN_SCHEMA procedure

This procedure grants the necessary privileges to the replication administrator to administer a schema at the current site. This procedure is most useful if your object group does not span schemas.

Syntax

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_SCHEMA (
   username IN VARCHAR2);
```

Parameters

Table 35–4 GRANT_ADMIN_REPSCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator. This user is then granted the necessary privileges and roles to administer the schema of the same name within a replicated object group at the current site.

Exceptions

Table 35–5 GRANT_ADMIN_REPSCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

REGISTER_USER_REPGROUP procedure

This procedure assigns proxy snapshot administrator or receiver privileges at the master site for use with remote sites. This procedure grants only the necessary privileges to the proxy snapshot administrator or receiver, avoiding having to grant the powerful privileges granted by the GRANT_ADMIN_SCHEMA or GRANT_ADMIN_ANY_SCHEMA procedures.

See Also: See "Advanced Techniques" in the *Oracle8i Replication* manual for more information on trusted versus untrusted security models.

Syntax

```
DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
   username          IN   VARCHAR2,
   privilege_type    IN   VARCHAR2,
   list_of_gnames    IN   VARCHAR2 |
   table_of_gnames   IN   dbms_utility.name_array);
```

Parameters

Table 35–6 REGISTER_USER_REPGROUP Procedure Parameters

Parameter	Description
username	Name of the user you are giving either proxy snapshot administrator or receiver privileges to.
privilege_type	Specifies the privilege type you are assigning. Use the following values for to define your privilege_type: RECEIVER for receiver privileges PROXY_SNAPADMIN for proxy snapadmin privileges.
list_of_gnames	Comma-separated list of object groups you want a user registered for receiver privileges. There must be no whitespace between entries in the list. If you set list_of_gnames to NULL, then the user is registered for all object groups, even object groups that are not yet known when this procedure is called. You must use named notation in order to set list_of_gnames to NULL. An invalid object group in the list causes registration to fail for the entire list. If you specify a value for list_of_gnames, then do not specify a value for table_of_gnames.
table_of_gnames	PL/SQL table of object groups you want a user registered for receiver privileges. The PL/SQL table must be of type DBMS_UTILITY.NAME_ARRAY. This table is 1-based. Use the single value NULL to register the user for all object groups. An invalid object group in the table causes registration to fail for the entire table. If you specify a value for table_of_gnames, then do not specify a value for list_of_gnames.

Exceptions

Table 35–7 REGISTER_USER_REPGROUP Procedure Exceptions

Exception	Description
nonmaster	Specified object group does not exist or the invocation database is not a master.
ORA-01917	User does not exist.
typefailure	Incorrect privilege type was specified.

REVOKE_ADMIN_ANY_SCHEMA procedure

This procedure revokes the privileges and roles from the replication administrator that would be granted by GRANT_ADMIN_ANY_SCHEMA.

> **Note:** Identical privileges and roles that were granted independently of GRANT_ADMIN_ANY_SCHEMA are also revoked.

Syntax

```
DBMS_REPCAT_ADMIN.REVOKE_ADMIN_ANY_SCHEMA (
   username IN VARCHAR2);
```

Parameters

Table 35–8 REVOKE_ADMIN_ANY_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator whose privileges you want to revoke.

Exceptions

Table 35–9 REVOKE_ADMIN_ANY_SCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

REVOKE_ADMIN_SCHEMA procedure

This procedure revokes the privileges and roles from the replication administrator that would be granted by GRANT_ADMIN_SCHEMA.

> **Note:** Identical privileges and roles that were granted independently of GRANT_ADMIN_SCHEMA are also revoked.

Syntax

```
DBMS_REPCAT_ADMIN.REVOKE_ADMIN_SCHEMA (
   username IN VARCHAR2);
```

Parameters

Table 35–10 REVOKE_ADMIN_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator whose privileges you want to revoke.

Exceptions

Table 35–11 REVOKE_ADMIN_SCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

UNREGISTER_USER_REPGROUP procedure

This procedure revokes the privileges and roles from the proxy snapshot administrator or receiver that would be granted by the REGISTER_USER_REPGROUP procedure.

Syntax

```
DBMS_REPCAT_ADMIN.UNREGISTER_USER_REPGROUP (
   username         IN   VARCHAR2,
   privilege_type   IN   VARCHAR2,
   list_of_gnames   IN   VARCHAR2 |
   table_of_gnames  IN   dbms_utility.name_array);
```

Parameters

Table 35–12 UNREGISTER_USER_REPGROUP Procedure Parameters

Parameter	Description
username	Name of the user you are unregistering.
privilege_type	Specifies the privilege type you are revoking. Use the following values for to define your privilege_type: RECEIVER for receiver privileges PROXY_SNAPADMIN for proxy snapadmin privileges.
list_of_gnames	Comma-separated list of object groups you want a user unregistered for receiver privileges. There must be no whitespace between entries in the list. If you set list_of_gnames to NULL, then the user is unregistered for all object groups registered. You must use named notation in order to set list_of_gnames to NULL. An invalid object group in the list causes unregistration to fail for the entire list. If you specify a value for list_of_gnames, then do not specify a value for table_of_gnames.
table_of_gnames	PL/SQL table of object groups you want a user unregistered for receiver privileges. The PL/SQL table must be of type DBMS_UTILITY.NAME_ARRAY. This table is 1-based. Use the single value NULL to unregister the user for all object groups registered. An invalid object group in the table causes unregistration to fail for the entire table. If you specify a value for table_of_gnames, then do not specify a value for list_of_gnames.

Exceptions

Table 35–13 UNREGISTER_USER_REPGROUP Procedure Exceptions

Exception	Description
nonmaster	Specified object group does not exist or the invocation database is not a master.
ORA-01917	User does not exist.
typefailure	Incorrect privilege type was specified.

36

DBMS_REPCAT_INSTANTIATE

The DBMS_REPCAT_INSTANTIATE package instantiates deployment templates.

Summary of Subprograms

Table 36–1 DBMS_REPCAT_INSTANTIATE Package Subprograms

Subprogram	Description
DROP_SITE_INSTANTIATION procedure on page 36-2	Public procedure that removes the target site from the DBA_REPCAT_TEMPLATE_SITES view.
INSTANTIATE_OFFLINE procedure on page 36-3	Public procedure that generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while offline.
INSTANTIATE_ONLINE procedure on page 36-5	Public procedure that generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while online.

DROP_SITE_INSTANTIATION procedure

This procedure drops a template instantiation at a target site. This removes all related meta data at the master site and disables the specified site from refreshing their snapshots. You must execute this procedure as the user that originally instantiated the template.

Note: To see who instantiated the template, query the REPCAT_TEMPLATE_SITES view.

Syntax

```
DBMS_REPCAT_INSTANTIATE.DROP_SITE_INSTANTIATION (
   refresh_template_name  IN   VARCHAR2,
   site_name              IN   VARCHAR2,
   repapi_site_id         IN   NUMBER := -1e-130);
```

Parameters

Table 36–2 DROP_SITE_INSTANTIATION Procedure Parameters

Parameter	Description
refresh_template_name	The name of the deployment template to be dropped.
site_name	Identifies the Oracle server site where you want to drop the specified template instantiation. (If you specify a site_name, then do not specify a repapi_site_id)

Table 36–2 DROP_SITE_INSTANTIATION Procedure Parameters

Parameter	Description
repapi_site_id	Identifies the REPAPI location where you want to drop the specified template instantiation. (If you specify a repapi_site_id, then do not specify a site_name.)

INSTANTIATE_OFFLINE procedure

This function generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while offline. This generated script should be used at remote snapshot sites that are *not* able to remain connected to the master site for an extended amount of time. This is an ideal solution where the remote snapshot site is a laptop. Use the packaging tool in Replication Manager to package the generated script and data into a single file, that can be posted on an FTP site or loaded to a CD-ROM, floppy disk, etc. See "Deploying Template" in the *Oracle8i Replication* manual for more information.

The script generated by this function is stored in the USER_REPCAT_TEMP_OUTPUT temporary view and is used by several Oracle tools, including Replication Manager, during the distribution of deployment templates. The number returned by this function is used to retrieve the appropriate information from the USER_REPCAT_TEMP_OUTPUT view.

The user that executes this public procedure will become the "registered" user of the instantiated template at the specified site.

Note: This procedure is used in performing an offline instantiation of a deployment template.

This procedure should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a snapshot). See these respective packages for more information on their usage.

Syntax

```
DBMS_REPCAT_INSTANTIATE.INSTANTIATE_OFFLINE(
   refresh_template_name   IN   VARCHAR2,
   site_name               IN   VARCHAR2,
   runtime_parm_id         IN   NUMBER   := -1e-130,
   next_date               IN   DATE     := SYSDATE,
   interval                IN   VARCHAR2 :'SYSDATe + 1')
  RETURN NUMBER;
```

Parameters

Table 36–3 INSTANTIATE_OFFLINE Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be instantiated.
site_name	Name of the remote site that is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, then specify the ID used when creating the runtime parameters. (The ID was retrieved by using the GET_RUNTIME_PARM_ID function.)
next_date	Specifies the next refresh date value to be used when creating the refresh group.
interval	Specifies the refresh interval to be used when creating the refresh group.

Exceptions

Table 36–4 INSTANTIATE_OFFLINE Function Exceptions

Exception	Description
miss_refresh_ template	The deployment template name specified is invalid or does not exist.
miss_user	The name of the authorized user is invalid or does not exist. Verify that the specified user is listed in the DBA_REPCAT_TEMPLATE_ AUTH view; if user is not listed, then the specified user is not authorized to instantiate the target deployment template.
bad_parms	All of the template parameters were not populated by the defined user parameter values and / or template default values. The number of pre-defined values may not have matched the number of template parameters or pre-defined value was invalid for the target parameter (i.e., type mismatch).

Returns

Table 36–5 INSTANTIATE_OFFLINE Function Returns

Return Value	Description
<*system generated number*>	Specify the generated system number for the output_id when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated instantiation script.

INSTANTIATE_ONLINE procedure

This function generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while online. This generated script should be used at remote snapshot sites that are able to remain connected to the master site for an extended amount of time, as the instantiation process at the remote snapshot site may be lengthy (depending on the amount of data that is populated to the new snapshots).

The script generated by this function is stored in the USER_REPCAT_TEMP_OUTPUT temporary view and is used by several Oracle tools, including Replication Manager, during the distribution of deployment templates. The number returned by this function is used to retrieve the appropriate information from the USER_REPCAT_ TEMP_OUTPUT view.

The user that executes this public procedure will become the "registered" user of the instantiated template at the specified site.

Syntax

```
DBMS_REPCAT_INSTANTIATE.INSTANTIATE_ONLINE(
   refresh_template_name   IN   VARCHAR2,
   site_name               IN   VARCHAR2,
   runtime_parm_id         IN   NUMBER   := -1e-130,
   next_date               IN   DATE     := SYSDATE,
   interval                IN   VARCHAR2 :'SYSDATe + 1')
  RETURN NUMBER;
```

Parameters

Table 36–6 Parameter for INSTANTIATE_ONLINE

Parameter	Description
refresh_template_name	Name of the deployment template to be instantiated.
site_name	Name of the remote site that is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, then specify the ID used when creating the runtime parameters (the ID was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	Specifies the next refresh date value to be used when creating the refresh group.
interval	Specifies the refresh interval to be used when creating the refresh group.

Exceptions

Table 36–7 Exception for INSTANTIATE_ONLINE

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist.
miss_user	The name of the authorized user is invalid or does not exist. Verify that the specified user is listed in the DBA_REPCAT_TEMPLATE_AUTH view; if user is not listed, then the specified user is not authorized to instantiate the target deployment template.
bad_parms	All of the template parameters were not populated by the defined user parameter values and / or template default values. The number of pre-defined values may not have matched the number of template parameters or pre-defined value was invalid for the target parameter (i.e., type mismatch).

Returns

Table 36–8 INSTANTIATE_ONLINE Function Returns

Return Value	Description
<*system generated number*>	Specify the generated system number for the output_id when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated instantiation script.

37

DBMS_REPCAT_RGT

DBMS_REPCAT_RGT controls the maintenance and definition of refresh group templates.

Summary of Subprograms

Table 37–1 DBMS_REPCAT_RGT Package Subprograms

Subprogram	Description
ALTER_REFRESH_TEMPLATE procedure on page 37-4	Lets the DBA alter existing deployment templates.
ALTER_TEMPLATE_OBJECT procedure on page 37-6	Alters objects that have been added to a specified deployment template.
ALTER_TEMPLATE_PARM procedure on page 37-9	Lets the DBA alter the parameters for a specific deployment template.
ALTER_USER_AUTHORIZATION procedure on page 37-10	Alters the contents of the DBA_REPCAT_TEMPLATE_AUTH view.
ALTER_USER_PARM_VALUE procedure on page 37-11	Changes existing parameter values that have been defined for a specific user.
COMPARE_TEMPLATES function on page 37-14	Lets a DBA compare the contents of two deployment templates.
COPY_TEMPLATE function on page 37-15	Lets the DBA copy a deployment template.
CREATE_OBJECT_FROM_EXISTING function on page 37-17	Creates a template object definition from existing database objects and adds it to a target deployment template.
CREATE_REFRESH_TEMPLATE function on page 37-18	Creates the deployment template, which allows you to define the template name, private/public status, and target refresh group.
CREATE_TEMPLATE_OBJECT function on page 37-21	Adds object definitions to a target deployment template container.
CREATE_TEMPLATE_PARM function on page 37-23	Creates parameters for a specific deployment template to allow custom data sets to be created at the remote snapshot site.
CREATE_USER_AUTHORIZATION function on page 37-26	Authorizes specific users to instantiate private deployment templates.
CREATE_USER_PARM_VALUE function on page 37-27	Pre-defines deployment template parameter values for specific users.
DELETE_RUNTIME_PARMS procedure on page 37-29	Deletes a runtime parameter value that you defined using the INSERT_RUNTIME_PARMS procedure.

Table 37–1 DBMS_REPCAT_RGT Package Subprograms

Subprogram	Description
DROP_ALL_OBJECTS procedure on page 37-29	Lets the DBA drop all objects or specific object types from a deployment template.
DROP_ALL_TEMPLATE_PARMS procedure on page 37-30	Lets the DBA drop template parameters for a specified deployment template.
DROP_ALL_TEMPLATE_SITES procedure on page 37-31	Removes all entries from the DBA_REPCAT_TEMPLATE_SITES view.
DROP_ALL_TEMPLATES procedure on page 37-32	Removes all deployment templates at the site where the procedure is called.
DROP_ALL_USER_AUTHORIZATIONS procedure on page 37-33	Lets the DBA drop all user authorizations for a specified deployment template.
DROP_ALL_USER_PARM_VALUES procedure on page 37-33	Drops user parameter values for a specific deployment template.
DROP_REFRESH_TEMPLATE procedure on page 37-34	Drops a deployment template.
DROP_SITE_INSTANTIATION procedure on page 37-35	Removes the target site from the DBA_REPCAT_TEMPLATE_SITES view.
DROP_TEMPLATE_OBJECT procedure on page 37-36	Removes a template object from a specific deployment template.
DROP_TEMPLATE_PARM procedure on page 37-37	Removes an existing template parameter from the DBA_REPCAT_TEMPLATE_PARMS view.
DROP_USER_AUTHORIZATION procedure on page 37-38	Removes a user authorization entry from the DBA_REPCAT_TEMPLATE_AUTH view.
DROP_USER_PARM_VALUE procedure on page 37-39	Removes a pre-defined user parameter value for a specific deployment template.
GET_RUNTIME_PARM_ID function on page 37-40	Retrieves an ID to be used when defining a runtime parameter value.
INSERT_RUNTIME_PARMS procedure on page 37-41	Defines runtime parameter values prior to instantiating a template.
INSTANTIATE_OFFLINE function on page 37-42	Generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while offline.

Table 37–1 DBMS_REPCAT_RGT Package Subprograms

Subprogram	Description
INSTANTIATE_ONLINE function on page 37-45	Generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while online.
LOCK_TEMPLATE_EXCLUSIVE procedure on page 47	Prevents users from reading or instantiating the template when a deployment template is being updated or modified.
LOCK_TEMPLATE_SHARED procedure on page 37-47	Makes a specified deployment template read-only.

ALTER_REFRESH_TEMPLATE procedure

This procedure lets the DBA alter existing deployment templates. Alterations may include defining a new deployment template name, a new refresh group, or a new owner and changing the public/private status.

Syntax

```
DBMS_REPCAT_RGT.ALTER_REFRESH_TEMPLATE (
   refresh_template_name      IN   VARCHAR2,
   new_owner                  IN   VARCHAR2 := '-',
   new_refresh_group_name     IN   VARCHAR2 := '-',
   new_refresh_template_name  IN   VARCHAR2 := '-',
   new_template_comment       IN   VARCHAR2 := '-',
   new_public_template        IN   VARCHAR2 := '-',
   new_last_modified          IN   DATE     := to_date('1', 'J'),
   new_modified_by            IN   NUMBER   := -1e-130);
```

Parameters

Table 37–2 ALTER_REFRESH_TEMPLATE Procedure Parameters

Parameter	Description
refresh_template_name	The name of the deployment template that you want to alter.
new_owner	The name of the new deployment template owner. Do not specify a value to keep the current owner.
new_refresh_group_name	If necessary, use this parameter to specify a new refresh group name that the template objects will be added to. Do not specify a value to keep the current refresh group.
new_refresh_template_name	Use this parameter to specify a new deployment template name. Do not specify a value to keep the current deployment template name.
new_template_comment	New deployment template comments. Do not specify a value to keep the current template comment.
new_public_template	Determines whether the deployment template is public or private. Only acceptable values are 'Y' and 'N' ('Y' = public and 'N' = private). Do not specify a value to keep the current value.
new_last_modified	Contains the date of the last modification made to this deployment template. If a value is not specified, then the current date is automatically used.
new_modified_by	Contains the name of the user who last modified this deployment template. If a value is not specified, then the current user is automatically used.

Exceptions

Table 37–3 ALTER_REFRESH_TEMPLATE Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
bad_public_template	The public_template parameter is specified incorrectly. The public_template parameter must be specified as a 'Y' for a public template or an 'N' for a private template.
dupl_refresh_template	A template with the specified name already exists. See the DBA_REPCAT_REFRESH_TEMPLATES view.

ALTER_TEMPLATE_OBJECT procedure

This procedure alters objects that have been added to a specified deployment template. The most common changes may include altering the object DDL and/or assigning the object to a different deployment template.

Changes made to the template is reflected only at new sites instantiating the deployment template. Remote sites that have already instantiated the template need to re-instantiate the deployment template to apply the changes.

Syntax

```
DBMS_REPCAT_RGT.ALTER_TEMPLATE_OBJECT (
   refresh_template_name       IN   VARCHAR2,
   object_name                 IN   VARCHAR2,
   object_type                 IN   VARCHAR2,
   new_refresh_template_name   IN   VARCHAR2 := '-',
   new_object_name             IN   VARCHAR2 := '-',
   new_object_type             IN   VARCHAR2 := '-',
   new_ddl_text                IN   CLOB     := '-',
   new_master_rollback_seg     IN   VARCHAR2 := '-',
   new_flavor_id               IN   NUMBER   := -1e-130);
```

Parameters

Table 37–4 ALTER_TEMPLATE_OBJECT Procedure Parameters

Parameter	Description
refresh_template_name	Deployment template name that contains the object that you want to alter.
object_name	Name of the template object that you want to alter.
object_type	Type of object that you want to alter.
new_refresh_template_name	Name of the new deployment template that you want to re-assign this object to. Do not specify a value to keep the object assigned to the current deployment template.
new_object_name	New name of the template object. Do not specify a value to keep the current object name.
new_object_type	If specified, then the new object type. Objects of the following type may be specified: SNAPSHOT PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM TRIGGER SEQUENCE DATABASE LINK
new_ddl_text	New object DDL for specified object. Do not specify any new DDL text to keep the current object DDL.
new_master_rollback_seg	New master rollback segment for specified object. Do not specify a value to keep the current rollback segment.
new_flavor_id	New flavor ID for the specified object. Do not specify a value to keep the current flavor ID.

Exceptions

Table 37–5 ALTER_TEMPLATE_OBJECT Procedure Exceptions

Exception	Description
miss_refresh_ template	Deployment template name specified is invalid or does not exist.
miss_flavor_id	Flavor ID specified is invalid or does not exist.
bad_object_type	Object type is specified incorrectly. See Table 37–4 for a list of valid object types.
miss_template_object	Template object name specified is invalid or does not exist.
dupl_template_object	New template name specified in the new_refresh_template_ name parameter already exists.

Usage Notes

Because the ALTER_TEMPLATE_OBJECT procedure utilizes a CLOB, you need to utilize the DBMS_LOB package when using the ALTER_TEMPLATE_OBJECT procedure. The following example illustrates how to use the DBMS_LOB package with the ALTER_TEMPLATE_OBJECT procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'CREATE SNAPSHOT snap_sales AS SELECT *
      FROM sales WHERE salesperson = :salesid and region_id = :region';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_TEMPLATE_OBJECT(
      refresh_template_name => 'rgt_personnel',
      object_name => 'SNAP_SALES',
      object_type => 'SNAPSHOT',
      new_ddl_text => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

ALTER_TEMPLATE_PARM procedure

This procedure allows the DBA to alter the parameters for a specific deployment template. Alterations may include renaming the parameter or redefining the default value and prompt string.

Syntax

```
DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM (
   refresh_template_name       IN   VARCHAR2,
   parameter_name              IN   VARCHAR2,
   new_refresh_template_name   IN   VARCHAR2 := '-',
   new_parameter_name          IN   VARCHAR2 := '-',
   new_default_parm_value      IN   CLOB     := NULL,
   new_prompt_string           IN   VARCHAR2 := '-',
   new_user_override           IN   VARCHAR2 := '-');
```

Parameters

Table 37–6 ALTER_TEMPLATE_PARM Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the parameter that you want to alter.
parameter_name	Name of the parameter that you want to alter.
new_refresh_template_name	Name of the deployment template that the specified parameter should be re-assigned to (useful when you want to move a parameter from one template to another). Do not specify a value to keep the parameter assigned to the current template.
new_parameter_name	New name of the template parameter. Do not specify a value to keep the current parameter name.
new_default_parm_value	New default value for the specified parameter. Do not specify a value to keep the current default value.
new_prompt_string	New prompt text for the specified parameter. Do not specify a value to keep the current prompt string.
new_user_override	Determines if the user can override the default value if prompted during the instantiation process (the user is prompted if no user parameter value has been defined for this parameter). Set this parameter to 'Y' to allow a user to override the default value or set this parameter to 'N' to not allow an override.

Exceptions

Table 37–7 ALTER_TEMPLATE_PARM Procedure Exceptions

Exception	Description
miss_refresh_ template	Deployment template name specified is invalid or does not exist.
miss_template_parm	Template parameter specified is invalid or does not exist.
dupl_template_parm	Combination of new_refresh_template_name and new_ parameter_name already exists.

Usage Notes

Because the ALTER_TEMPLATE_PARM procedure utilizes a CLOB, you need to utilize the DBMS_LOB package when using the ALTER_TEMPLATE_PARM procedure. The following example illustrates how to use the DBMS_LOB package with the ALTER_TEMPLATE_PARM procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM(
      refresh_template_name => 'rgt_personnel',
      parameter_name => 'region',
      new_default_parm_value => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

ALTER_USER_AUTHORIZATION procedure

This procedure alters the contents of the DBA_REPCAT_TEMPLATE_AUTH view. Specifically, you can change user / deployment template authorization assignments. This procedure is helpful, for example, if an employee moves positions and requires the snapshot environment of another deployment template; the DBA simply assigns the employee the new deployment template and the user is authorized to instantiate the target template.

Syntax

```
DBMS_REPCAT_RGT.ALTER_USER_AUTHORIZATION (
   user_name                    IN   VARCHAR2,
   refresh_template_name        IN   VARCHAR2,
   new_user_name                IN   VARCHAR2 := '-',
   new_refresh_template_name    IN   VARCHAR2 := '-');
```

Parameters

Table 37–8 ALTER_USER_AUTHORIZATION Procedure Parameters

Parameter	Description
user_name	Name of the user whose authorization you want to alter.
refresh_template_name	Name of the deployment template that is currently assigned to the specified user that you want to alter.
new_user_name	Use this parameter to define a new user for this template authorization. Do not specify a value to keep the current user
new_refresh_template_name	The deployment template that the specified user (either the existing or, if specified, the new user) is authorized to instantiate. Do not specify a value to keep the current deployment template.

Exceptions

Table 37–9 ALTER_USER_AUTHORIZATION Procedure Exceptions

Exception	Description
miss_user_authorization	The combination of user_name and refresh_template_name values specified does not exist in the DBA_REPCAT_TEMPLATE_AUTH view.
miss_user	The user name specified for the new_user_name or user_name parameter is invalid or does not exist.
miss_refresh_template	The deployment template specified for the new_refresh_template parameter is invalid or does not exist.
dupl_user_authorization	A row already exists for the specified user name and deployment template name. See the DBA_REPCAT_AUTH_TEMPLATES view.

ALTER_USER_PARM_VALUE procedure

This procedure changes existing parameter values that have been defined for a specific user. This procedure is especially helpful if your snapshot environment uses

assignment tables; simply change a user parameter value to quickly and securely change the data set of a remote snapshot site.

See "Deployment Template Design" in the *Oracle8i Replication* manual for more information on using assignment tables.

Syntax

```
DBMS_REPCAT_RGT.ALTER_USER_PARM_VALUE(
   refresh_template_name         IN   VARCHAR2,
   parameter_name                IN   VARCHAR2,
   user_name                     IN   VARCHAR2,
   new_refresh_template_name     IN   VARCHAR2 := '-',
   new_parameter_name            IN   VARCHAR2 := '-',
   new_user_name                 IN   VARCHAR2 := '-',
   new_parm_value                IN   CLOB     := NULL);
```

Parameters

Table 37–10 ALTER_USER_PARM_VALUE Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the user parameter value that you want to alter.
parameter_name	Name of the parameter that you want to alter.
user_name	Name of the user whose parameter value you want to alter.
new_refresh_template_name	Name of the deployment template that the specified user parameter value should be re-assigned to (useful when you are authorizing a user for a different template). Do not specify a value to keep the parameter assigned to the current template.
new_parameter_name	The new template parameter name. Do not specify a value to keep the user value defined for the existing parameter.
new_user_name	The new user name that this parameter value is for. Do not specify a value to keep the parameter value assigned to the current user.
new_parm_value	The new parameter value for the specified user parameter. Do not specify a value to keep the current parameter value.

Exceptions

Table 37–11 ALTER_USER_PARM_VALUE Procedure Exceptions

Exception	Description
miss_refresh_ template	Deployment template name specified is invalid or does not exist.
miss_template_parm	Template parameter specified is invalid or does not exist.
miss_user	User name specified for the user_name or new_user_name parameters is invalid or does not exist.
miss_user_parm_ values	User parameter value specified does not exist.
dupl_user_parm_ values	New user parameter specified already exists.

Usage Notes

Because the ALTER_USER_PARM_VALUE procedure utilizes a CLOB, you need to utilize the DBMS_LOB package when using the ALTER_USER_PARM_VALUE procedure. The following example illustrates how to use the DBMS_LOB package with the ALTER_USER_PARM_VALUE procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_USER_PARM_VALUE(
      refresh_template_name => 'rgt_personnel',
      parameter_name => 'region',
      user_name => 'BOB',
      new_parm_value => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

COMPARE_TEMPLATES function

This function allows a DBA to compare the contents of two deployment templates. Any discrepancies between the two deployment templates is stored in the USER_REPCAT_TEMP_OUTPUT table.

The COMPARE_TEMPLATES function returns a number that you specify in the WHERE clause when querying the USER_REPCAT_TEMP_OUTPUT table. For example, if the COMPARE_TEMPLATES procedure returns the number 10, then you would execute the following SELECT statement to view all discrepancies between two specified templates (your SELECT statement returns no rows if the templates are identical):

```
SELECT text FROM USER_REPCAT_TEMP_OUTPUT
   WHERE output_id =  10 ORDER BY LINE;
```

The contents of the USER_REPCAT_TEMP_OUTPUT are lost after you disconnect or a ROLLBACK has been performed.

Syntax

```
DBMS_REPCAT_RGT.COMPARE_TEMPLATES (
   source_template_name    IN   VARCHAR2,
   compare_template_name   IN   VARCHAR2)
  RETURN NUMBER;
```

Parameters

Table 37–12 COMPARE_TEMPLATES Function Parameters

Parameter	Description
source_template_name	Name of the first deployment template to be compared.
compare_template_name	Name of the second deployment template to be compared.

Exceptions

Table 37–13 COMPARE_TEMPLATES Function Exceptions

Exception	Description
miss_refresh_template	The deployment template name to be compared is invalid or does not exist.

Returns

Table 37–14 COMPARE_TEMPLATES Function Returns

Return Value	Description
<system generated number>	Specify the number returned for the output_id value when you select from the USER_REPCAT_TEMP_OUTPUT view to view the discrepancies between the compared templates.

COPY_TEMPLATE function

This function allows the DBA to copy a deployment template. COPY_TEMPLATE is helpful when a new deployment template will use many of the object contained in an existing deployment template. This function copies the deployment template, template objects, template parameters, and user parameter values. The DBA can optionally have the function copy the user authorizations for this template. The number returned by this function is used internally by Oracle to manage deployment templates.

> **Note:** The values in the DBA_REPCAT_TEMPLATE_SITES view are not copied.

This function also allows the DBA to copy a deployment template to another master site, which is helpful for deployment template distribution and to split network loads between multiple sites.

Syntax

```
DBMS_REPCAT_RGT.COPY_TEMPLATE (
   old_refresh_template_name     IN   VARCHAR2,
   new_refresh_template_name     IN   VARCHAR2,
   copy_user_authorizations      IN   VARCHAR2,
   dblink                        IN   VARCHAR2 := NULL)
  RETURN NUMBER;
```

Parameters

Table 37–15 COPY_TEMPLATE Function Parameters

Parameter	Description
old_refresh_ template_name	Name of the deployment template to be copied.
new_refresh_ template_name	Name of the new deployment template.
copy_user_ authorizations	Specifies whether the template authorizations for the original template should be copied for the new deployment template. Valid values for this parameter are 'Y', 'N' and NULL. **Note**: All users must exist at the target database.
dblink	Optionally defines where the deployment template should be copied from (this is helpful to distribute deployment templates to other master sites). If none is specified, then the deployment template is copied from the local master site.

Exceptions

Table 37–16 COPY_TEMPLATE Function Exceptions

Exception	Description
miss_refresh_ template	Deployment template name to be copied is invalid or does not exist.
dupl_refresh_ template	Name of the new refresh template specified already exists.
bad_copy_auth	Value specified for the copy_user_authorization parameter is invalid. Valid values are 'Y', 'N', and NULL.

Returns

Table 37–17 COPY_TEMPLATES Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

CREATE_OBJECT_FROM_EXISTING function

This function creates a template object definition from existing database objects and adds it to a target deployment template. The object DDL that created the original database object is executed when the target deployment template is instantiated at the remote snapshot site. This is ideal for adding existing triggers and procedures to your template. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_OBJECT_FROM_EXISTING(
   refresh_template_name  IN   VARCHAR2,
   object_name            IN   VARCHAR2,
   sname                  IN   VARCHAR2,
   oname                  IN   VARCHAR2,
   otype                  IN   VARCHAR2)
  RETURN NUMBER;
```

Parameters

Table 37–18 CREATE_OBJECT_FROM_EXISTING Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that you want to add this object to.
object_name	If necessary, the new name of the existing object that you are adding to your deployment template (allows you to define a new name for an existing object).
sname	The schema that contains the object that you are creating your template object from.
oname	Name of the object that you are creating your template object from.
otype	The type of database object that you are adding to the template (i.e., PROCEDURE, TRIGGER, etc.). The object type must be specified using the following numerical identifiers (DATABASE LINK or SNAPSHOT are not a valid object types for this function): SEQUENCE PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM TRIGGER

Exceptions

Table 37–19 CREATE_OBJECT_FROM_EXISTING Function Exceptions

Exception	Description
miss_refresh_ template	The specified refresh template name is invalid or missing. Query the DBA_REPCAT_REFRESH_TEMPLATE view for a list of existing deployment templates.
bad_object_type	The object type is specified incorrectly (see Table 37–24 for more information).
dupl_template_object	An object of the same name and type has already been added to the specified deployment template.
objectmissing	Existing object specified does not exist.

Returns

Table 37–20 CREATE_OBJECT_FROM_EXISTING Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

CREATE_REFRESH_TEMPLATE function

This function creates the deployment template, which allows you to define the template name, private / public status, and target refresh group. Each time that you create a template object, user authorization, or template parameter, you reference the deployment template created with this function. This function adds a row to the DBA_REPCAT_REFRESH_TEMPLATES view. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_REFRESH_TEMPLATE (
   owner                  IN   VARCHAR2,
   refresh_group_name     IN   VARCHAR2,
   refresh_template_name  IN   VARCHAR2,
   template_comment       IN   VARCHAR2 := NULL,
   public_template        IN   VARCHAR2 := NULL,
   last_modified          IN   DATE     := SYSDATE,
   modified_by            IN   VARCHAR2 := USER,
   creation_date          IN   DATE     := SYSDATE,
   created_by             IN   VARCHAR2 := USER)
  RETURN NUMBER
```

Parameters

Table 37–21 CREATE_REFRESH_TEMPLATE Function Parameters

Parameter	Description
owner	User name of the deployment template owner is specified with this parameter. If an owner is not specified, then the name of the user creating the template is automatically used.
refresh_group_name	Name of the refresh group that is created when this template is instantiated. All objects created by this template are assigned to the specified refresh group.
refresh_template_name	Name of the deployment template that you are creating. This name is referenced in all activities that involve this deployment template.
template_comment	User comments defined with this are listed in the DBA_REPCAT_REFRESH_TEMPLATES view.
public_template	Specifies whether the deployment template is public or private. Only acceptable values are 'Y' and 'N' ('Y' = public and 'N' = private).
last_modified	The date of the last modification made to this deployment template. If a value is not specified, then the current date is automatically used.
modified_by	Name of the user who last modified this deployment template. If a value is not specified, then the current user is automatically used.
creation_date	The date that this deployment template was created. If a value is not specified, then the current date is automatically used.
created_by	Name of the user who created this deployment template. If a value is not specified, then the current user is automatically used.

Exceptions

Table 37–22 CREATE_REFRESH_TEMPLATE Function Exceptions

Exception	Description
dupl_refresh_template	A template with the specified name already exists. See the DBA_REPCAT_REFRESH_TEMPLATES view to see a list of existing templates.
bad_public_template	The public_template parameter is specified incorrectly. The public_template parameter must be specified as a 'Y' for a public template or an 'N' for a private template.

Returns

Table 37–23 CREATE_REFRESH_TEMPLATE Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

CREATE_TEMPLATE_OBJECT function

This function adds object definitions to a target deployment template container. The specified object DDL is executed when the target deployment template is instantiated at the remote snapshot site. In addition to adding snapshots, this function can add tables, procedures, and other objects to your template. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
   refresh_template_name  IN   VARCHAR2,
   object_name            IN   VARCHAR2,
   object_type            IN   VARCHAR2,
   ddl_text               IN   CLOB,
   master_rollback_seg    IN   VARCHAR2 := NULL,
   flavor_id              IN   NUMBER   := -1e-130)
  RETURN NUMBER;
```

Parameters

Table 37–24 CREATE_TEMPLATE_OBJECT Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that you want to add this object to.
object_name	Name of the template object that you are creating.
object_type	The type of database object that you are adding to the template (i.e., SNAPSHOT, TRIGGER, PROCEDURE, etc.). Objects of the following type may be specified: SNAPSHOT, INDEX, TABLE, VIEW, SYNONYM, SEQUENCE, TRIGGER, PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, MATERIALIZED VIEW, DATABASE LINK
ddl_text	Contains the DDL that creates the object that you are adding to the template. Be sure to end your DDL with a semi-colon. (Remember, you can use a colon (:) to create a template parameter for your template object; see "Creating Snapshots with Deployment Templates" in the *Oracle8i Replication* book for more information.
master_rollback_seg	Specifies the name of the rollback segment to use when executing the defined object DDL at the remote snapshot site.
flavor_id	Defines the flavor ID for this template object.

Exceptions

Table 37–25 CREATE_TEMPLATE_OBJECT Function Exceptions

Exception	Description
miss_refresh_template	Specified refresh template name is invalid or missing. Query the DBA_REPCAT_REFRESH_TEMPLATE view for a list of existing deployment templates.
bad_object_type	Object type is specified incorrectly. See Table 37–24 for a list of valid object types.
dupl_template_object	An object of the same name and type has already been added to the specified deployment template.

Returns

Table 37–26 CREATE_TEMPLATE_OBJECT Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

Usage Notes

Because CREATE_TEMPLATE_OBJECT utilizes a CLOB, you need to utilize the DBMS_LOB package when using the CREATE_TEMPLATE_OBJECT function. The following example illustrates how to use the DBMS_LOB package with the CREATE_TEMPLATE_OBJECT function:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'CREATE SNAPSHOT snap_sales AS SELECT *
        FROM sales WHERE salesperson = :salesid';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT(
        refresh_template_name => 'rgt_personnel',
        object_name => 'snap_sales',
        object_type => 'SNAPSHOT',
        ddl_text => templob,
        master_rollback_seg => 'RBS');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

CREATE_TEMPLATE_PARM function

This function creates parameters for a specific deployment template to allow custom data sets to be created at the remote snapshot site. This function is only required when the DBA wants to define a set of template variables before adding any template objects (when objects are added to the template using the CREATE_TEMPLATE_OBJECT function, any variables in the object DDL are automatically added to the DBA_REPCAT_TEMPLATE_PARMS view).

The DBA typically uses the ALTER_TEMPLATE_PARM function to modify the default parameter values and/or prompt strings (see ALTER_TEMPLATE_PARM procedure on page 37-9 for more information). The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_TEMPLATE_PARM (
   refresh_template_name  IN   VARCHAR2,
   parameter_name         IN   VARCHAR2,
   default_parm_value     IN   CLOB     := NULL,
   prompt_string          IN   VARCHAR2 := NULL,
   user_override          IN   VARCHAR2 := NULL)
  RETURN NUMBER;
```

Parameters

Table 37–27 CREATE_TEMPLATE_PARM Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that you want to create the parameter for.
parameter_name	Name of the parameter you are creating.
default_parm_value	Default values for this parameter are defined using this parameter. If a user parameter value or runtime parameter value is not present, then this default value is used during the instantiation process.
prompt_string	The descriptive prompt text that is displayed for this template parameter during the instantiation process.
user_override	Determines if the user can override the default value if prompted during the instantiation process (the user is prompted if no user parameter value has been defined for this parameter). Set this parameter to 'Y' to allow a user to override the default value or set this parameter to 'N' to not allow an override.

Exceptions

Table 37–28 CREATE_TEMPLATE_PARM Function Exceptions

Exception	Description
miss_refresh_ template	The specified refresh template name is invalid or missing.
dupl_template_parm	A parameter of the same name has already been defined for the specified deployment template.

Returns

Table 37–29 CREATE_TEMPLATE_PARM Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

Usage Notes

Because the CREATE_TEMPLATE_PARM function utilizes a CLOB, you need to utilize the DBMS_LOB package when using the CREATE_TEMPLATE_PARM function. The following example illustrates how to use the DBMS_LOB package with the CREATE_ TEMPLATE_PARM function:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_PARM(
        refresh_template_name => 'rgt_personnel',
        parameter_name => 'region',
        default_parm_value => templob,
        prompt_string => 'Enter your region ID:',
        user_override => 'Y');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

CREATE_USER_AUTHORIZATION function

This function authorizes specific users to instantiate private deployment templates. Users not authorized for a private deployment template are not able to instantiate the private template. This function adds a row to the DBA_REPCAT_AUTH_TEMPLATES view.

Before you authorize a user, verify that the user exists at the master site where the user will instantiate the deployment template. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_USER_AUTHORIZATION (
   user_name               IN   VARCHAR2,
   refresh_template_name   IN   VARCHAR2)
  RETURN NUMBER;
```

Parameters

Table 37–30 CREATE_USER_AUTHORIZATION Function Parameters

Parameter	Description
user_name	Name of the user that you want to authorize to instantiate the specified template. Specify multiple users by separating user names with a comma (i.e., 'john, mike, bob')
refresh_template_name	Name of the template that you want to authorize the specified user to instantiate.

Exceptions

Table 37–31 CREATE_USER_AUTHORIZATION Function Exceptions

Exception	Description
miss_user	User name supplied is invalid or does not exist.
miss_refresh_template	Refresh template name supplied is invalid or does not exist.
dupl_user_authorization	An authorization has already been created for the specified user and deployment template. See the DBA_REPCAT_AUTH_TEMPLATES view for a listing of template authorizations.

Returns

Table 37–32 CREATE_USER_AUTHORIZATION Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

CREATE_USER_PARM_VALUE function

This function is used to pre-define deployment template parameter values for specific users. For example, if you want to pre-defined the region parameter as WEST for user 33456, then you would use the this function.

Any values specified with this function take precedence over default values specified for the template parameter. The number returned by this function is used internally by Oracle to manage deployment templates.

Syntax

```
DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE (
   refresh_template_name    IN   VARCHAR2,
   parameter_name           IN   VARCHAR2,
   user_name                IN   VARCHAR2,
   parm_value               IN   CLOB := NULL)
  RETURN NUMBER;
```

Parameters

Table 37–33 CREATE_USER_PARM_VALUE Function Parameters

Parameter	Description
refresh_template_name	Specifies the name of the deployment template that contains the parameter you are creating a user value for.
parameter_name	Name of the template parameter that you are defining a user parameter value for.
user_name	Specifies the name of the user that you are pre-defining a parameter value for.
parm_value	The pre-defined parameter value that will be used during the instantiation process initiated by the specified user.

Exceptions

Table 37–34 CREATE_USER_PARM_VALUE Function Exceptions

Exception	Description
miss_refresh_ template	Specified deployment template name is invalid or missing.
dupl_user_parm_ values	A parameter value for the specified user, parameter, and deployment template has already been defined. Query the DBA_ REPCAT_USER_PARMS view for a listing of existing user parameter values.
miss_template_parm	Specified deployment template parameter name is invalid or missing.
miss_user	Specified user name is invalid or missing.

Returns

Table 37–35 CREATE_USER_PARM_VALUE Function Returns

Return Value	Description
<system generated number>	System generated number is used internally by Oracle.

Usage Notes

Because the CREATE_USER_PARM_VALUE function utilizes a CLOB, you need to utilize the DBMS_LOB package when using the this function. The following example illustrates how to use the DBMS_LOB package with the CREATE_USER_PARM_ VALUE function:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE(
        refresh_template_name => 'rgt_personnel',
        parameter_name => 'region',
        user_name => 'BOB',
        user_parm_value => templob);
```

```
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

DELETE_RUNTIME_PARMS procedure

Use this procedure before instantiating a deployment template to delete a runtime parameter value that you defined using the INSERT_RUNTIME_PARMS procedure.

Syntax

```
DBMS_REPCAT_RGT.DELETE_RUNTIME_PARMS(
   runtime_parm_id    IN   NUMBER,
   parameter_name     IN   VARCHAR2);
```

Parameters

Table 37–36 DELETE_RUNTIME_PARMS Procedure Parameters

Parameter	Description
runtime_parm_id	Specifies the ID that you previously assigned the runtime parameter value to (this value was retrieved using the GET_RUNTIME_PARM_ID function).
parameter_name	Specifies the name of the parameter value that you want to drop (query the DBA_REPCAT_TEMPLATE_PARMS for a list of deployment template parameters).

Exceptions

Table 37–37 DELETE_RUNTIME_PARMS Procedure Exceptions

Exception	Description
miss_template_parm	The specified deployment template parameter name is invalid or missing.

DROP_ALL_OBJECTS procedure

This procedure allows the DBA to drop all objects or specific object types from a deployment template.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_OBJECTS (
   refresh_template_name   IN   VARCHAR2,
   object_type             IN   VARCHAR2 := NULL);
```

Parameters

Table 37–38 DROP_ALL_OBJECTS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the objects that you want to drop.
object_type	If NULL, then all objects in the template are dropped. If an object type is specified, then only objects of that type are dropped. Objects of the following type may be specified: SNAPSHOT, PROCEDURE INDEX, FUNCTION TABLE, PACKAGE VIEW, PACKAGE BODY SYNONYM, MATERIALIZED VIEW SEQUENCE, DATABASE LINK TRIGGER

Exceptions

Table 37–39 DROP_ALL_OBJECTS Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.
bad_object_type	Object type is specified incorrectly. See Table 37–38 for a list of valid object types.

DROP_ALL_TEMPLATE_PARMS procedure

This procedure allows the DBA to drop template parameters for a specified deployment template. The DBA can use this procedure to drop all parameters that

are not referenced by a template objects or drop all objects that reference a parameter and the parameters themselves.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_TEMPLATE_PARMS (
   refresh_template_name   IN   VARCHAR2,
   drop_objects            IN   VARCHAR2 := N);
```

Parameters

Table 37–40 DROP_ALL_TEMPLATE_PARMS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the parameters that you want to drop.
drop_objects	If no value is specified, then this defaults to N, which drops all parameters not referenced by a template object. If Y is specified, then all objects that reference a template parameter and the template parameters themselves are dropped.

Exceptions

Table 37–41 DROP_ALL_TEMPLATE_PARMS Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_ALL_TEMPLATE_SITES procedure

This procedure removes all entries from the DBA_REPCAT_TEMPLATE_SITES view, which keeps a record of sites that have instantiated a particular deployment template.

Caution: This is a dangerous procedure that cannot be undone. Additionally, Oracle Lite sites that have instantiated the dropped template will no longer be able to refresh their snapshots.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_TEMPLATE_SITES (
   refresh_template_name   IN   VARCHAR2);
```

Parameters

Table 37–42 DROP_ALL_TEMPLATE_SITES Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the sites that you want to drop.

Exceptions

Table 37–43 DROP_ALL_TEMPLATE_SITES Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_ALL_TEMPLATES procedure

This procedure removes all deployment templates at the site where the procedure is called.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_TEMPLATES;
```

Parameters

None.

DROP_ALL_USER_AUTHORIZATIONS procedure

This procedure allows the DBA to drop all user authorizations for a specified deployment template. Executing this procedure removes rows from the DBA_REPCAT_AUTH_TEMPLATES view.

This procedure might be implemented after converting a private template to a public template and the user authorizations are no longer required.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_USER_AUTHORIZATIONS (
   refresh_template_name   IN   VARCHAR2);
```

Parameters

Table 37–44 DROP_ALL_USER_AUTHORIZATIONS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the objects that you want to drop.

Exceptions

Table 37–45 DROP_ALL_USER_AUTHORIZATIONS Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_ALL_USER_PARM_VALUES procedure

This procedure drops user parameter values for a specific deployment template. This procedure is very flexible in allowing the DBA to define a set of user parameter values to be deleted. For example, defining the following parameters have the effect:

refresh_template_name: drops all user parameters for the specified deployment template.

refresh_template_name, user_name: drops all of the specified user parameters for the specified deployment template.

refresh_template_name, parameter_name: drops all user parameter values for the specified deployment template parameter.

refresh_template_name, parameter_name, user_name: drops the specified user's value for the specified deployment template parameter (equivalent to DROP_USER_PARM).

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_USER_PARMS (
   refresh_template_name   IN   VARCHAR2,
   user_name               IN   VARCHAR2,
   parameter_name          IN   VARCHAR2);
```

Parameters

Table 37–46 DROP_ALL_USER_PARMS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the parameter values that you want to drop.
user_name	Name of the user whose parameter values you want to drop.
parameter_name	Template parameter that contains the values that you want to drop.

Exceptions

Table 37–47 DROP_ALL_USER_PARMS Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_user	User name specified is invalid or does not exist.
miss_user_parm_values	Deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM view.

DROP_REFRESH_TEMPLATE procedure

This procedure drops a deployment template. Dropping a deployment template has a cascading effect, removing all related template parameters, user authorizations, template objects, and user parameters (this procedure does not drop template sites).

Syntax

```
DBMS_REPCAT_RGT.DROP_REFRESH_TEMPLATE (
   refresh_template_name  IN   VARCHAR2);
```

Parameters

Table 37–48 DROP_REFRESH_TEMPLATE Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be dropped.

Exceptions

Table 37–49 DROP_REFRESH_TEMPLATE Procedure Exceptions

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist. Query the DBA_REPCAT_REFRESH_TEMPLATE view for a list of deployment templates.

DROP_SITE_INSTANTIATION procedure

Purpose

This procedure drops a template instantiation at a target site. This procedure removes all related meta data at the master site and disables the specified site from refreshing their snapshots.

Syntax

```
DBMS_REPCAT_RGT.DROP_SITE_INSTANTIATION(
   refresh_template_name  IN   VARCHAR2,
   user_name              IN   VARCHAR2,
   site_name              IN   VARCHAR2,
   repapi_site_id         IN   NUMBER := -1e-130);
```

Parameters

Table 37–50 Parameter for DROP_SITE_INSTANTIATION

Parameter	Description
refresh_template_name	The name of the deployment template to be dropped.
user_name	Enter the name of the user that originally instantiated the template at the remote snapshot site. Query the REPCAT_TEMPLATE_SITES view to see the users that instantiated templates.
site_name	Identifies the Oracle server site where you want to drop the specified template instantiation. (If you specify a site_name, then do not specify a repapi_site_id).
repapi_site_id	Identifies the REPAPI location where you want to drop the specified template instantiation. (If you specify a repapi_site_id, then do not specify a site_name).

DROP_TEMPLATE_OBJECT procedure

This procedure removes a template object from a specific deployment template. For example, a DBA would use this procedure to remove an outdated snapshot from a deployment template. Changes made to the template are reflected at new sites instantiating the deployment template. Remote sites that have already instantiated the template need to re-instantiate the deployment template to apply the changes.

Syntax

```
DBMS_REPCAT_RGT.DROP_TEMPLATE_OBJECT (
   refresh_template_name  IN   VARCHAR2,
   object_name            IN   VARCHAR2,
   object_type            IN   VARCHAR2);
```

Parameters

Table 37–51 DROP_TEMPLATE_OBJECT Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that you are dropping the object from.
object_name	Name of the template object to be dropped.

Table 37–51 DROP_TEMPLATE_OBJECT Procedure Parameters

Parameter	Description
object_type	The type of object that is to be dropped. Objects of the following type may be specified: SNAPSHOT PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM MATERIALIZED VIEW SEQUENCE DATABASE LINK TRIGGER

Exceptions

Table 37–52 DROP_TEMPLATE_OBJECT Procedure Exceptions

Exception	Description
miss_refresh_ template	The deployment template name specified is invalid or does not exist.
miss_template_object	The template object specified is invalid or does not exist. Query the DBA_REPCAT_TEMPLATE_OBJECT view to see a list of deployment template objects.

DROP_TEMPLATE_PARM procedure

This procedure removes an existing template parameter from the DBA_REPCAT_ TEMPLATE_PARMS view. This procedure is helpful when you have dropped a template object and a particular parameter is no longer needed.

Syntax

```
DBMS_REPCAT_RGT.DROP_TEMPLATE_PARM (
   refresh_template_name  IN   VARCHAR2,
   parameter_name         IN   VARCHAR2);
```

Parameters

Table 37–53 DROP_TEMPLATE_PARM Procedure Parameters

Parameter	Description
refresh_template_name	The deployment template name that has the parameter that you want to drop
parameter_name	Name of the parameter that you want to drop.

Exceptions

Table 37–54 DROP_TEMPLATE_PARM Procedure Exceptions

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist.
miss_template_parm	The parameter name specified is invalid or does not exist. Query the DBA_REPCAT_TEMPLATE_PARMS view to see a list of template parameters.

DROP_USER_AUTHORIZATION procedure

This procedure removes a user authorization entry from the DBA_REPCAT_TEMPLATE_AUTH view. This procedure is used when removing a user's template authorization. If a user's authorization is removed, then the user is no longer able to instantiate the target deployment template.

> **See Also:** DROP_ALL_USER_AUTHORIZATIONS procedure on page 37-33.

Syntax

```
DBMS_REPCAT_RGT.DROP_USER_AUTHORIZATION (
   refresh_template_name   IN   VARCHAR2,
   user_name               IN   VARCHAR2);
```

Parameters

Table 37–55 DROP_USER_AUTHORIZATION Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that the user's authorization is being removed from.
user_name	Name of the user whose authorization is being removed.

Exceptions

Table 37–56 DROP_USER_AUTHORIZATION Procedure Exceptions

Exception	Description
miss_user	Specified user name is invalid or does not exist.
miss_user_authorization	Specified user and deployment template combination does not exist. Query the DBA_REPCAT_TEMPLATE_AUTH view to see a list of user / deployment template authorizations.
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_USER_PARM_VALUE procedure

This procedure removes a pre-defined user parameter value for a specific deployment template. This procedure is often executed after a user's template authorization has been removed.

Syntax

```
DBMS_REPCAT_RGT.DROP_USER_PARM_VALUE (
   refresh_template_name    IN   VARCHAR2,
   parameter_name           IN   VARCHAR2,
   user_name                IN   VARCHAR2);
```

Parameters

Table 37–57 DROP_USER_PARM_VALUE Procedure Parameters

Parameter	Description
refresh_template_name	Deployment template name that contains the parameter value that you want to drop.

Table 37–57 DROP_USER_PARM_VALUE Procedure Parameters

Parameter	Description
parameter_name	Parameter name that contains the pre-defined value that you want to drop.
user_name	Name of the user whose parameter value you want to drop.

Exceptions

Table 37–58 DROP_USER_PARM_VALUE Procedure Exceptions

Exception	Description
miss_refresh_ template	Deployment template name specified is invalid or does not exist.
miss_user	User name specified is invalid or does not exist.
miss_user_parm_ values	Deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM view.

GET_RUNTIME_PARM_ID function

This function retrieves an ID to be used when defining a runtime parameter value. All runtime parameter values are assigned to this ID and are also used during the instantiation process.

Syntax

```
DBMS_REPCAT_RGT.GET_RUNTIME_PARM_ID
  RETURN NUMBER;
```

Parameters

None.

Returns

Table 37–59 GET_RUNTIME_PARM_ID Function Returns

Return Value	Corresponding Datatype
<system generated number>	Runtime parameter values are assigned to the system generated number and is also used during the instantiation process.

INSERT_RUNTIME_PARMS procedure

This procedure defines runtime parameter values prior to instantiating a template. This procedure should be used to define parameter values when no user parameter values have been defined and you do not want to accept the default parameter values.

Before using the this procedure, be sure to execute the GET_RUNTIME_PARM_ID function to retrieve a parameter ID to be used when inserting a runtime parameter. This ID is used for defining runtime parameter values and instantiating deployment template.

Syntax

```
DBMS_REPCAT_RGT.INSERT_RUNTIME_PARMS (
   runtime_parm_id    IN   NUMBER,
   parameter_name     IN   VARCHAR2,
   parameter_value    IN   CLOB);
```

Parameters

Table 37–60 INSERT_RUNTIME_PARMS Procedure Parameters

Parameter	Description
runtime_parm_id	The ID retrieved by the GET_RUNTIME_PARM_ID function. This ID is also used when instantiating the deployment template (be sure to use the same ID for all parameter values for a deployment template).
parameter_name	Name of the template parameter that you are defining a runtime parameter value for (query the DBA_REPCAT_TEMPLATE_PARMS view for a list of template parameters).
parameter_value	The runtime parameter value that you want to use during the deployment template instantiation process.

Exceptions

Table 37–61 INSERT_RUNTIME_PARMS Procedure Exceptions

Exception	Description
miss_refresh_ template	The deployment template name specified is invalid or does not exist.
miss_user	The user name specified is invalid or does not exist.

Table 37–61 INSERT_RUNTIME_PARMS Procedure Exceptions

Exception	Description
miss_user_parm_values	The deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM view.

Usage Notes

Because the this procedure utilizes a CLOB, you need to utilize the DBMS_LOB package when using the INSERT_RUNTIME_PARMS procedure. The following example illustrates how to use the DBMS_LOB package with the INSERT_RUNTIME_PARMS procedure:

```
DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, DBMS_LOB.SESSION);
   tempstring := 'REGION 20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.INSERT_RUNTIME_PARMS(
      runtime_parm_id => 20,
      parameter_name => 'region',
      parameter_value => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

INSTANTIATE_OFFLINE function

This function generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while offline. This generated script should be used at remote snapshot sites that are able to remain connected to the master site for an extended amount of time, as the instantiation process at the remote snapshot site may be lengthy (depending on the amount of data that is populated to the new snapshots). This procedure needs to be executed separately for each user instantiation.

The script generated by this function is stored in the USER_REPCAT_TEMP_OUTPUT temporary view and is used by several Oracle tools, including Replication Manager, during the distribution of deployment templates. The number returned by this function is used to retrieve the appropriate information from the USER_REPCAT_TEMP_OUTPUT view.

Note: This procedure is used in performing an offline instantiation of a deployment template. Additionally, this procedure is for replication administrators that are instantiating for another user. Users wanting to perform their own instantiation should use the public version of the "INSTANTIATE_OFFLINE function" on page 37-42.

This procedure should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a snapshot). See these respective packages for more information on their usage.

Syntax

```
DBMS_REPCAT_RGT.INSTANTIATE_OFFLINE(
     refresh_template_name   IN   VARCHAR2,
     site_name               IN   VARCHAR2,
     user_name               IN   VARCHAR2 := NULL,
     runtime_parm_id         IN   NUMBER   := -1e-130,
     next_date               IN   DATE     := SYSDATE,
     interval                IN   VARCHAR2 :'SYSDATE + 1')
    RETURN NUMBER;
```

Parameters

Table 37–62 INSTANTIATE_OFFLINE Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be instantiated.
site_name	Name of the remote site that is instantiating the deployment template.
user_name	Name of the authorized user that is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, then specify the ID used when creating the runtime parameters (the ID was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	Specifies the next refresh date value to be used when creating the refresh group.
interval	Specifies the refresh interval to be used when creating the refresh group.

Exceptions

Table 37–63 INSTANTIATE_OFFLINE Function Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_user	Name of the authorized user is invalid or does not exist. Verify that the specified user is listed in the DBA_REPCAT_TEMPLATE_AUTH view; if user is not listed, then the specified user is not authorized to instantiate the target deployment template.
bad_parms	All of the template parameters were not populated by the defined user parameter values and/or template default values. The number of pre-defined values may not have matched the number of template parameters or pre-defined value was invalid for the target parameter (i.e., type mismatch).

Returns

Table 37–64 INSTANTIATE_OFFLINE Function Returns

Return Value	Description
<system generated number>	Specify the generated system number for the `output_id` when you select from the `USER_REPCAT_TEMP_OUTPUT` view to retrieve the generated instantiation script.

INSTANTIATE_ONLINE function

This function generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while online. This generated script should be used at remote snapshot sites that are able to remain connected to the master site for an extended amount of time, as the instantiation process at the remote snapshot site may be lengthy (depending on the amount of data that is populated to the new snapshots). This procedure needs to be executed separately for each user instantiation.

The script generated by this function is stored in the `USER_REPCAT_TEMP_OUTPUT` temporary view and is used by several Oracle tools, including Replication Manager, during the distribution of deployment templates. The number returned by this function is used to retrieve the appropriate information from the `USER_REPCAT_TEMP_OUTPUT` view.

> **Note:** This procedure is for replication administrators that are instantiating for another user. Users wanting to perform their own instantiation should use the public version of the "INSTANTIATE_OFFLINE function" on page 37-42.

Syntax

```
DBMS_REPCAT_RGT.INSTANTIATE_ONLINE(
   refresh_template_name   IN   VARCHAR2,
   site_name               IN   VARCHAR2 := NULL,
   user_name               IN   VARCHAR2 := NULL,
   runtime_parm_id         IN   NUMBER   := -1e-130,
   next_date               IN   DATE     := SYSDATE,
   interval                IN   VARCHAR2 :'SYSDATe + 1')
  RETURN NUMBER;
```

Parameters

Table 37–65 INSTANTIATE_ONLINE Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be instantiated.
site_name	Name of the remote site that is instantiating the deployment template.
user_name	Name of the authorized user that is instantiating the deployment template.
runtime_parm_id	If you have defined runtime parameter values using the INSERT_RUNTIME_PARMS procedure, then specify the ID used when creating the runtime parameters (the ID was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	Specifies the next refresh date value to be used when creating the refresh group.
interval	Specifies the refresh interval to be used when creating the refresh group.

Exceptions

Table 37–66 INSTANTIATE_ONLINE Function Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_user	Name of the authorized user is invalid or does not exist. Verify that the specified user is listed in the DBA_REPCAT_TEMPLATE_AUTH view; if user is not listed, then the specified user is not authorized to instantiate the target deployment template.
bad_parms	All of the template parameters were not populated by the defined user parameter values and/or template default values. The number of pre-defined values may not have matched the number of template parameters or pre-defined value was invalid for the target parameter (i.e., type mismatch).

Returns

Table 37–67 INSTANTIATE_ONLINE Function Returns

Return Value	Description
<system generated number>	Specify the generated system number for the `output_id` when you select from the `USER_REPCAT_TEMP_OUTPUT` view to retrieve the generated instantiation script.

LOCK_TEMPLATE_EXCLUSIVE procedure

When a deployment template is being updated or modified, you should use the `LOCK_TEMPLATE_EXCLUSIVE` procedure to prevent users from reading or instantiating the template.

The lock will be released when a `ROLLBACK` or `COMMIT` is performed.

Note: This procedure should be executed before you make any modifications to your deployment template.

Syntax

```
DBMS_REPCAT_RGT.LOCK_TEMPLATE_EXCLUSIVE;
```

Parameters

None.

LOCK_TEMPLATE_SHARED procedure

This procedure is used to make a specified deployment template "read-only." This procedure should be called before instantiating a template, as this will ensure that nobody can change the deployment template while it is being instantiated.

The lock will be released when a `ROLLBACK` or `COMMIT` is performed.

Syntax

```
DBMS_REPCAT_RGT.LOCK_TEMPLATE_SHARED;
```

Parameters

None.

38

DBMS_REPUTIL

DBMS_REPUTIL contains subprograms to generate shadow tables, triggers, and packages for table replication, as well as subprograms to generate wrappers for replication of standalone procedure invocations and packaged procedure invocations.

This package is referenced only by the generated code.

Summary of Subprograms

Table 38–1 DBMS_REPUTIL Package Subprograms

Subprogram	Description
REPLICATION_OFF procedure on page 38-2	modifies tables without replicating the modifications to any other sites in the replicated environment, or disables row-level replication when using procedural replication.
REPLICATION_ON procedure on page 38-2	Re-enables replication of changes after replication has been temporarily suspended.
REPLICATION_IS_ON function on page 38-3	Determines whether or not replication is running.
FROM_REMOTE function on page 38-3	Returns TRUE at the beginning of procedures in the internal replication packages, and returns FALSE at the end of these procedures.
GLOBAL_NAME function on page 38-3	Determines the global database name of the local database (the global name is the returned value).

REPLICATION_OFF procedure

This procedure modifies tables without replicating the modifications to any other sites in the replicated environment, or disables row-level replication when using procedural replication. In general, you should suspend replication activity for all master groups in your replicated environment before setting this flag.

Syntax

```
DBMS_REPUTIL.REPLICATION_OFF;
```

Parameters

None.

REPLICATION_ON procedure

This procedure re-enables replication of changes after replication has been temporarily suspended.

Syntax

```
DBMS_REPUTIL.REPLICATION_ON;
```

Parameters

None.

REPLICATION_IS_ON function

This function determines whether or not replication is running. A returned value of TRUE indicates that the generated replication triggers are enabled. FALSE indicates that replication is disabled at the current site for the replicated object group.

The returning value of this function is set by calling the REPLICATION_ON or REPLICATION_OFF procedures in the DBMS_REPUTIL package.

Syntax

```
DBMS_REPUTIL.REPLICATION_IS_ON
  RETURN BOOLEAN;
```

Parameters

None.

FROM_REMOTE function

This function returns TRUE at the beginning of procedures in the internal replication packages, and returns FALSE at the end of these procedures. You may need to check this function if you have any triggers that could be fired as the result of an update by an internal package.

Syntax

```
DBMS_REPUTIL.FROM_REMOTE
   RETURN BOOLEAN;
```

Parameters

None.

GLOBAL_NAME function

This function determines the global database name of the local database (the global name is the returned value).

Syntax

```
DBMS_REPUTIL.GLOBAL_NAME
   RETURN VARCHAR2;
```

Parameters

None.

39

DBMS_RESOURCE_MANAGER

The DBMS_RESOURCE_MANAGER package maintains plans, consumer groups, and plan directives. It also provides semantics so that you may group together changes to the plan schema.

> **See Also:** For more information on using the Database Resource Manager, see *Oracle8i Administrator's Guide.*

Requirements

The invoker must have the ADMINISTER_RESOURCE_MANAGER system privilege to execute these procedures. The procedures to grant and revoke this privilege are in the package DBMS_RESOURCE_MANAGER_PRIVS.

Summary of Subprograms

Table 39–1 DBMS_RESOURCE_MANAGER Package Subprograms

Subprogram	Description
CREATE_PLAN procedure on page 39-3	Creates entries which define resource plans.
UPDATE_PLAN procedure on page 39-4	Updates entries which define resource plans.
DELETE_PLAN procedure on page 39-4	Deletes the specified plan as well as all the plan directives it refers to.
DELETE_PLAN_CASCADE procedure on page 39-5	Deletes the specified plan as well as all its descendants (plan directives, subplans, consumer groups).
CREATE_CONSUMER_GROUP procedure on page 39-6	Creates entries which define resource consumer groups.
UPDATE_CONSUMER_GROUP procedure on page 39-6	Updates entries which define resource consumer groups.
DELETE_CONSUMER_GROUP procedure on page 39-7	Deletes entries which define resource consumer groups.
CREATE_PLAN_DIRECTIVE procedure on page 39-7	Creates resource plan directives.
UPDATE_PLAN_DIRECTIVE procedure on page 39-9	Updates resource plan directives.
DELETE_PLAN_DIRECTIVE procedure on page 39-10	Deletes resource plan directives.
CREATE_PENDING_AREA procedure on page 39-10	Creates a work area for changes to resource manager objects.
VALIDATE_PENDING_AREA procedure on page 39-12	Validates pending changes for the resource manager.
CLEAR_PENDING_AREA procedure on page 39-12	Clears the work area for the resource manager.

Table 39–1 DBMS_RESOURCE_MANAGER Package Subprograms

Subprogram	Description
SUBMIT_PENDING_AREA procedure on page 39-12	Submits pending changes for the resource manager.
SET_INITIAL_CONSUMER_GROUP procedure on page 39-16	Assigns the initial resource consumer group for a user.
SWITCH_CONSUMER_GROUP_FOR_SESS procedure on page 39-17	Changes the resource consumer group of a specific session.
SWITCH_CONSUMER_GROUP_FOR_USER procedure on page 39-17	Changes the resource consumer group for all sessions with a given user name.

CREATE_PLAN procedure

This procedure creates entries which define resource plans.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN (
   plan                      IN VARCHAR2,
   comment                   IN VARCHAR2,
   cpu_mth                   IN VARCHAR2 DEFAULT 'EMPHASIS',
   max_active_sess_target_mth IN VARCHAR2 DEFAULT 'MAX_ACTIVE_SESS_ABSOLUTE',
   parallel_degree_limit_mth  IN VARCHAR2 DEFAULT 'PARALLEL_DEGREE_LIMIT_
ABSOLUTE');
```

Parameters

Table 39–2 CREATE_PLAN Procedure Parameters

Parameter	Description
plan	Name of resource plan.
cpu_mth	Allocation method for CPU resources.
max_active_sess_target_mth	Allocation method for maximum active sessions.
parallel_degree_limit_mth	Allocation method for degree of parallelism.
comment	User's comment.

UPDATE_PLAN procedure

This procedure updates entries which define resource plans.

Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_PLAN (
   plan                           IN VARCHAR2,
   new_comment                    IN VARCHAR2 DEFAULT NULL,
   new_cpu_mth                    IN VARCHAR2 DEFAULT NULL,
   new_max_active_sess_target_mth IN VARCHAR2 DEFAULT NULL,
   new_parallel_degree_limit_mth  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 39–3 UPDATE_PLAN Procedure Parameters

Parameter	Description
plan	Name of resource plan.
new_comment	New user's comment.
new_cpu_mth	Name of new allocation method for CPU resources.
new_max_active_sess_target_mth	Name of new method for maximum active sessions.
new_parallel_degree_limit_mth	Name of new method for degree of parallelism.

Usage Notes

If the parameters to UPDATE_PLAN are not specified, then they remain unchanged in the data dictionary.

DELETE_PLAN procedure

This procedure deletes the specified plan as well as all the plan directives to which it refers.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN (
   plan IN VARCHAR2);
```

Parameters

Table 39–4 DELETE_PLAN Procedure Parameters

Parameter	Description
plan	Name of resource plan to delete.

DELETE_PLAN_CASCADE procedure

This procedure deletes the specified plan and all of its descendants (plan directives, subplans, consumer groups). Mandatory objects and directives are not deleted.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN_CASCADE (
   plan IN VARCHAR2);
```

Parameters

Table 39–5 DELETE_PLAN_CASCADE Procedure Parameters

Parameters	Description
plan	Name of plan.

Errors

If DELETE_PLAN_CASCADE encounters any error, then it rolls back, and nothing is deleted.

> **Note:** If you want to use any default resource allocation method, then you do not need not specify it when creating or updating a plan.

Usage Notes

Defaults are:

- cpu_method = EMPHASIS
- parallel_degree_limit_mth = PARALLEL_DEGREE_LIMIT_ABSOLUTE
- max_active_sess_target_mth = MAX_ACTIVE_SESS_ABSOLUTE

Note: The parameter `max_active_sess_target_mth` is undocumented in this release: It is reserved for future use.

CREATE_CONSUMER_GROUP procedure

This procedure lets you create entries which define resource consumer groups.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
   consumer_group IN VARCHAR2,
   comment        IN VARCHAR2,
   cpu_mth        IN VARCHAR2 DEFAULT 'ROUND-ROBIN');
```

Parameters

Table 39–6 CREATE_CONSUMER_GROUP Procedure Parameters

Parameter	Description
consumer_group	Name of consumer group.
comment	User's comment.
cpu_mth	Name of CPU resource allocation method.

UPDATE_CONSUMER_GROUP procedure

This procedure lets you update entries which define resource consumer groups.

Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_CONSUMER_GROUP (
   consumer_group IN VARCHAR2,
   new_comment    IN VARCHAR2 DEFAULT NULL,
   new_cpu_mth    IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 39–7 UPDATE_CONSUMER_GROUP Procedure Parameter

Parameter	Description
consumer_group	Name of consumer group.

Table 39–7 UPDATE_CONSUMER_GROUP Procedure Parameter

Parameter	Description
new_comment	New user's comment.
new_cpu_mth	Name of new method for CPU resource allocation.

If the parameters to the UPDATE_CONSUMER_GROUP procedure are not specified, then they remain unchanged in the data dictionary.

DELETE_CONSUMER_GROUP procedure

This procedure lets you delete entries which define resource consumer groups.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_CONSUMER_GROUP (
   consumer_group IN VARCHAR2);
```

Parameters

Table 39–8 DELETE_CONSUMER_GROUP Procedure Parameters

Parameters	Description
consumer_group	Name of consumer group to be deleted.

CREATE_PLAN_DIRECTIVE procedure

This procedure lets you create resource plan directives.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   plan                       IN VARCHAR2,
   group_or_subplan           IN VARCHAR2,
   comment                    IN VARCHAR2,
   cpu_p1                     IN NUMBER   DEFAULT NULL,
   cpu_p2                     IN NUMBER   DEFAULT NULL,
   cpu_p3                     IN NUMBER   DEFAULT NULL,
   cpu_p4                     IN NUMBER   DEFAULT NULL,
   cpu_p5                     IN NUMBER   DEFAULT NULL,
   cpu_p6                     IN NUMBER   DEFAULT NULL,
   cpu_p7                     IN NUMBER   DEFAULT NULL,
   cpu_p8                     IN NUMBER   DEFAULT NULL,
   max_active_sess_target_p1 IN NUMBER   DEFAULT NULL,
   parallel_degree_limit_p1  IN NUMBER   DEFAULT NULL);
```

Parameters

Table 39–9 CREATE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
plan	Name of resource plan.
group_or_subplan	Name of consumer group or subplan.
comment	Comment for the plan directive.
cpu_p1	First parameter for the CPU resource allocation method.
cpu_p2	Second parameter for the CPU resource allocation method.
cpu_p3	Third parameter for the CPU resource allocation method.
cpu_p4	Fourth parameter for the CPU resource allocation method.
cpu_p5	Fifth parameter for the CPU resource allocation method.
cpu_p6	Sixth parameter for the CPU resource allocation method.
cpu_p7	Seventh parameter for the CPU resource allocation method.
cpu_p8	Eighth parameter for the CPU resource allocation method.
max_active_sess_target_p1	First parameter for the maximum active sessions allocation method (Reserved for future use).
parallel_degree_limit_p1	First parameter for the degree of parallelism allocation method.

All parameters default to NULL. However, for the EMPHASIS CPU resource allocation method, this case would starve all the users.

UPDATE_PLAN_DIRECTIVE procedure

This procedure lets you update resource plan directives.

Syntax

```
DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE (
   plan                          IN VARCHAR2,
   group_or_subplan              IN VARCHAR2,
   new_comment                   IN VARCHAR2 DEFAULT NULL,
   new_cpu_p1                    IN NUMBER   DEFAULT NULL,
   new_cpu_p2                    IN NUMBER   DEFAULT NULL,
   new_cpu_p3                    IN NUMBER   DEFAULT NULL,
   new_cpu_p4                    IN NUMBER   DEFAULT NULL,
   new_cpu_p5                    IN NUMBER   DEFAULT NULL,
   new_cpu_p6                    IN NUMBER   DEFAULT NULL,
   new_cpu_p7                    IN NUMBER   DEFAULT NULL,
   new_cpu_p8                    IN NUMBER   DEFAULT NULL,
   new_max_active_sess_target_p1 IN NUMBER   DEFAULT NULL,
   new_parallel_degree_limit_p1  IN NUMBER   DEFAULT NULL);
```

Parameters

Table 39–10 UPDATE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
plan	Name of resource plan.
group_or_subplan	Name of consumer group or subplan.
comment	Comment for the plan directive.
cpu_p1	First parameter for the CPU resource allocation method.
cpu_p2	Second parameter for the CPU resource allocation method.
cpu_p3	Third parameter for the CPU resource allocation method.
cpu_p4	Fourth parameter for the CPU resource allocation method.
cpu_p5	Fifth parameter for the CPU resource allocation method.
cpu_p6	Sixth parameter for the CPU resource allocation method.
cpu_p7	Seventh parameter for the CPU resource allocation method.

Table 39–10 UPDATE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
cpu_p8	Eighth parameter for the CPU resource allocation method.
max_active_sess_target_p1	First parameter for the maximum active sessions allocation method (Reserved for future use).
parallel_degree_limit_p1	First parameter for the degree of parallelism allocation method.

If the parameters for UPDATE_PLAN_DIRECTIVE are left unspecified, then they remain unchanged in the data dictionary.

DELETE_PLAN_DIRECTIVE procedure

This procedure lets you delete resource plan directives.

Syntax

```
DBMS_RESOURCE_MANAGER.DELETE_PLAN_DIRECTIVE (
   plan            IN VARCHAR2,
   group_or_subplan IN VARCHAR2);
```

Parameters

Table 39–11 DELETE_PLAN_DIRECTIVE Procedure Parameters

Parameter	Description
plan	Name of resource plan.
group_or_subplan	Name of group or subplan.

CREATE_PENDING_AREA procedure

This procedure lets you make changes to resource manager objects.

All changes to the plan schema must be done within a pending area. The pending area can be thought of as a "scratch" area for plan schema changes. The administrator creates this pending area, makes changes as necessary, possibly validates these changes, and only when the submit is completed do these changes become active.

You may, at any time while the pending area is active, view the current plan schema with your changes by selecting from the appropriate user views.

At any time, you may clear the pending area if you want to stop the current changes. You may also call the VALIDATE procedure to confirm whether the changes you has made are valid. You do not have to do your changes in a given order to maintain a consistent group of entries. These checks are also implicitly done when the pending area is submitted.

Note: Oracle allows "orphan" consumer groups (i.e., consumer groups that have no plan directives that refer to them). This is in anticipation that an administrator may want to create a consumer group that is not currently being used, but will be used in the future.

Syntax

```
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA;
```

Parameters

None.

Usage Notes

The following rules must be adhered to, and they are checked whenever the validate or submit procedures are executed:

1. No plan schema may contain any loops.
2. All plans and consumer groups referred to by plan directives must exist.
3. All plans must have plan directives that refer to either plans or consumer groups.
4. All percentages in any given level must not add up to greater than 100 for the emphasis resource allocation method.
5. No plan may be deleted that is currently being used as a top plan by an active instance.
6. For Oracle8*i*, the plan directive parameter, `parallel_degree_limit_p1`, may only appear in plan directives that refer to consumer groups (i.e., not at subplans).

7. There cannot be more than 32 plan directives coming from any given plan (i.e., no plan can have more than 32 children).
8. There cannot be more than 32 consumer groups in any active plan schema.
9. Plans and consumer groups use the same namespace; therefore, no plan can have the same name as any consumer group.
10. There must be a plan directive for OTHER_GROUPS somewhere in any active plan schema.This ensures that a session not covered by the currently active plan is allocated resources as specified by the OTHER_GROUPS directive.

If any of the above rules are broken when checked by the VALIDATE or SUBMIT procedures, then an informative error message is returned. You may then make changes to fix the problem(s) and reissue the validate or submit procedures.

VALIDATE_PENDING_AREA procedure

This procedure lets you validate pending changes for the resource manager.

Syntax

```
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA;
```

Parameters

None.

CLEAR_PENDING_AREA procedure

This procedure lets you clear pending changes for the resource manager.

Syntax

```
DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA;
```

Parameters

None.

SUBMIT_PENDING_AREA procedure

This procedure lets you submit pending changes for the resource manager: It clears the pending area after validating and committing the changes (if valid).

Note: A call to SUBMIT_PENDING_AREA may fail even if VALIDATE_PENDING_AREA succeeds. This may happen if a plan being deleted is loaded by an instance after a call to VALIDATE_PENDING_AREA, but before a call to SUBMIT_PENDING_AREA.

Syntax

```
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA;
```

Parameters

None.

Example

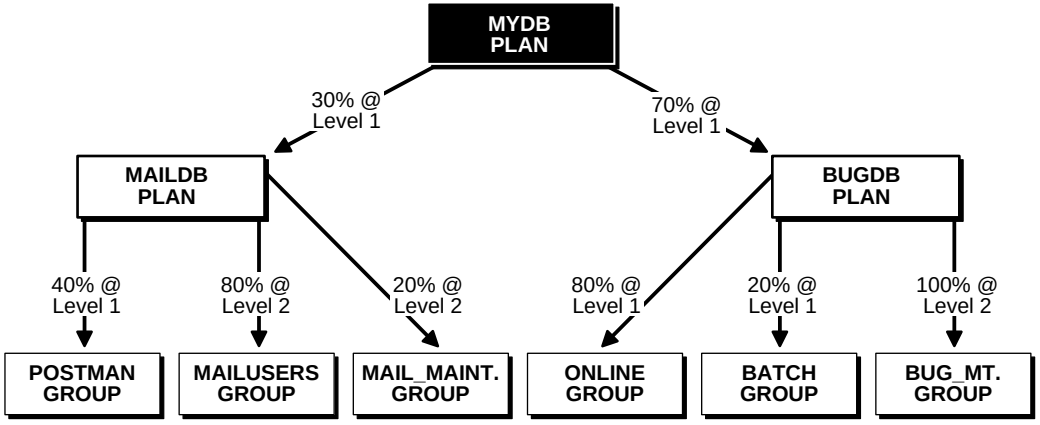
One of the advantages of plans is that they can refer to each other. The entries in a plan can either be consumer groups or subplans. For example, the following is also a set of valid CPU plan directives:

Table 39–12 MYDB PLAN CPU Plan Directives

Subplan/Group	CPU_Level 1
MAILDB Plan	30%
BUGDB Plan	70%

If these plan directives were in effect and there were an infinite number of runnable sessions in all consumer groups, then the MAILDB plan would be assigned 30% of the available CPU resources, while the BUGDB plan would be assigned 70% of the available CPU resources. Breaking this further down, sessions in the "Postman" consumer group would be run 12% (40% of 30%) of the time, while sessions in the

"Online" consumer group would be run 56% (80% of 70%) of the time. The following diagram depicts this scenario:



Conceptually below the consumer groups are the active sessions. In other words, a session belongs to a resource consumer group, and this consumer group is used by a plan to determine allocation of processing resources.

A multi-plan (plan with one or more subplans) definition of CPU plan directives cannot be collapsed into a single plan with one set of plan directives, because each plan is its own entity. The CPU quanta that is allotted to a plan or subplan gets used only within that plan, unless that plan contains no consumer groups with active sessions. Therefore, in this example, if the Bug Maintenance Group did not use any of its quanta, then it would get recycled within that plan, thus going back to level 1 within the BUGDB PLAN. If the multi-plan definition in the above example got collapsed into a single plan with multiple consumer groups, then there would be no way to explicitly recycle the Bug Maintenance Group's unused quanta. It would have to be recycled globally, thus giving the mail sessions an opportunity to use it.

The resources for a database can be partitioned at a high level among multiple applications and then repartitioned within an application. If a given group within an application does not need all the resources it is assigned, then the resource is only repartitioned within the same application.

This example uses the default plan and consumer group allocation methods:

```
create_pending_area();
create_plan(plan => 'BUGDB_PLAN', comment => 'Resource
   plan/method for bug users' sessions');
create_plan(plan => 'MAILDB_PLAN', comment => 'Resource
```

```
   plan/method for mail users' sessions');
create_plan(plan => 'MYDB_PLAN', comment => 'Resource
   plan/method for bug and mail users' sessions');
create_consumer_group(consumer_group => 'Bug_Online_group',
   comment => 'Resource consumer group/method for online bug users'
   sessions');
create_consumer_group(consumer_group => 'Bug_Batch_group',
   comment => 'Resource consumer group/method for bug users' sessions
   who run batch jobs');
create_consumer_group(consumer_group =>
   'Bug_Maintenance_group', comment => 'Resource consumer
   group/method for users' sessions who maintain the bug db');
create_consumer_group(consumer_group => 'Mail_users_group',
   comment => 'Resource consumer group/method for mail users'
   sessions');
create_consumer_group(consumer_group => 'Mail_Postman_group',
   comment => 'Resource consumer group/method for mail postman');
create_consumer_group(consumer_group =>
   'Mail_Maintenance_group', comment => 'Resource consumer
   group/method for users' sessions who maintain the mail db');
create_plan_directive(plan => 'BUGDB_PLAN', group_or_subplan
   => 'Bug_Online_group', comment => 'online bug users'
   sessions at level 0', cpu_p1 => 80, cpu_p2=> 0,
   parallel_degree_limit_p1 => 8);
create_plan_directive(plan => 'BUGDB_PLAN', group_or_subplan
   => 'Bug_Batch_group', comment => 'batch bug users'
   sessions at level 0', cpu_p1 => 20, cpu_p2 => 0,
   parallel_degree_limit_p1 => 2);
create_plan_directive(plan => 'BUGDB_PLAN', group_or_subplan
   => 'Bug_Maintenance_group', comment => 'bug maintenance users'
   sessions at level 1', cpu_p1 => 0, cpu_p2 => 100,
   parallel_degree_limit_p1 => 3);
create_plan_directive(plan => 'MAILDB_PLAN', group_or_subplan
   => 'Mail_Postman_group', comment => 'mail postman at
   level 0', cpu_p1 => 40, cpu_p2 => 0,
   parallel_degree_limit_p1 => 4);
create_plan_directive(plan => 'MAILDB_PLAN', group_or_subplan
   => 'Mail_users_group', comment => 'mail users' sessions
   at level 1', cpu_p1 => 0, cpu_p2 => 80,
   parallel_degree_limit_p1 => 4);
create_plan_directive(plan => 'MAILDB_PLAN', group_or_subplan =>
   'Mail_Maintenance_group', comment => 'mail
   maintenance users' sessions at level 1', cpu_p1 => 0,
   cpu_p2 => 20, parallel_degree_limit_p1 => 2);
create_plan_directive(plan => 'MYDB_PLAN', group_or_subplan =>
```

```
   'MAILDB_PLAN', comment=> 'all mail users' sessions at
   level 0', cpu_p1 => 30);
create_plan_directive(plan => 'MYDB_PLAN', group_or_subplan =>
   'BUGDB_PLAN', comment => 'all bug users' sessions at
   level 0', cpu_p1 = 70);
validate_pending_area();
submit_pending_area();
```

The above call to VALIDATE_PENDING_AREA is optional, because the validation is implicitly done in SUBMIT_PENDING_AREA.

SET_INITIAL_CONSUMER_GROUP procedure

The initial consumer group of a user is the consumer group to which any session created by that user initially belongs. This procedure sets the initial resource consumer group for a user.

Syntax

```
DBMS_RESOURCE_MANAGER.SET_INITIAL_CONSUMER_GROUP (
   user           IN VARCHAR2,
   consumer_group IN VARCHAR2);
```

Parameters

Table 39–13 SET_INITIAL_CONSUMER_GROUP Procedure Parameters

Parameters	Description
user	Name of the user.
consumer_group	The user's initial consumer group.

Usage Notes

The ADMINISTER_RESOURCE_MANAGER or the ALTER USER system privilege are required to be able to execute this procedure. The user, or PUBLIC, must be directly granted switch privilege to a consumer group before it can be set to be the user's initial consumer group. Switch privilege for the initial consumer group cannot come from a role granted to that user.

Note: These semantics are similar to those for ALTER USER DEFAULT ROLE.

If the initial consumer group for a user has never been set, then the user's initial consumer group is automatically the consumer group: DEFAULT_CONSUMER_GROUP.

DEFAULT_CONSUMER_GROUP has switch privileges granted to PUBLIC; therefore, all users are automatically granted switch privilege for this consumer group. Upon deletion of a consumer group, all users having the deleted group as their initial consumer group now have DEFAULT_CONSUMER_GROUP as their initial consumer group. All currently active sessions belonging to a deleted consumer group are switched to DEFAULT_CONSUMER_GROUP.

SWITCH_CONSUMER_GROUP_FOR_SESS procedure

This procedure lets you change the resource consumer group of a specific session. It also changes the consumer group of any (PQ) slave sessions that are related to the top user session.

Syntax

```
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS (
   session_id     IN NUMBER,
   session_serial IN NUMBER,
   consumer_group IN VARCHAR2);
```

Parameters

Table 39–14 SWITCH_CONSUMER_GROUP_FOR_SESS Procedure Parameters

Parameter	Description
session_id	SID column from the view V$SESSION.
session_serial	SERIAL# column from view V$SESSION.
consumer_group	Name of the consumer group to switch to.

SWITCH_CONSUMER_GROUP_FOR_USER procedure

This procedure lets you change the resource consumer group for all sessions with a given user ID. It also change the consumer group of any (PQ) slave sessions that are related to the top user session.

Syntax

```
DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER (
   user           IN VARCHAR2,
   consumer_group IN VARCHAR2);
```

Parameters

Table 39–15 SWITCH_CONSUMER_GROUP_FOR_USER Procedure Parameters

Parameter	Description
user	Name of the user.
consumer_group	Name of the consumer group to switch to.

Usage Notes

The SWITCH_CONSUMER_GROUP_FOR_SESS and SWITCH_CONSUMER_GROUP_FOR_USER procedures let you to raise or lower the allocation of CPU resources of certain sessions or users. This provides a functionality similar to the nice command on UNIX.

These procedures cause the session to be moved into the newly specified consumer group immediately.

40

DBMS_RESOURCE_MANAGER_PRIVS

The DBMS_RESOURCE_MANAGER_PRIVS package maintains privileges associated with the Resource Manager.

> **See Also:** For more information on using the Database Resource Manager, see *Oracle8i Administrator's Guide.*

Summary of Subprograms

Table 40–1 DBMS_RESOURCE_MANAGER_PRIVS Package Subprograms

Subprogram	Description
GRANT_SYSTEM_PRIVILEGE procedure on page 40-2	Performs a grant of a system privilege.
REVOKE_SYSTEM_PRIVILEGE procedure on page 40-3	Performs a revoke of a system privilege.
GRANT_SWITCH_CONSUMER_GROUP procedure on page 40-3	Grants the privilege to switch to resource consumer groups.
REVOKE_SWITCH_CONSUMER_GROUP procedure on page 40-5	Revokes the privilege to switch to resource consumer groups.

GRANT_SYSTEM_PRIVILEGE procedure

This procedure performs a grant of a system privilege to a user or role.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE (
   grantee_name   IN VARCHAR2,
   privilege_name IN VARCHAR2 DEFAULT 'ADMINISTER_RESOURCE_MANAGER',
   admin_option   IN BOOLEAN);
```

Parameters

Table 40–2 GRANT_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
grantee_name	Name of the user or role to whom privilege is to be granted.
privilege_name	Name of the privilege to be granted.
admin_option	TRUE if the grant is with admin_option, FALSE otherwise.

Currently, Oracle provides only one system privilege for the Resource Manager: ADMINISTER_RESOURCE_MANAGER. Database administrators have this system privilege with the admin option. The grantee and the revokee can either be a user or a role. Users that have been granted the system privilege with the admin option can also grant this privilege to others.

Example

The following call grants this privilege to a user called scott without the admin option:

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE (
   grantee_name => 'scott',
   admin_option => FALSE);
```

REVOKE_SYSTEM_PRIVILEGE procedure

This procedure performs a revoke of a system privilege from a user or role.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SYSTEM_PRIVILEGE (
   revokee_name   IN VARCHAR2,
   privilege_name IN VARCHAR2 DEFAULT 'ADMINISTER_RESOURCE_MANAGER');
```

Parameters

Table 40–3 REVOKE_SYSTEM_PRIVILEGE Procedure Parameters

Parameter	Description
revokee_name	Name of the user or role from whom privilege is to be revoked.
privilege_name	Name of the privilege to be revoked.

Example

The following call revokes the ADMINISTER_RESOURCE_MANAGER from user scott:

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SYSTEM_PRIVILEGE ('scott');
```

GRANT_SWITCH_CONSUMER_GROUP procedure

This procedure grants the privilege to switch to a resource consumer group.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (
   grantee_name   IN VARCHAR2,
   consumer_group IN VARCHAR2,
   grant_option   IN BOOLEAN);
```

Parameters

Table 40–4 GRANT_SWITCH_CONSUMER_GROUP Procedure Parameters

Parameter	Description
grantee_name	Name of the user or role to whom privilege is to be granted.
consumer_group	Name of consumer group.
grant_option	TRUE if grantee should be allowed to grant access, FALSE otherwise.

Usage Notes

If you grant permission to switch to a particular consumer group to a user, then that user can immediately switch their current consumer group to the new consumer group.

If you grant permission to switch to a particular consumer group to a role, then any users who have been granted that role and have enabled that role can immediately switch their current consumer group to the new consumer group.

If you grant permission to switch to a particular consumer group to PUBLIC, then any user can switch to that consumer group.

If the grant_option parameter is TRUE, then users granted switch privilege for the consumer group may also grant switch privileges for that consumer group to others.

In order to set the initial consumer group of a user, you must grant the switch privilege for that group to the user.

See Also: Chapter 39, "DBMS_RESOURCE_MANAGER"

Example

```
DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_COMSUMER_GROUP (
   'scott', 'mail_maintenance_group', true);

DBMS_RESOURCE_MANAGER.SET_INITIAL_CONSUMER_GROUP (
   'scott', 'mail_maintenance_group');
```

REVOKE_SWITCH_CONSUMER_GROUP procedure

This procedure revokes the privilege to switch to a resource consumer group.

Syntax

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
   revokee_name   IN VARCHAR2,
   consumer_group IN VARCHAR2);
```

Parameters

Table 40–5 REVOKE_SWITCH_CONSUMER_GROUP Procedure Parameter

Parameter	Description
revokee_name	Name of user / role from which to revoke access.
consumer_group	Name of consumer group.

Usage Notes

If you revoke a user's switch privilege for a particular consumer group, then any subsequent attempts by that user to switch to that consumer group will fail.

If you revoke the initial consumer group from a user, then that user will automatically be part of the DEFAULT_CONSUMER_GROUP consumer group when logging in.

If you revoke the switch privilege for a consumer group from a role, then any users who only had switch privilege for the consumer group via that role will not be subsequently able to switch to that consumer group.

If you revoke the switch privilege for a consumer group from PUBLIC, then any users who could previously only use the consumer group via PUBLIC will not be subsequently able to switch to that consumer group.

Example

The following example revokes the privileges to switch to mail_maintenance_group from Scott:

```
DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
   'scott', 'mail_maintenance_group');
```

41

DBMS_RLS

The DBMS_RLS package contains the fine-grained access control administrative interface.

Note: DBMS_RLS is only available with the Enterprise Edition.

Dynamic Predicates

The functionality to support fine-grained access control is based on dynamic predicates, where security rules are not embedded in views, but are acquired at the statement parse time, when the base table or view is referenced in a DML statement.

A dynamic predicate for a table or view is generated by a PL/SQL function, which is associated with a security policy through a PL/SQL interface. For example:

```
DBMS_RLS.ADD_POLICY (
   'scott', 'emp', 'emp_policy', 'secusr', 'emp_sec', 'select');
```

Whenever EMP table, under SCOTT schema, is referenced in a query or subquery (SELECT), the server calls the EMP_SEC function (under SECUSR schema). This returns a predicate specific to the current user for the EMP_POLICY policy. The policy function may generate the predicates based on whatever session environment variables are available during the function call. These variables usually appear in the form of application contexts.

The server then produces a transient view with the text:

```
SELECT * FROM scott.emp WHERE P1
```

Here, P1 (e.g., SAL > 10000, or even a subquery) is the predicate returned from the EMP_SEC function. The server treats the EMP table as a view and does the view expansion just like the ordinary view, except that the view text is taken from the transient view instead of the data dictionary.

If the predicate contains subqueries, then the owner (definer) of the policy function is used to resolve objects within the subqueries and checks security for those objects. In other words, users who have access privilege to the policy protected objects do not need to know anything about the policy. They do not need to be granted object privileges for any underlying security policy. Furthermore, the users also do not require EXECUTE privilege on the policy function, because the server makes the call with the function definer's right.

Note: The transient view can preserve the updatability of the parent object because it is derived from a single table or view with predicate only; i.e., no JOIN, ORDER BY, GROUP BY, etc.

The DBMS_RLS package also provides the interface to drop and enable/disable security policies. For example, you can drop or disable the EMP_POLICY with the following PL/SQL statements:

```
DBMS_RLS.DROP_POLICY('scott', 'emp', 'emp_policy');
DBMS_RLS.ENABLE_POLICY('scott', 'emp', 'emp_policy', FALSE)
```

Security

A security check is performed when the transient view is created with subquery. The schema owning the policy function, which generates the dynamic predicate, is the transient view's definer for the purpose of security check and object look-up.

Usage Notes

The DBMS_RLS procedures cause current DML transactions, if any, to commit before the operation. However, the procedures do not cause a commit first if they are inside a DDL event trigger. With DDL transactions, the DBMS_RLS procedures are part of the DDL transaction.

For example, you may create a trigger for CREATE TABLE. Inside the trigger, you may add a column through ALTER TABLE, and you can add a policy through DBMS_RLS. All these operations are in the same transaction as CREATE TABLE, even though each one is a DDL statement. The CREATE TABLE succeeds only if the trigger is completed successfully.

Summary of Subprograms

Table 41–1 DBMS_RLS Subprograms

Subprogram	Description
ADD_POLICY procedure on page 41-4	Creates a fine-grained access control policy to a table or view.
DROP_POLICY procedure on page 41-6	Drops a fine-grained access control policy from a table or view.
REFRESH_POLICY procedure on page 41-6	Causes all the cached statements associated with the policy to be re-parsed.
ENABLE_POLICY procedure on page 41-7	Enables or disables a fine-grained access control policy.

ADD_POLICY procedure

This procedure creates a fine-grained access control policy to a table or view.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: Usage Notes on page 41-3

A commit is also performed at the end of the operation.

Syntax

```
DBMS_RLS.ADD_POLICY (
   object_schema   IN VARCHAR2 := NULL,
   object_name     IN VARCHAR2,
   policy_name     IN VARCHAR2,
   function_schema IN VARCHAR2 := NULL,
   policy_function IN VARCHAR2,
   statement_types IN VARCHAR2 := NULL,
   update_check    IN BOOLEAN  := FALSE,
   enable          IN BOOLEAN  := TRUE);
```

Parameters

Table 41–2 ADD_POLICY Procedure Parameters

Parameter	Description
object_schema	Schema containing the table or view (logon user, if NULL).
object_name	Name of table or view to which the policy is added.
policy_name	Name of policy to be added. It must be unique for the same table or view.
function_schema	Schema of the policy function (logon user, if NULL).
policy_function	Name of a function which generates a predicate for the policy. If the function is defined within a package, then the name of the package must be present.
statement_types	Statement types that the policy will apply. It can be any combination of SELECT, INSERT, UPDATE, and DELETE. The default is to apply to all of these types.

Table 41–2 ADD_POLICY Procedure Parameters

Parameter	Description
update_check	Optional argument for INSERT or UPDATE statement types. The default is FALSE. Setting update_check to TRUE causes the server to also check the policy against the value after insert or update.
enable	Indicates if the policy is enabled when it is added. The default is TRUE

Usage Notes

- SYS is free of any security policy.
- The policy functions which generate dynamic predicates are called by the server. Following is the interface for the function:

```
FUNCTION policy_function (object_schema IN VARCHAR2, object_name VARCHAR2)
    RETURN VARCHAR2
--- object_schema is the schema owning the table of view.
--- object_name is the name of table of view that the policy will apply.
```

  The maximum length of the predicate that the policy function can return is 2,000 bytes.

- The policy functions must have the purity level of WNDS (write no database state).

  See Also: The *PL/SQL User's Guide and Reference* has more details about the RESTRICT_REFERENCES pragma.

- Dynamic predicates generated out of different policies for the same object have the combined effect of a conjunction (ANDed) of all the predicates.
- The security check and object lookup are performed against the owner of the policy function for objects in the subqueries of the dynamic predicates.
- If the function returns a zero length predicate, then it is interpreted as no restriction being applied to the current user for the policy.
- When table alias is required (e.g., parent object is a type table) in the predicate, the name of the table or view itself must be used as the name of the alias. The server constructs the transient view as something like "select c1, c2, ... from tab where <predicate>".

- The checking of the validity of the function is done at runtime for ease of installation and other dependency issues during import/export.

DROP_POLICY procedure

This procedure drops a fine-grained access control policy from a table or view.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: Usage Notes on page 41-3

A commit is also performed at the end of the operation.

Syntax

```
DBMS_RLS.DROP_POLICY (
   object_schema IN VARCHAR2 := NULL,
   object_name   IN VARCHAR2,
   policy_name   IN VARCHAR2);
```

Parameters

Table 41–3 DROP_POLICY Procedure Parameters

Parameter	Description
object_schema	Schema containing the table or view (logon user if NULL).
object_name	Name of table or view.
policy_name	Name of policy to be dropped from the table or view.

REFRESH_POLICY procedure

This procedure causes all the cached statements associated with the policy to be re-parsed. This guarantees that the latest change to this policy will have immediate effect after the procedure is executed.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: Usage Notes on page 41-3

A commit is also performed at the end of the operation.

Syntax

```
DBMS_RLS.REFRESH_POLICY (
   object_schema IN VARCHAR2 := NULL,
   object_name   IN VARCHAR2 := NULL,
   policy_name   IN VARCHAR2 := NULL);
```

Parameters

Table 41–4 REFRESH_POLICY Procedure Parameters

Parameter	Description
object_schema	Schema containing the table or view.
object_name	Name of table or view that the policy is associated with.
policy_name	Name of policy to be refreshed.

Errors

The procedure returns an error if it tries to refresh a disabled policy.

ENABLE_POLICY procedure

This procedure enables or disables a fine-grained access control policy. A policy is enabled when it is created.

The procedure causes the current transaction, if any, to commit before the operation is carried out. However, this does not cause a commit first if it is inside a DDL event trigger.

See Also: Usage Notes on page 41-3

A commit is also performed at the end of the operation.

Syntax

```
DBMS_RLS.ENABLE_POLICY (
   object_schema IN VARCHAR2 := NULL,
   object_name   IN VARCHAR2,
   policy_name   IN VARCHAR2,
   enable        IN BOOLEAN);
```

Parameters

Table 41–5 ENABLE_POLICY Procedure Parameters

Parameter	Description
object_schema	Schema containing the table or view (logon user if NULL).
object_name	Name of table or view that the policy is associated with.
policy_name	Name of policy to be enabled or disabled.
enable	TRUE to enable the policy, FALSE to disable the policy.

Example

This example illustrates the necessary steps to enforce a fine-grained access control policy.

In an Oracle HR application, PER_PEOPLE is a view for the PER_ALL_PEOPLE table, and both objects are under APPS schema.

```
CREATE TABLE per_all_people
            (person_id NUMBER(15),
             last_name VARCHAR2(30),
             emp_no VARCHAR2(15), ...);
CREATE VIEW per_people AS
       SELECT * FROM per_all_people;
```

There should be a security policy that limits access to the PER_PEOPLE view based on the user's role in the company. The predicates for the policy can be generated by the SECURE_PERSON function in the HR_SECURITY package. The package is under schema APPS and contains functions to support all security policies related to the HR application. Also, all the application contexts are under the APPS_SEC namespace.

```
CREATE PACKAGE BODY hr_security IS
  FUNCTION secure_person(obj_schema VARCHAR2, obj_name VARCHAR2)
                    RETURN VARCHAR2 IS
```

```
      d_predicate VARCHAR2(2000);
  BEGIN
      -- for users with HR_ROLE set to EMP, map logon user name
      -- to employee id. FND_USER table stores relationship
      -- among database users, application users,
      -- and people held in the HR person table.
      IF SYS_CONTEXT('apps_sec', 'hr_role') = 'EMP' THEN
        d_predicate = 'person_id IN
                      (SELECT employee_id FROM apps.fnd_user
                       WHERE user_name = SYS_CONTEXT(''userenv'', ''session_
user''))';
      -- for users with HR_ROLE set to MGR (manager), map
      -- security profile id to a list of employee id that
      -- the user can access
      ELSE IF SYS_CONTEXT('apps_sec', 'hr_role') = 'MGR' THEN
        d_predicate = 'person_id IN
                      (SELECT ppl.employee_id FROM per_person_list ppl WHERE
                       ppl.security_profile_id = SYS_CONTEXT(''apps_sec'',
''security_profile_id''))
                      OR EXISTS (SELECT NULL FROM apps.per security_profiles psp
WHERE
                       SYS_CONTEXT(''apps_sec'', ''security_profile_id'') =
                       psp.security_profile_id AND psp.view_all_flag = ''Y''))';
      ELSE
           d_predicate = '1=2';  -- deny access to other users, may use
something like 'keycol=null'
      END IF;
      RETURN d_predicate;
     END secure_person;
    END hr_security;
```

The next step is to associate a policy (here we call it PER_PEOPLE_SEC) for the PER_PEOPLE view to the HR_SECURITY.SECURE_PERSON function that generates the dynamic predicates:

```
DBMS_RLS.ADD_POLICY('apps', 'per_people', 'per_people_sec', 'apps'
                    'hr_security.secure_person', 'select, update, delete');
```

Now, any SELECT, UPDATE, and DELETE statement with the PER_PEOPLE view involved will pick up one of the three predicates based on the value of the application context HR_ROLE.

Note: The same security function that secured the PER_ALL_PEOPLE table can also be used to generate the dynamic predicates to secure the PER_ADDRESSES table, because they have the same policy to limit access to data.

42

DBMS_ROWID

The DBMS_ROWID package lets you create ROWIDs and get information about ROWIDs from PL/SQL programs and SQL statements. You can find the data block number, the object number, and other components of the ROWID without having to write code to interpret the base-64 character external ROWID.

Note: DBMS_ROWID is not to be used with universal ROWIDs (UROWIDs).

Usage Notes

Some of the functions in this package take a single parameter: a ROWID. This can be a character or a PL/SLQ ROWID, either restricted or extended, as required.

You can call the DBMS_ROWID functions and procedures from PL/SQL code, and you can also use the functions in SQL statements.

Note: ROWID_INFO is a procedure. It can only be used in PL/SQL code.

You can use functions from the DBMS_ROWID package just like any built-in SQL function; in other words, you can use them wherever an expression can be used. In this example, the ROWID_BLOCK_NUMBER function is used to return just the block number of a single row in the EMP table:

```
SELECT dbms_rowid.rowid_block_number(rowid)
    FROM emp
    WHERE ename = 'KING';
```

PL/SQL Example This example returns the ROWID for a row in the EMP table, extracts the data object number from the ROWID, using the ROWID_OBJECT function in the DBMS_ROWID package, then displays the object number:

```
DECLARE
  object_no   INTEGER;
  row_id      ROWID;
  ...
BEGIN
  SELECT ROWID INTO row_id FROM emp
    WHERE empno = 7499;
  object_no := dbms_rowid.rowid_object(row_id);
  dbms_output.put_line('The obj. # is '|| object_no);
  ...
```

Requirements

This package runs with the privileges of calling user, rather than the package owner ('sys').

ROWID Types

RESTRICTED	Restricted ROWID
EXTENDED	Extended ROWID

For example:

```
rowid_type_restricted constant integer := 0;
rowid_type_extended   constant integer := 1;
```

Note: Extended ROWIDs are only used in Oracle8*i* and above.

ROWID Verification Results

VALID	Valid ROWID
INVALID	Invalid ROWID

For example:

```
rowid_is_valid   constant integer := 0;
rowid_is_invalid constant integer := 1;
```

Object Types

UNDEFINED	Object Number not defined (for restricted ROWIDs)

For example:

```
rowid_object_undefined constant integer := 0;
```

ROWID Conversion Types

INTERNAL	Convert to / from column of ROWID type
EXTERNAL	Convert to / from string format

For example:

```
rowid_convert_internal constant integer := 0;
rowid_convert_external constant integer := 1;
```

Exceptions

ROWID_INVALID	Invalid rowid format
ROWID_BAD_BLOCK	Block is beyond end of file

For example:

```
ROWID_INVALID exception;
   pragma exception_init(ROWID_INVALID, -1410);

ROWID_BAD_BLOCK exception;
   pragma exception_init(ROWID_BAD_BLOCK, -28516);
```

Summary of Subprograms

Table 42–1 DBMS_ROWID Package Subprograms

Subprogram	Description
ROWID_CREATE function on page 42-5	Creates a ROWID, for testing only.
ROWID_INFO procedure on page 42-6	Returns the type and components of a ROWID.
ROWID_TYPE function on page 42-7	Returns the ROWID type: 0 is restricted, 1 is extended.
ROWID_OBJECT function on page 42-8	Returns the object number of the extended ROWID.
ROWID_RELATIVE_FNO function on page 42-9	Returns the file number of a ROWID.
ROWID_BLOCK_NUMBER function on page 42-10	Returns the block number of a ROWID.
ROWID_ROW_NUMBER function on page 42-10	Returns the row number.
ROWID_TO_ABSOLUTE_FNO function on page 42-11	Returns the absolute file number associated with the ROWID for a row in a specific table.
ROWID_TO_EXTENDED function on page 42-12	Converts a ROWID from restricted format to extended.
ROWID_TO_RESTRICTED function on page 42-14	Converts an extended ROWID to restricted format.

Table 42–1 DBMS_ROWID Package Subprograms

Subprogram	Description
ROWID_VERIFY function on page 42-14	Checks if a ROWID can be correctly extended by the ROWID_TO_EXTENDED function.

ROWID_CREATE function

This function lets you create a ROWID, given the component parts as parameters.

This is useful for testing ROWID operations, because only the Oracle Server can create a valid ROWID that points to data in a database.

Syntax

```
DBMS_ROWID.ROWID_CREATE (
   rowid_type    IN NUMBER,
   object_number IN NUMBER,
   relative_fno  IN NUMBER,
   block_number  IN NUMBER,
   row_number    IN NUMBER)
  RETURN ROWID;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_create,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 42–2 ROWID_CREATE Function Parameters

Parameter	Description
rowid_type	Type (restricted or extended). Set the rowid_type parameter to 0 for a restricted ROWID. Set it to 1 to create an extended ROWID. If you specify rowid_type as 0, then the required object_number parameter is ignored, and ROWID_CREATE returns a restricted ROWID.
object_number	Data object number (rowid_object_undefined for restricted).
relative_fno	Relative file number.
block_number	Block number in this file.

Table 42–2 ROWID_CREATE Function Parameters

Parameter	Description
file_number	File number in this block.

Example

Create a dummy extended ROWID:

```
my_rowid := DBMS_ROWID.ROWID_CREATE(1, 9999, 12, 1000, 13);
```

Find out what the rowid_object function returns:

```
obj_number := DBMS_ROWID.ROWID_OBJECT(my_rowid);
```

The variable obj_number now contains 9999.

ROWID_INFO procedure

This procedure returns information about a ROWID, including its type (restricted or extended), and the components of the ROWID. This is a procedure, and it cannot be used in a SQL statement.

Syntax

```
DBMS_ROWID.ROWID_INFO (
   rowid_in         IN   ROWID,
   rowid_type       OUT  NUMBER,
   object_number    OUT  NUMBER,
   relative_fno     OUT  NUMBER,
   block_number     OUT  NUMBER,
   row_number       OUT  NUMBER);
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_info,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 42–3 ROWID_INFO Procedure Parameters

Parameter	Description
rowid_in	ROWID to be interpreted. This determines if the ROWID is a restricted (0) or extended (1) ROWID.

Table 42–3 ROWID_INFO Procedure Parameters

Parameter	Description
rowid_type	Returns type (restricted / extended).
object_number	Returns data object number (rowid_object_undefined for restricted).
relative_fno	Returns relative file number.
block_number	Returns block number in this file.
file_number	Returns file number in this block.

See Also: "ROWID_TYPE function" on page 42-7

Example

This example reads back the values for the ROWID that you created in the ROWID_CREATE:

```
DBMS_ROWID.ROWID_INFO(my_rowid, rid_type, obj_num,
  file_num, block_num, row_num);

DBMS_OUTPUT.PUT_LINE('The type is ' || rid_type);
DBMS_OUTPUT.PUT_LINE('Data object number is ' || obj_num);
-- and so on...
```

ROWID_TYPE function

This function returns 0 if the ROWID is a restricted ROWID, and 1 if it is extended.

Syntax

```
DBMS_ROWID.ROWID_TYPE (
   rowid_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_type,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 42–4 ROWID_TYPE Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

Example

```
IF DBMS_ROWID.ROWID_TYPE(my_rowid) = 1 THEN
    my_obj_num := DBMS_ROWID.ROWID_OBJECT(my_rowid);
```

ROWID_OBJECT function

This function returns the data object number for an extended ROWID. The function returns zero if the input ROWID is a restricted ROWID.

Syntax

```
DBMS_ROWID.ROWID_OBJECT (
   rowid_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_object,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 42–5 ROWID_OBJECT Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

Note: The ROWID_OBJECT_UNDEFINED constant is returned for restricted ROWIDs.

Example

```
SELECT dbms_rowid.rowid_object(ROWID)
    FROM emp
    WHERE empno = 7499;
```

ROWID_RELATIVE_FNO function

This function returns the relative file number of the ROWID specified as the IN parameter. (The file number is relative to the tablespace.)

Syntax

```
DBMS_ROWID.ROWID_RELATIVE_FNO (
   rowid_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_relative_fno,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 42–6 ROWID_RELATIVE_FNO Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

Example

The example PL/SQL code fragment returns the relative file number:

```
DECLARE
    file_number    INTEGER;
    rowid_val      ROWID;
BEGIN
    SELECT ROWID INTO rowid_val
      FROM dept
      WHERE loc = 'Boston';
    file_number :=
      dbms_rowid.rowid_relative_fno(rowid_val);
    ...
```

ROWID_BLOCK_NUMBER function

This function returns the database block number for the input ROWID.

Syntax

```
DBMS_ROWID.ROWID_BLOCK_NUMBER (
   row_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_block_number,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 42–7 ROWID_BLOCK_NUMBER Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

Example

The example SQL statement selects the block number from a ROWID and inserts it into another table:

```
INSERT INTO T2 (SELECT dbms_rowid.rowid_block_number(ROWID)
    FROM some_table
    WHERE key_value = 42);
```

ROWID_ROW_NUMBER function

This function extracts the row number from the ROWID IN parameter.

Syntax

```
DBMS_ROWID.ROWID_ROW_NUMBER (
   row_id IN ROWID)
  RETURN NUMBER;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_row_number,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 42–8 ROWID_ROW_NUMBER Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.

Example

Select a row number:

```
SELECT dbms_rowid.rowid_row_number(ROWID)
    FROM emp
    WHERE ename = 'ALLEN';
```

ROWID_TO_ABSOLUTE_FNO function

This function extracts the absolute file number from a ROWID, where the file number is absolute for a row in a given schema and table. The schema name and the name of the schema object (such as a table name) are provided as IN parameters for this function.

Syntax

```
DBMS_ROWID.ROWID_TO_ABSOLUTE_FNO (
   row_id      IN ROWID,
   schema_name IN VARCHAR2,
   object_name IN VARCHAR2)
  RETURN NUMBER;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_to_absolute_fno,WNDS,WNPS,RNPS);
```

Parameters

Table 42–9 ROWID_TO_ABSOLUTE_FNO Function Parameters

Parameter	Description
row_id	ROWID to be interpreted.
schema_name	Name of the schema which contains the table.
object_name	Table name.

Example

```
DECLARE
    abs_fno        INTEGER;
    rowid_val      CHAR(18);
    object_name    VARCHAR2(20) := 'EMP';
BEGIN
    SELECT ROWID INTO rowid_val
    FROM emp
    WHERE empno = 9999;
     abs_fno := dbms_rowid.rowid_to_absolute_fno(
     rowid_val, 'SCOTT', object_name);
```

Note: For partitioned objects, the name must be a table name, not a partition or a sub/partition name.

ROWID_TO_EXTENDED function

This function translates a restricted ROWID that addresses a row in a schema and table that you specify to the extended ROWID format. Later, it may be removed from this package into a different place.

Syntax

```
DBMS_ROWID.ROWID_TO_EXTENDED (
   old_rowid       IN ROWID,
   schema_name     IN VARCHAR2,
   object_name     IN VARCHAR2,
   conversion_type IN INTEGER)
  RETURN ROWID;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_to_extended,WNDS,WNPS,RNPS);
```

Parameters

Table 42–10 ROWID_TO_EXTENDED Function Parameters

Parameter	Description
old_rowid	ROWID to be converted.
schema_name	Name of the schema which contains the table (optional).

Table 42–10 ROWID_TO_EXTENDED Function Parameters

Parameter	Description
object_name	Table name (optional).
conversion_type	rowid_convert_internal/external_convert_external (whether old_rowid was stored in a column of ROWID type, or the character string).

Returns

ROWID_TO_EXTENDED returns the ROWID in the extended character format. If the input ROWID is NULL, then the function returns NULL. If a zero-valued ROWID is supplied (00000000.0000.0000), then a zero-valued restricted ROWID is returned.

Example

Assume that there is a table called RIDS in the schema SCOTT, and that the table contains a column ROWID_COL that holds ROWIDs (restricted), and a column TABLE_COL that point to other tables in the SCOTT schema. You can convert the ROWIDs to extended format with the statement:

```
UPDATE SCOTT.RIDS
    SET rowid_col =
    dbms_rowid.rowid_to_extended (
       rowid_col, 'SCOTT", TABLE_COL, 0);
```

Usage Notes

If the schema and object names are provided as IN parameters, then this function verifies SELECT authority on the table named, and converts the restricted ROWID provided to an extended ROWID, using the data object number of the table. That ROWID_TO_EXTENDED returns a value, however, does not guarantee that the converted ROWID actually references a valid row in the table, either at the time that the function is called, or when the extended ROWID is actually used.

If the schema and object name are not provided (are passed as NULL), then this function attempts to fetch the page specified by the restricted ROWID provided. It treats the file number stored in this ROWID as the absolute file number. This can cause problems if the file has been dropped, and its number has been reused prior to the migration. If the fetched page belongs to a valid table, then the data object number of this table is used in converting to an extended ROWID value. This is very inefficient, and Oracle recommends doing this only as a last resort, when the target table is not known. The user must still know the correct table name at the time of using the converted value.

If an extended ROWID value is supplied, the data object number in the input extended ROWID is verified against the data object number computed from the table name parameter. If the two numbers do not match, the INVALID_ROWID exception is raised. If they do match, the input ROWID is returned.

> **See Also:** The ROWID_VERIFY function has a method to determine if a given ROWID can be converted to the extended format.

ROWID_TO_RESTRICTED function

This function converts an extended ROWID into restricted ROWID format.

Syntax

```
DBMS_ROWID.ROWID_TO_RESTRICTED (
   old_rowid       IN ROWID,
   conversion_type IN INTEGER)
  RETURN ROWID;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_to_restricted,WNDS,RNDS,WNPS,RNPS);
```

Parameters

Table 42–11 ROWID_TO_RESTRICTED Function Parameters

Parameter	Description
old_rowid	ROWID to be converted.
conversion_type	Internal or external - format of returned ROWID. rowid_convert_internal/external_convert_external (whether returned ROWID will be stored in a column of ROWID type or the character string)

ROWID_VERIFY function

This function verifies the ROWID. It returns 0 if the input restricted ROWID can be converted to extended format, given the input schema name and table name, and it returns 1 if the conversion is not possible.

Note: You can use this function in a WHERE clause of a SQL statement, as shown in the example.

Syntax

```
DBMS_ROWID.ROWID_VERIFY (
   rowid_in        IN ROWID,
   schema_name     IN VARCHAR2,
   object_name     IN VARCHAR2,
   conversion_type IN INTEGER
  RETURN NUMBER;
```

Pragmas

```
pragma RESTRICT_REFERENCES(rowid_verify,WNDS,WNPS,RNPS);
```

Parameters

Table 42–12 ROWID_VERIFY Function Parameters

Parameter	Description
rowid_in	ROWID to be verified.
schema_name	Name of the schema which contains the table.
object_name	Table name.
conversion_type	rowid_convert_internal/external_convert_external (whether old_rowid was stored in a column of ROWID type or the character string).

Example

Considering the schema in the example for the ROWID_TO_EXTENDED function, you can use the following statement to find bad ROWIDs prior to conversion. This enables you to fix them beforehand.

```
SELECT ROWID, rowid_col
    FROM SCOTT.RIDS
    WHERE dbms_rowid.rowid_verify(rowid_col, NULL, NULL, 0) =1;
```

See Also: Chapter 59, "UTL_RAW", Chapter 60, "UTL_REF"

43

DBMS_SESSION

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

Requirements

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

Summary of Subprograms

Table 43–1 DBMS_SESSION Subprograms

Subprogram	Description
SET_ROLE procedure on page 43-3	Sets role.
SET_SQL_TRACE procedure on page 43-3	Turns tracing on or off.
SET_NLS procedure on page 43-4	Sets national language support (NLS).
CLOSE_DATABASE_LINK procedure on page 43-4	Closes database link.
RESET_PACKAGE procedure on page 43-5	Deinstantiates all packages in the session.
UNIQUE_SESSION_ID function on page 43-6	Returns an identifier that is unique for all sessions currently connected to this database.
IS_ROLE_ENABLED function on page 43-7	Determines if the named role is enabled for the session.
IS_SESSION_ALIVE function on page 43-7	Determines if the specified session is alive.
SET_CLOSE_CACHED_OPEN_CURSORS procedure on page 43-8	Turns `close_cached_open_cursors` on or off.
FREE_UNUSED_USER_MEMORY procedure on page 43-8	Lets you reclaim unused memory after performing operations requiring large amounts of memory.
SET_CONTEXT procedure on page 43-11	Sets or resets the value of a context attribute.
LIST_CONTEXT procedure on page 43-12	Returns a list of active namespace and context for the current session.
SWITCH_CURRENT_CONSUMER_GROUP procedure on page 43-13	Facilitates changing the current resource consumer group of a user's current session.

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 43–2 SET_ROLE Procedure Parameters

Parameter	Description
role_cmd	This text is appended to "set role" and then run as SQL.

SET_SQL_TRACE procedure

This procedure turns tracing on or off. It is equivalent to the following SQL statement:

```
ALTER SESSION SET SQL_TRACE ...
```

Syntax

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

Parameters

Table 43–3 SET_SQL_TRACE Procedure Parameters

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

SET_NLS procedure

This procedure sets up your national language support (NLS). It is equivalent to the following SQL statement:

```
ALTER SESSION SET <nls_parameter> = <value>
```

Syntax

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

Parameters

Table 43–4 SET_NLS Procedure Parameters

Parameter	Description
param	NLS 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''')"

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 43–5 CLOSE_DATABASE_LINK Procedure Parameters

Parameter	Description
dblink	Name of the database link to close.

RESET_PACKAGE procedure

This procedure deinstantiates all packages in this session: It frees all package state.

Memory used for caching execution state is associated with all PL/SQL functions, procedures, and packages that have been 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;
```

Parameters

None.

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 will 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 recreate 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).

Example

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;
```

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;
```

Parameters

None.

Pragmas

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

Returns

Table 43–6 UNIQUE_SESSION_ID Function Returns

Return	Description
unique_session_id	Returns up to 24 bytes.

IS_ROLE_ENABLED function

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

Syntax

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

Parameters

Table 43–7 IS_ROLE_ENABLED Function Parameters

Parameter	Description
rolename	Name of the role.

Returns

Table 43–8 IS_ROLE_ENABLED Function Returns

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 alive.

Syntax

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

Parameters

Table 43–9 IS_SESSION_ALIVE Function Parameters

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

Returns

Table 43–10 IS_SESSION_ALIVE Function Returns

Return	Description
is_session_alive	TRUE or FALSE, depending on whether the session is alive.

SET_CLOSE_CACHED_OPEN_CURSORS procedure

This procedure turns close_cached_open_cursors on or off. It is equivalent to the following SQL statement:

```
ALTER SESSION SET CLOSE_CACHED_OPEN_CURSORS ...
```

Syntax

```
DBMS_SESSION.SET_CLOSE_CACHED_OPEN_CURSORS (
   close_cursors BOOLEAN);
```

Parameters

Table 43–11 SET_CLOSE_CACHED_OPEN_CURSORS Procedure Parameters

Parameter	Description
close_cursors	TRUE or FALSE

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;
```

Parameters

None.

Returns

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.
- **MTS 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.

Example

The PL/SQL fragment below 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;
```

SET_CONTEXT procedure

This procedure sets or resets the value of a context attribute.

Syntax

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

Parameters

Table 43–12 SET_CONTEXT Procedure Parameters

Parameter	Description
namespace	Name of the namespace to use for the application context (limited to 30 bytes).
attribute	Name of the attribute to be set (limited to 30 bytes).
value	Value to be set (limited to 256 bytes).

Usage Notes

The caller of this function must be in the calling stack of a procedure which has been associated to the context namespace through a CREATE CONTEXT statement. The checking of the calling stack does not cross DBMS boundary.

There is no limit on the number of attributes that can be set in a namespace. An attribute value remains for user session, or until it is reset by the user.

LIST_CONTEXT procedure

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 43–13 LIST_CONTEXT Procedure Parameters

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

Returns

Table 43–14 LIST_CONTEXT Procedure Returns

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.

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 43–15 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.

Returns

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

> **Note:** You can switch back to the old consumer group later using the value returned in old_consumer_group.

Exceptions

Table 43–16 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.

Example

```
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 will be thrown, but the consumer group will 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;
/
```

44

DBMS_SHARED_POOL

DBMS_SHARED_POOL provides access to the shared pool, which is the shared memory area where cursors and PL/SQL objects are stored. DBMS_SHARED_POOL enables you to display the sizes of objects in the shared pool, and mark them for keeping or unkeeping in order to reduce memory fragmentation.

Installation Notes

To create DBMS_SHARED_POOL, run the DBMSPOOL.SQL script. The PRVTPOOL.PLB script is automatically executed after DBMSPOOL.SQL runs. These scripts are *not* run by CATPROC.SQL.

Usage Notes

The procedures provided here may be useful when loading large PL/SQL objects. When large PL/SQL objects are loaded, users response time is affected because of the large number of smaller objects that need to be aged out from the shared pool to make room (due to memory fragmentation). In some cases, there may be insufficient memory to load the large objects.

DBMS_SHARED_POOL is also useful for frequently executed triggers. You may want to keep compiled triggers on frequently used tables in the shared pool. Additionally, DBMS_SHARED_POOL supports sequences. Sequence numbers are lost when a sequence is aged out of the shared pool. DBMS_SHARED_POOL is useful for keeping sequences in the shared pool and thus preventing the loss of sequence numbers.

Summary of Subprograms

Table 44–1 DBMS_SHARED_POOL Package Subprograms

Subprogram	Description
SIZES procedure on page 44-2	Shows objects in the shared pool that are larger than the specified size
KEEP procedure on page 44-3	Keeps an object in the shared pool
UNKEEP procedure on page 44-5	Unkeeps the named object
ABORTED_REQUEST_THRESHOLD procedure on page 44-5	Sets the aborted request threshold for the shared pool

SIZES procedure

This procedure shows objects in the shared_pool that are larger than the specified size. The name of the object is also given, which can be used as an argument to either the KEEP or UNKEEP calls below.

Syntax

```
DBMS_SHARED_POOL.SIZES (
   minsize NUMBER);
```

Parameters

Table 44–2 SIZES Procedure Parameters

Parameter	Description
minsize	Size, in kilobytes, over which an object must be occupying in the shared pool, in order for it to be displayed.

Usage Notes

Issue the SQLDBA or SQLPLUS 'SET SERVEROUTPUT ON SIZE XXXXX' command prior to using this procedure so that the results are displayed.

KEEP procedure

This procedure keeps an object in the shared pool. Once an object has been kept in the shared pool, it is not subject to aging out of the pool. This may be useful for certain semi-frequently used large objects, because when large objects are brought into the shared pool, a larger number of other objects (much more than the size of the object being brought in) may need to be aged out in order to create a contiguous area large enough.

Syntax

```
DBMS_SHARED_POOL.KEEP (
   name VARCHAR2,
   flag CHAR      DEFAULT 'P');
```

> **Note:** This procedure may not be supported in the future if automatic mechanisms are implemented to make this unnecessary.

Parameters

Table 44–3 KEEP Procedure Parameters

Parameter	Description
name	Name of the object to keep. The value for this identifier is the concatenation of the address and hash_value columns from the v$sqlarea view. This is displayed by the SIZES procedure. Currently, TABLE and VIEW objects may not be keeped.
flag	(Optional) If this is not specified, then the package assumes that the first parameter is the name of a package / procedure / function and resolves the name. Set to 'P' or 'p' to fully specify that the input is the name of a package / procedure / function. Set to 'T' or 't' to specify that the input is the name of a type. Set to 'R' or 'r' to specify that the input is the name of a trigger. Set to 'Q' or 'q' to specify that the input is the name of a sequence. In case the first argument is a cursor address and hash-value, the parameter should be set to any character except 'P' or 'p' or 'Q' or 'q' or 'R' or 'r' or 'T' or 't'.

Exceptions

An exception is raised if the named object cannot be found.

Usage Notes

There are two kinds of objects:

- PL / SQL objects, triggers, sequences, and types which are specified by name
- SQL cursor objects which are specified by a two-part number (indicating a location in the shared pool).

For example:

```
DBMS_SHARED_POOL.KEEP('scott.hispackage')
```

This keeps package HISPACKAGE, owned by SCOTT. The names for PL / SQL objects follow SQL rules for naming objects (i.e., delimited identifiers, multi-byte names, etc. are allowed). A cursor can be keeped by DBMS_SHARED_

POOL.KEEP('0034CDFF, 20348871'). The complete hexadecimal address must be in the first 8 characters.

UNKEEP procedure

This procedure unkeeps the named object.

Syntax

```
DBMS_SHARED_POOL.UNKEEP (
   name VARCHAR2,
   flag CHAR     DEFAULT 'P');
```

> **Caution:** This procedure may not be supported in the future if automatic mechanisms are implemented to make this unnecessary.

Parameters

Table 44–4 UNKEEP Procedure Parameters

Parameter	Description
name	Name of the object to unkeep. See description of the name object for the KEEP procedure.
flag	See description of the flag parameter for the KEEP procedure.

Exceptions

An exception is raised if the named object cannot be found.

ABORTED_REQUEST_THRESHOLD procedure

This procedure sets the aborted request threshold for the shared pool.

Syntax

```
DBMS_SHARED_POOL.ABORTED_REQUEST_THRESHOLD (
   threshold_size NUMBER);
```

Parameters

Table 44–5 ABORTED_REQUEST_THRESHOLD Procedure Parameters

Parameter	Description
threshold_size	Size, in bytes, of a request which does not try to free unpinned (not "unkeep-ed") memory within the shared pool. The range of threshold_size is 5000 to ~2 GB inclusive.

Exceptions

An exception is raised if the threshold is not in the valid range.

Usage Notes

Usually, if a request cannot be satisfied on the free list, then the RDBMS tries to reclaim memory by freeing objects from the LRU list and checking periodically to see if the request can be fulfilled. After finishing this step, the RDBMS has performed a near equivalent of an 'ALTER SYSTEM FLUSH SHARED_POOL'.

Because this impacts all users on the system, this procedure "localizes" the impact to the process failing to find a piece of shared pool memory of size greater than thresh_hold size. This user gets the 'out of memory' error without attempting to search the LRU list.

45

DBMS_SNAPSHOT

DBMS_SNAPSHOT enables you to refresh snapshots that are not part of the same refresh group and purge logs.

Note: DBMS_MVIEW is a synonym for DBMS_SNAPSHOT. This synonym may be used in the future with data warehousing.

Summary of Subprograms

Table 45–1 DBMS_SNAPSHOT Package Subprograms

Subprogram	Description
BEGIN_TABLE_REORGANIZATION procedure on page 45-2	Performs a process to preserve snapshot data needed for refresh.
END_TABLE_REORGANIZATION on page 45-3	Ensures that the snapshot data for the master table is valid and that the master table is in the proper state.
I_AM_A_REFRESH function on page 45-4	Returns the value of the I_AM_REFRESH package state.
PURGE_DIRECT_LOAD_LOG procedure on page 45-4	Purges rows from the direct loader log after they are no longer needed by any snapshots (used with data warehousing).
PURGE_LOG procedure on page 45-4	Purges rows from the snapshot log.
PURGE_SNAPSHOT_FROM_LOG procedure on page 45-5	Purges rows from the snapshot log.
REFRESH procedure on page 45-7	Refreshes a list of snapshots.
REFRESH_ALL_MVIEWS procedure on page 45-9	Refreshes all snapshots that have not been refreshed because the most recent bulk load to a dependent detail table.
REFRESH_DEPENDENT procedure on page 45-10	Refreshes all table-based snapshots that depend on a specified detail table or list of detail tables.
REGISTER_SNAPSHOT procedure on page 45-12	Enables the administration of individual snapshots.
UNREGISTER_SNAPSHOT procedure on page 45-14	Enables the administration of individual snapshots. Invoked at a master site to unregister a snapshot.

BEGIN_TABLE_REORGANIZATION procedure

This procedure performs a process to preserve snapshot data needed for refresh. It must be called before a master table is reorganized.

> **See Also:** See "Administering a Replicated Environment" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION (
   tabowner     IN   VARCHAR2,
   tabname      IN   VARCHAR2);
```

Parameters

Table 45–2 BEGIN_TABLE_REORGANIZATION Procedure Parameters

Parameter	Description
tabowner	Owner of the table being reorganized.
tabname	Name of the table being reorganized.

END_TABLE_REORGANIZATION

This procedure must be called after a master table is reorganized. It ensures that the snapshot data for the master table is valid and that the master table is in the proper state.

> **See Also:** See "Administering a Replicated Environment" in the *Oracle8i Replication* manual.

Syntax

```
DBMS_SNAPSHOT.END_TABLE_REORGANIZATION (
   tabowner     IN   VARCHAR2,
   tabname      IN   VARCHAR2);
```

Parameters

Table 45–3 END_TABLE_REORGANIZATION Procedure Parameters

Parameter	Description
tabowner	Owner of the table being reorganized.
tabname	Name of the table being reorganized.

I_AM_A_REFRESH function

This function returns the value of the I_AM_REFRESH package state.

Syntax

```
DBMS_SNAPSHOT.I_AM_A_REFRESH
  RETURN BOOLEAN;
```

Parameters

None

Returns

A return value of TRUE indicates that all local replication triggers for snapshots are effectively disabled in this session because each replication trigger first checks this state. A return value of FALSE indicates that these triggers are enabled.

PURGE_DIRECT_LOAD_LOG procedure

This procedure remove entries from the direct loader log after they are no longer needed for any known snapshot (materialized view). This procedure will usually be used in environments using Oracle's Data Warehousing technology.

See Also: For more information, see *Oracle8i Tuning*.

Syntax

```
DBMS_SNAPSHOT.PURGE_DIRECT_LOAD_LOG;
```

Parameters

None.

PURGE_LOG procedure

This procedure purges rows from the snapshot log.

Syntax

```
DBMS_SNAPSHOT.PURGE_LOG (
   master        IN   VARCHAR2,
   num           IN   BINARY_INTEGER := 1,
   flag          IN   VARCHAR2       := 'NOP');
```

Parameters

Table 45–4 PURGE_LOG Procedure Parameters

Parameter	Description
master	Name of the master table.

num	Number of least recently refreshed snapshots whose rows you want to remove from snapshot log. For example, the following statement deletes rows needed to refresh the two least recently refreshed snapshots:
	`dbms_snapshot.purge_log('master_table', 2);`
	To delete all rows in the snapshot log, indicate a high number of snapshots to disregard, as in this example:
	`dbms_snapshot.purge_log('master_table',9999);`
	This statement completely purges the snapshot log that corresponds to MASTER_TABLE if fewer than 9999 snapshots are based on MASTER_TABLE. A simple snapshot whose rows have been purged from the snapshot log must be completely refreshed the next time it is refreshed.

flag	Specify DELETE to guarantee that rows are deleted from the snapshot log for at least one snapshot. This argument can override the setting for the argument num. For example, the following statement deletes rows from the least recently refreshed snapshot that actually has dependent rows in the snapshot log: `dbms_snapshot.purge_log('master_table',1,'DELETE');`

PURGE_SNAPSHOT_FROM_LOG procedure

This procedure is called on the master site to delete the rows in snapshot refresh related data dictionary tables maintained at the master site for the specified snapshot identified by its snapshot_id or the combination of the snapowner, snapname, and the snapsite. If the snapshot specified is the oldest snapshot to have refreshed from any of the master tables, then the snapshot log is also purged. This procedure does not unregister the snapshot.

In case there is an error while purging one of the snapshot logs, the successful purge operations of the previous snapshot logs are not rolled back. This is to minimize the size of the snapshot logs. In case of an error, this procedure can be invoked again until all the snapshot logs are purged.

Syntax

```
DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
   snapshot_id    IN   BINARY_INTEGER  |
   snapowner      IN   VARCHAR2,
   snapname       IN   VARCHAR2,
   snapsite       IN   VARCHAR2);
```

Parameters

Table 45–5 PURGE_SNAPSHOT_FROM_LOG Procedure Parameters

Parameter	Description
snapshot_id	If you want to execute this procedure based on the ID of the target snapshot, then specify the snapshot ID using the snapshot_id parameter. Query the DBA_SNAPSHOT_LOGS view at the snapshot log site for a listing of snapshot IDs. Executing this procedure based on the snapshot ID is useful if the target snapshot is not listed in the list of registered snapshots (DBA_REGISTERED_SNAPSHOTS). If you specify a snapshot ID, then do not specify values for the snapowner, snapname, or snapsite parameters.
snapowner	If do not specify a snapshot_id, enter the owner of the target snapshot using the snapowner parameter. Query the DBA_REGISTERED_SNAPSHOTS view at the snapshot log site to view the snapshot owners.
snapname	If do not specify a snapshot_id, enter the name of the target snapshot using the snapname parameter. Query the DBA_REGISTERED_SNAPSHOTS view at the snapshot log site to view the snapshot names.
snapsite	If do not specify a snapshot_id, then enter the site of the target snapshot using the snapsite parameter. Query the DBA_REGISTERED_SNAPSHOTS view at the snapshot log site to view the snapshot sites.

REFRESH procedure

This procedure refreshes a list of snapshots.

Syntax

```
DBMS_SNAPSHOT.REFRESH (
   { list                IN     VARCHAR2,
   | tab                 IN OUT DBMS_UTILITY.UNCL_ARRAY,}
   method                IN     VARCHAR2       := NULL,
   rollback_seg          IN     VARCHAR2       := NULL,
   push_deferred_rpc     IN     BOOLEAN        := TRUE,
   refresh_after_errors  IN     BOOLEAN        := FALSE,
   purge_option          IN     BINARY_INTEGER := 1,
   parallelism           IN     BINARY_INTEGER := 0,
   heap_size             IN     BINARY_INTEGER := 0,
   atomic_refresh        IN     BOOLEAN        := TRUE);
```

Parameters

Table 45–6 REFRESH Procedure Parameters

Parameter	Description
list tab	Comma-separated list of snapshots that you want to refresh. (Synonyms are not supported.) These snapshots can be located in different schemas and have different master tables; however, all of the listed snapshots must be in your local database. Alternatively, you may pass in a PL/SQL table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a snapshot.
method	A string of refresh methods indicating how to refresh the listed snapshots. `F' or `f' indicates fast refresh, `?' indicates force refresh, `C' or `c' indicates complete refresh, and `A' or `a' indicates always refresh, If a snapshot does not have a corresponding refresh method (that is, if more snapshots are specified than refresh methods), then that snapshot is refreshed according to its default refresh method. For example, the following EXECUTE statement within SQL*Plus: `dbms_snapshot.refresh ('s_emp,s_dept,scott.s_salary','CF');` performs a complete refresh of the S_EMP snapshot, a fast refresh of the S_DEPT snapshot, and a default refresh of the SCOTT.S_SALARY snapshot.
rollback_seg	Name of the snapshot site rollback segment to use while refreshing snapshots.

Table 45–6 REFRESH Procedure Parameters

Parameter	Description
push_deferred_rpc	Used by updatable snapshots only. Set this to TRUE if you want to push changes from the snapshot to its associated master before refreshing the snapshot. Otherwise, these changes may appear to be temporarily lost.
refresh_after_errors	If this is TRUE, then an updatable snapshot will continue to refresh even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master table. If this is TRUE and if atomic_refresh is FALSE, then this procedure will continue to refresh other snapshots if it fails while refreshing a snapshot.
purge_option	If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), then 0 means do not purge, 1 means lazy purge, and 2 means aggressive purge. In most cases, lazy purge is the optimal setting. Set purge to aggressive to trim the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, then set purge to do not purge and occasionally execute PUSH with purge set to aggressive to reduce the queue.
parallelism	0 means serial propagation, n > 0 means parallel propagation with n parallel server processes, and 1 means parallel propagation using only one parallel server process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set this parameter unless directed to do so by Oracle Worldwide Support.
atomic_refresh	If this is set to TRUE, then the list of snapshots will be refreshed in a single transaction. All of the refreshed snapshots will be updated to a single point in time. If the refresh fails for any of the snapshots, then none of the snapshots will be updated. If this is set to FALSE, then each of the snapshots will be refreshed in a separate transaction. The number of job queue processes must be set to 1 or greater if this parameter is FALSE. If FALSE, and if the Summary Management option is not purchased, then an error is raised.

REFRESH_ALL_MVIEWS procedure

This procedure refreshes all snapshots (materialized views) with the following properties:

- The snapshot has not been refreshed since the most recent change to a detail table on which it depends.
- The snapshot and all of the detail tables on which it depends are local.
- The snapshot is in the view DBA_MVIEW_ANALYSIS.

This procedure is intended for use with data warehouses.

Syntax

```
DBMS_SNAPSHOT.REFRESH_ALL_MVIEWS (
   number_of_failures     OUT   BINARY_INTEGER,
   method                 IN    VARCHAR2         := NULL,
   rollback_seg           IN    VARCHAR2         := NULL,
   refresh_after_errors   IN    BOOLEAN          := FALSE,
   atomic_refresh         IN    BOOLEAN          := TRUE);
```

Parameters

Table 45–7 REFRESH_ALL_MVIEWS Procedure Parameters

Parameter	Description
number_of_failures	Returns the number of failures that occurred during processing.
method	A single refresh method indicating the type of refresh to perform for each snapshot that is refreshed. `F' or `f' indicates fast refresh, `?' indicates force refresh, `C' or `c' indicates complete refresh, and `A' or `a' indicates always refresh. If no method is specified, then a snapshot is refreshed according to its default refresh method.
rollback_seg	Name of the snapshot site rollback segment to use while refreshing snapshots.
refresh_after_errors	If this is TRUE, then an updatable snapshot will continue to refresh even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master table. If this is TRUE, and if atomic_refresh is FALSE, then this procedure continues to refresh other snapshots if it fails while refreshing a snapshot.

Table 45–7 REFRESH_ALL_MVIEWS Procedure Parameters

Parameter	Description
atomic_refresh	If this is set to TRUE, then the refreshed snapshots will be refreshed in a single transaction. All of the refreshed snapshots will be updated to a single point in time. If the refresh fails for any of the snapshots, then none of the snapshots will be updated. If this is set to FALSE, then each of the refreshed snapshots will be refreshed in a separate transaction. The number of job queue processes must be set to 1 or greater if this parameter is FALSE. If FALSE, if and the Summary Management option is not purchased, then an error is raised.

REFRESH_DEPENDENT procedure

This procedure refreshes all snapshots (materialized views) with the following properties:

- The snapshot depends on a detail table in the list of specified detail tables.
- The snapshot has not been refreshed since the most recent change to a detail table on which it depends.
- The snapshot and all of the detail tables on which it depends are local.
- The snapshot is in the view DBA_MVIEW_ANALYSIS.

This procedure is intended for use with data warehouses.

Syntax

```
DBMS_SNAPSHOT.REFRESH_DEPENDENT (
   number_of_failures     OUT    BINARY_INTEGER,
   { list                 IN     VARCHAR2,
   | tab                  IN OUT DBMS_UTILITY.UNCL_ARRAY,}
   method                 IN     VARCHAR2    := NULL,
   rollback_seg           IN     VARCHAR2    := NULL,
   refresh_after_errors   IN     BOOLEAN     := FALSE,
   atomic_refresh         IN     BOOLEAN     := TRUE);
```

Parameters

Table 45–8 REFRESH_DEPENDENT Procedure Parameters

Parameter	Description
number_of_failures	Returns the number of failures that occurred during processing.
list tab	Comma-separated list of detail tables on which snapshots can depend. (Synonyms are not supported.) These tables and the snapshots that depend on them can be located in different schemas. However, all of the tables and snapshots must be in your local database. Alternatively, you may pass in a PL/SQL table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a table.
method	A string of refresh methods indicating how to refresh the dependent snapshots. All of the snapshots that depend on a particular table are refreshed according to the refresh method associated with that table. `F' or `f' indicates fast refresh, `?' indicates force refresh, `C' or `c' indicates complete refresh, and `A' or `a' indicates always refresh. If a table does not have a corresponding refresh method (that is, if more tables are specified than refresh methods), then any snapshot that depends on that table is refreshed according to its default refresh method. For example, the following EXECUTE statement within SQL*Plus: `dbms_snapshot.refresh_dependent ('emp,dept,scott.salary','CF');` performs a complete refresh of the snapshots that depend on the EMP table, a fast refresh of the snapshots that depend on the DEPT table, and a default refresh of the snapshots that depend on the SCOTT.SALARY table.
rollback_seg	Name of the snapshot site rollback segment to use while refreshing snapshots.
refresh_after_errors	If this is TRUE, then an updatable snapshot will continue to refresh even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master table. If this is TRUE, and if atomic_refresh is FALSE, then this procedure will continue to refresh other snapshots if it fails while refreshing a snapshot.

Table 45–8 REFRESH_DEPENDENT Procedure Parameters

Parameter	Description
atomic_refresh	If this is set to TRUE, then the refreshed snapshots will be refreshed in a single transaction. All of the refreshed snapshots will be updated to a single point in time. If the refresh fails for any of the snapshots, then none of the snapshots will be updated. If this is set to FALSE, then each of the refreshed snapshots will be refreshed in separate transactions. The number of job queue processes must be set to 1 or greater if this parameter is FALSE. If FALSE, if and the Summary Management option is not purchased, then an error is raised.

REGISTER_SNAPSHOT procedure

This procedure enables the administration of individual snapshots.

Syntax

```
DBMS_SNAPSHOT.REGISTER_SNAPSHOT (
   snapowner    IN   VARCHAR2,
   snapname     IN   VARCHAR2,
   snapsite     IN   VARCHAR2,
   snapshot_id  IN   DATE | BINARY_INTEGER,
   flag         IN   BINARY_INTEGER,
   qry_txt      IN   VARCHAR2,
   rep_type     IN   BINARY_INTEGER := DBMS_SNAPSHOT.REG_UNKNOWN);
```

Note: This procedure is overloaded. The `snapshot_id` and `flag` parameters are mutually exclusive.

Parameters

Table 45–9 REGISTER_SNAPSHOT Procedure Parameters

Parameter	Description
sowner	Owner of the snapshot.
snapname	Name of the snapshot.
snapsite	Name of the snapshot site for a snapshot registering at an Oracle8 or greater master. This should not contain any double quotes.

Table 45–9 REGISTER_SNAPSHOT Procedure Parameters

Parameter	Description
snapshot_id	The identification number of the snapshot. Specify an Oracle8 snapshot as a BINARY_INTEGER; specify an Oracle7 snapshot registering at an Oracle8 or greater master sites as a DATE.
flag	PL/SQL package variable indicating whether subsequent create or move commands are registered in the query text.
query_txt	The first 32,000 bytes of the query.
rep_type	Version of the snapshot. Valid constants that can be assigned include reg_uknown (the default), reg_v7_group, reg_v8_group, and reg_repapi_group.

Usage Notes

This procedure is executed at the master site, and can be done by a remote procedure call. If REGISTER_SNAPSHOT is called multiple times with the same SNAPOWNER, SNAPNAME, and SNAPSITE, then the most recent values for SNAPSHOT_ID, FLAG, and QUERY_TXT are stored. If a query exceeds the maximum VARCHAR2 size, then QUERY_TXT contains the first 32000 characters of the query and the remainder is truncated. When invoked manually, the values of SNAPSHOT_ID and FLAG have to be looked up in the snapshot views by the person who calls the procedure.

If you do *not* want the snapshot query registered at the master site, then call the SET_REGISTER_QUERY_TEXT procedure with the option set to FALSE. To see the most recent setting of the option, call the GET_REG_QUERY_TEXT_FLAG function at the snapshot site before issuing the DDL.

UNREGISTER_SNAPSHOT procedure

This procedure enables the administration of individual snapshots. Invoked at a master site to unregister a snapshot.

Syntax

```
DBMS_SNAPSHOT.UNREGISTER_SNAPSHOT (
   snapowner       IN   VARCHAR2,
   snapname        IN   VARCHAR2,
   snapsite        IN   VARCHAR2);
```

Parameters

Table 45–10 UNREGISTER_SNAPSHOT Procedure Parameters

Parameters	Description
snapowner	Owner of the snapshot.
snapname	Name of the snapshot.
snapsite	Name of the snapshot site.

46

DBMS_SPACE

The DBMS_SPACE package enables you to analyze segment growth and space requirements.

Security

This package runs with SYS privileges.

Requirements

The execution privilege is granted to PUBLIC. Subprograms in this package run under the caller security. The user must have ANALYZE privilege on the object.

Summary of Subprograms

Table 46–1 DBMS_ALERT Package Subprograms

Subprogram	Description
UNUSED_SPACE procedure on page 46-2	Returns information about unused space in an object (table, index, or cluster).
FREE_BLOCKS procedure on page 46-3	Returns information about free blocks in an object (table, index, or cluster).

UNUSED_SPACE procedure

This procedure returns information about unused space in an object (table, index, or cluster).

Syntax

```
DBMS_SPACE.UNUSED_SPACE (
   segment_owner              IN  VARCHAR2,
   segment_name               IN  VARCHAR2,
   segment_type               IN  VARCHAR2,
   total_blocks               OUT NUMBER,
   total_bytes                OUT NUMBER,
   unused_blocks              OUT NUMBER,
   unused_bytes               OUT NUMBER,
   last_used_extent_file_id   OUT NUMBER,
   last_used_extent_block_id  OUT NUMBER,
   last_used_block            OUT NUMBER,
   partition_name             IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 46–2 UNUSED_SPACE Procedure Parameters

Parameter	Description
segment_owner	Schema name of the segment to be analyzed.
segment_name	Segment name of the segment to be analyzed.
segment_type	Type of the segment to be analyzed: TABLE TABLE PARTITION TABLE SUBPARTITION INDEX INDEX PARTITION INDEX SUBPARTITION CLUSTER LOB
total_blocks	Returns total number of blocks in the segment.
total_bytes	Returns total number of blocks in the segment, in bytes.
unused_blocks	Returns number of blocks which are not used.
unused_bytes	Returns, in bytes, number of blocks which are not used.
last_used_extent_ file_ id	Returns the file ID of the last extent which contains data.
last_used_extent_ block_id	Returns the block ID of the last extent which contains data.
last_used_block	Returns the last block within this extent which contains data.
partition_name	Partition name of the segment to be analyzed. This is only used for partitioned tables; the name of subpartition should be used when partitioning is compose.

FREE_BLOCKS procedure

This procedure returns information about free blocks in an object (table, index, or cluster).

Syntax

```
DBMS_SPACE.FREE_BLOCKS (
   segment_owner     IN  VARCHAR2,
   segment_name      IN  VARCHAR2,
   segment_type      IN  VARCHAR2,
   freelist_group_id IN  NUMBER,
   free_blks         OUT NUMBER,
   scan_limit        IN  NUMBER DEFAULT NULL,
   partition_name    IN  VARCHAR2 DEFAULT NULL);
```

Pragmas

```
pragma restrict_references(free_blocks,WNDS);
```

Parameters

Table 46–3 FREE_BLOCKS Procedure Parameters

Parameter	Description
segment_owner	Schema name of the segment to be analyzed.
segment_name	Segment name of the segment to be analyzed.
segment_type	Type of the segment to be analyzed: TABLE TABLE PARTITION TABLE SUBPARTITION INDEX INDEX PARTITION INDEX SUBPARTITION CLUSTER LOB
freelist_group_id	Freelist group (instance) whose free list size is to be computed.
free_blks	Returns count of free blocks for the specified group.
scan_limit	Maximum number of free list blocks to read (optional). Use a scan limit of X you are interested only in the question, "Do I have X blocks on the free list?"
partition_name	Partition name of the segment to be analyzed. This is only used for partitioned tables; the name of subpartition should be used when partitioning is compose.

Examples

Example 1

The following declares the necessary bind variables and executes.

```
DBMS_SPACE.UNUSED_SPACE('SCOTT', 'EMP', 'TABLE', :total_blocks,
   :total_bytes,:unused_blocks, :unused_bytes, :lastextf,
   :last_extb, :lastusedblock);
```

This fills the unused space information for bind variables in EMP table in SCOTT schema.

Example 2

The following uses the CLUS cluster in SCOTT schema with 4 freelist groups. It returns the number of blocks in freelist group 3 in CLUS.

```
DBMS_SPACE.FREE_BLOCKS('SCOTT', 'CLUS', 'CLUSTER', 3, :free_blocks);
```

Note: An error is raised if `scan_limit` is not a positive number.

47

DBMS_SPACE_ADMIN

The DBMS_SPACE_ADMIN package provides functionality for locally managed tablespaces.

Security

This package runs with SYS privileges; therefore, any user who has privilege to execute the package can manipulate the bitmaps.

Constants

SEGMENT_VERIFY_EXTENTS	Verifies that the space owned by segment is appropriately reflected in the bitmap as used.
SEGMENT_VERIFY_EXTENTS_GLOBAL	Verifies that the space owned by segment is appropriately reflected in the bitmap as used and that no other segment claims any of this space to be used by it.
SEGMENT_MARK_CORRUPT	Marks a temp segment as corrupt whereby facilitating its elimination from the dictionary (without space reclaim).
SEGMENT_MARK_VALID	Marks a corrupt temp segment as valid. It is useful when the corruption in the segment extent map or elsewhere has been resolved and the segment can be dropped normally.
SEGMENT_DUMP_EXTENT_MAP	Dumps the extent map for a given segment.
TABLESPACE_VERIFY_BITMAP	Verifies the bitmap of the tablespace with extent maps of the segments in that tablespace to make sure everything is consistent.
TABLESPACE_EXTENT_MAKE_FREE	Makes this range (extent) of space free in the bitmaps.
TABLESPACE_EXTENT_MAKE_USED	Makes this range (extent) of space used in the bitmaps.

Summary of Subprograms

Table 47–1 DBMS_SPACE_ADMIN Package Subprograms

Subprogram	Description
SEGMENT_VERIFY procedure on page 47-3	Verifies the consistency of the extent map of the segment.
SEGMENT_CORRUPT procedure on page 47-4	Marks the segment corrupt or valid so that appropriate error recovery can be done.
SEGMENT_DROP_CORRUPT procedure on page 47-5	Drops a segment currently marked corrupt (without reclaiming space).
SEGMENT_DUMP procedure on page 47-6	Dumps the segment header and extent map(s) of a given segment.
TABLESPACE_VERIFY procedure on page 47-7	Verifies that the bitmaps and extent maps for the segments in the tablespace are in sync.
TABLESPACE_FIX_BITMAPS procedure on page 47-7	Marks the appropriate DBA range (extent) as free or used in bitmap.
TABLESPACE_REBUILD_BITMAPS procedure on page 47-8	Rebuilds the appropriate bitmap(s).
TABLESPACE_REBUILD_QUOTAS procedure on page 47-9	Rebuilds quotas for given tablespace.
TABLESPACE_MIGRATE_FROM_LOCAL procedure on page 47-10	Migrates a locally-managed tablespace to dictionary-managed tablespace.

SEGMENT_VERIFY procedure

This procedure verifies that the extent map of the segment is consistent with the bitmap.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_VERIFY (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE,
   verify_option           IN    POSITIVE  DEFAULT SEGMENT_VERIFY_EXTENTS);
```

Parameters

Table 47–2 SEGMENT_VERIFY Procedure Parameters

Parameters	Description
tablespace_name	Name of tablespace in which segment resides.
header_relative_file	Relative file number of segment header.
header_block	Block number of segment header.
verify_option	What kind of check to do: SEGMENT_VERIFY_EXTENTS or SEGMENT_VERIFY_EXTENTS_GLOBAL.

Usage Notes

Anomalies are output as dba-range, bitmap-block, bitmap-block-range, anomaly-information, in the trace file for all dba-ranges found to have incorrect space representation. The kinds of problems which would be reported are free space not considered free, used space considered free, and the same space considered used by multiple segments.

Example

The following example verifies that the segment with segment header at relative file number 4, block number 33, has its extent maps and bitmaps in sync.

```
SQLPLUS > EXECUTE DBMS_SPACE_ADMIN.SEGMENT_VERIFY('USERS', 4, 33, 1);
```

Note: All DBMS_SPACE_ADMIN package examples use the tablespace USERS which contains SCOTT.EMP.

SEGMENT_CORRUPT procedure

This procedure marks the segment corrupt or valid so that appropriate error recovery can be done.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_CORRUPT (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE,
   corrupt_option          IN    POSITIVE  DEFAULT SEGMENT_MARK_CORRUPT);
```

Parameters

Table 47–3 SEGMENT_CORRUPT Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace in which segment resides.
header_relative_file	Relative file number of segment header.
header_block	Block number of segment header.
corrupt_option	SEGMENT_MARK_CORRUPT (default) or SEGMENT_MARK_VALID.

Example

The following example marks the segment as corrupt:

```
SQLPLUS > EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33, 3);
```

Alternately, the next example marks a corrupt segment valid:

```
SQLPLUS > EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('USERS', 4, 33, 4);
```

SEGMENT_DROP_CORRUPT procedure

This procedure drops a segment currently marked corrupt (without reclaiming space). For this to work, the segment should have been marked *temporary*. To mark a corrupt segment as temporary, issue a DROP command on the segment.

The space for the segment is not released, and it must be fixed by using the TABLESPACE_FIX_BITMAPS or TABLESPACE_REBUILD_BITMAPS procedure. These are described later in this chapter.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE);
```

Parameters

Table 47–4 SEGMENT_DROP_CORRUPT Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace in which segment resides.
header_relative_file	Relative file number of segment header.
header_block	Block number of segment header.

Example

```
SQLPLUS > EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DROP_CORRUPT('USERS', 4, 33);
```

SEGMENT_DUMP procedure

This procedure dumps the segment header and extent map blocks of the given segment.

Syntax

```
DBMS_SPACE_ADMIN.SEGMENT_DUMP (
   tablespace_name         IN    VARCHAR2,
   header_relative_file    IN    POSITIVE,
   header_block            IN    POSITIVE,
   dump_option             IN    POSITIVE  DEFAULT SEGMENT_DUMP_EXTENT_MAP);
```

Parameters

Table 47–5 SEGMENT_DUMP Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace in which segment resides.
header_relative_file	Relative file number of segment header.
header_block	Block number of segment header.
dump_option	SEGMENT_DUMP_EXTENT_MAP

Example

```
SQLPLUS > EXECUTE DBMS_SPACE_ADMIN.SEGMENT_DUMP('USERS', 4, 33);
```

TABLESPACE_VERIFY procedure

This procedure verifies that the bitmaps and extent maps for the segments in the tablespace are in sync.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_VERIFY (
   tablespace_name          IN    VARCHAR2,
   verify_option            IN    POSITIVE DEFAULT TABLESPACE_VERIFY_BITMAP);
```

Parameters

Table 47–6 TABLESPACE_VERIFY Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.
verify_option	TABLESPACE_VERIFY_BITMAP

Example

```
SQLPLUS > EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_VERIFY('USERS');
```

TABLESPACE_FIX_BITMAPS procedure

This procedure marks the appropriate DBA range (extent) as free or used in bitmap.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS (
   tablespace_name          IN    VARCHAR2,
   dbarange_relative_file   IN    POSITIVE,
   dbarange_begin_block     IN    POSITIVE,
   dbarange_end_block       IN    POSITIVE,
   fix_option               IN    POSITIVE);
```

Parameters

Table 47–7 TABLESPACE_FIX_BITMAPS Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.
dbarange_relative_file	Relative file number of DBA range (extent).
dbarange_begin_block	Block number of beginning of extent.
dbarange_end_block	Block number (inclusive) of end of extent.
fix_option	TABLESPACE_EXTENT_MAKE_FREE or TABLESPACE_EXTENT_MAKE_USED.

Example

The following example marks bits for 50 blocks for relative file number 4, beginning at block number 33 and ending at 83, as USED in bitmaps.

```
SQLPLUS > EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS('USERS', 4, 33, 83,
7);
```

Alternately, specifying an option of 8 marks the bits FREE in bitmaps. The BEGIN and END blocks should be in extent boundary and should be extent multiple. Otherwise, an error is raised.

TABLESPACE_REBUILD_BITMAPS procedure

This procedure rebuilds the appropriate bitmap(s). If no bitmap block DBA is specified, then it rebuilds all bitmaps for the given tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS (
   tablespace_name         IN    VARCHAR2,
   bitmap_relative_file    IN    POSITIVE   DEFAULT NULL,
   bitmap_block            IN    POSITIVE   DEFAULT NULL);
```

Parameters

Table 47–8 TABLESPACE_REBUILD_BITMAPS Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.
bitmap_relative_file	Relative file number of bitmap block to rebuild.
bitmap_block	Block number of bitmap block to rebuild.

Example

The following example rebuilds bitmaps for all the files in the USERS tablespace.

```
SQLPLUS > EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS('USERS');
```

Note: Currently, only full rebuild is supported.

TABLESPACE_REBUILD_QUOTAS procedure

This procedure rebuilds quotas for the given tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS (
   tablespace_name          IN     VARCHAR2);
```

Parameters

Table 47–9 TABLESPACE_REBUILD_QUOTAS Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.

Example

```
SQLPLUS > EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS('USERS');
```

TABLESPACE_MIGRATE_FROM_LOCAL procedure

This procedure migrates a locally-managed tablespace to a dictionary-managed tablespace.

Syntax

```
DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL (
   tablespace_name         IN    VARCHAR2);
```

Parameters

Table 47–10 TABLESPACE_MIGRATE_FROM_LOCAL Procedure Parameters

Parameter	Description
tablespace_name	Name of tablespace.

Usage Notes

The tablespace must be kept online and read write during migration.

Migration of temporary tablespaces and migration of SYSTEM tablespaces are not supported.

> **Note:** Migration of SYSTEM tablespaces will be supported in the future.

Example

```
SQLPLUS > EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL('USERS');
```

Examples

1. TABLESPACE_VERIFY has discovered that some segment has allocated blocks which are marked free in the bitmap, but no overlap between segments was reported. In this case, the following actions are recommended:
   - Call SEGMENT_EXTENT_MAP_DUMP procedure to dump the DBA ranges allocated to the segment.
   - For each range, call TABLESPACE_FIX_BITMAPS with the TABLESPACE_MAKE_USED option to mark the space as used.
   - Call TABLESPACE_REBUILD_QUOTAS to fix up quotas.

2. Segment could not be dropped because the bitmap has some of the segment blocks marked free. The system has automatically marked it corrupted. In this case, do the following:
   - Call SEGMENT_VERIFY procedure with the SEGMENT_CHECK_ALL options. If no overlaps were reported, then do the following:
     * Call SEGMENT_EXTENT_MAP_DUMP procedure to dump the DBA ranges allocated to the segment.
     * For each range, call TABLESPACE_FIX_BITMAPS with the TABLESPACE_MAKE_FREE option to mark the space as free.
     * Call SEGMENT_DROP_CORRUPT to drop the SEG$ entry.
     * Call TABLESPACE_REBUILD_QUOTAS to fix up quotas
3. TABLESPACE_VERIFY has reported some overlaps. In this case, some of the real data will need to be sacrificed, perhaps, based on the previous internal errors. After the victim object is chosen, table T1, do the following:
   - Make a list of all objects which T1 overlaps.
   - Drop T1 using SQL. If necessary, then follow up by SEGMENT_DROP_CORRUPT.
   - Call SEGMENT_VERIFY on all objects that T1 has overlapped. If necessary, then call TABLESPACE_FIX_BITMAPS to mark appropriate bitmap blocks as used.
   - Rerun TABLESPACE_VERIFY.
4. A set of bitmap blocks is media corrupt. Do the following:
   - Call TABLESPACE_REBUILD_BITMAPS, either on all bitmap blocks or on a single block, if only one block is corrupt.
   - Call TABLESPACE_REBUILD_QUOTAS to rebuild quotas.
   - Call TABLESPACE_VERIFY to check if the bitmaps are consistent.

48

DBMS_SQL

Oracle lets you to write stored procedures and anonymous PL/SQL blocks that use dynamic SQL. Dynamic SQL statements are not embedded in your source program; rather, they are stored in character strings that are input to, or built by, the program at runtime. This enables you to create more general-purpose procedures. For example, dynamic SQL lets you create a procedure that operates on a table whose name is not known until runtime.

Additionally, DBMS_SQL enables you to parse any data manipulation language (DML) or data definition language (DDL) statement. Therefore, you can parse DDL statements directly using PL/SQL. For example, you might now choose to enter a DROP TABLE statement from within a stored procedure by using the PARSE procedure supplied with the DBMS_SQL package.

Note: Oracle8*i* introduces native dynamic SQL, an alternative to DBMS_SQL. Using native dynamic SQL, you can place dynamic SQL statements directly into PL/SQL blocks.

In most situations, native dynamic SQL can replace DBMS_SQL. Native dynamic SQL is easier to use and performs better than DBMS_SQL.

See Also: For more information on native dynamic SQL, see *PL/SQL User's Guide and Reference.*

For a comparison of DBMS_SQL and native dynamic SQL, see *Oracle8i Application Developer's Guide - Fundamentals.*

Using DBMS_SQL

The ability to use dynamic SQL from within stored procedures generally follows the model of the Oracle Call Interface (OCI).

> **See Also:** *Oracle Call Interface Programmer's Guide*

PL/SQL differs somewhat from other common programming languages, such as C. For example, addresses (also called pointers) are not user-visible in PL/SQL. As a result, there are some differences between the Oracle Call Interface and the DBMS_SQL package. These differences include the following:

- The OCI uses bind by address, while the DBMS_SQL package uses bind by value.
- With DBMS_SQL you must call VARIABLE_VALUE to retrieve the value of an OUT parameter for an anonymous block, and you must call COLUMN_VALUE after fetching rows to actually retrieve the values of the columns in the rows into your program.
- The current release of the DBMS_SQL package does not provide CANCEL cursor procedures.
- Indicator variables are not required, because NULLs are fully supported as values of a PL/SQL variable.

A sample usage of the DBMS_SQL package is shown below. For users of the Oracle Call Interfaces, this code should seem fairly straightforward.

Example

This example does not actually require the use of dynamic SQL, because the text of the statement is known at compile time. It does, however, illustrate the concepts of this package.

The DEMO procedure deletes all of the employees from the EMP table whose salaries are greater than the salary that you specify when you run DEMO.

```
CREATE OR REPLACE PROCEDURE demo(salary IN NUMBER) AS
    cursor_name INTEGER;
    rows_processed INTEGER;
BEGIN
    cursor_name := dbms_sql.open_cursor;
    DBMS_SQL.PARSE(cursor_name, 'DELETE FROM emp WHERE sal > :x',
                   dbms_sql.native);
    DBMS_SQL.BIND_VARIABLE(cursor_name, ':x', salary);
```

```
    rows_processed := dbms_sql.execute(cursor_name);
    DBMS_SQL.close_cursor(cursor_name);
EXCEPTION
WHEN OTHERS THEN
    DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;
```

Constants

```
v6 constant INTEGER     := 0;
native constant INTEGER := 1;
v7 constant INTEGER     := 2;
```

Types

```
TYPE varchar2s IS TABLE OF VARCHAR2(256) INDEX BY BINARY_INTEGER;
TYPE desc_rec  IS RECORD (
      col_type            BINARY_INTEGER := 0,
      col_max_len         BINARY_INTEGER := 0,
      col_name            VARCHAR2(32)   := '',
      col_name_len        BINARY_INTEGER := 0,
      col_schema_name     VARCHAR2(32)   := '',
      col_schema_name_len BINARY_INTEGER := 0,
      col_precision       BINARY_INTEGER := 0,
      col_scale           BINARY_INTEGER := 0,
      col_charsetid       BINARY_INTEGER := 0,
      col_charsetform     BINARY_INTEGER := 0,
      col_null_ok         BOOLEAN        := TRUE);
TYPE desc_tab IS TABLE OF desc_rec INDEX BY BINARY_INTEGER;
```

Bulk SQL Types

```
type Number_Table   IS TABLE OF NUMBER         INDEX BY BINARY_INTEGER;
type Varchar2_Table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
type Date_Table     IS TABLE OF DATE           INDEX BY BINARY_INTEGER;
type Blob_Table     IS TABLE OF BLOB           INDEX BY BINARY_INTEGER;
type Clob_Table     IS TABLE OF CLOB           INDEX BY BINARY_INTEGER;
type Bfile_Table    IS TABLE OF BFILE          INDEX BY BINARY_INTEGER;
```

Exceptions

```
inconsistent_type exception;
  pragma exception_init(inconsistent_type, -6562);
```

This exception is raised by procedure COLUMN_VALUE or VARIABLE_VALUE when the type of the given OUT parameter (for where to put the requested value) is different from the type of the value.

Execution Flow

OPEN_CURSOR

To process a SQL statement, you must have an open cursor. When you call the OPEN_CURSOR function, you receive a cursor ID number for the data structure representing a valid cursor maintained by Oracle. These cursors are distinct from cursors defined at the precompiler, OCI, or PL/SQL level, and are used only by the DBMS_SQL package.

PARSE

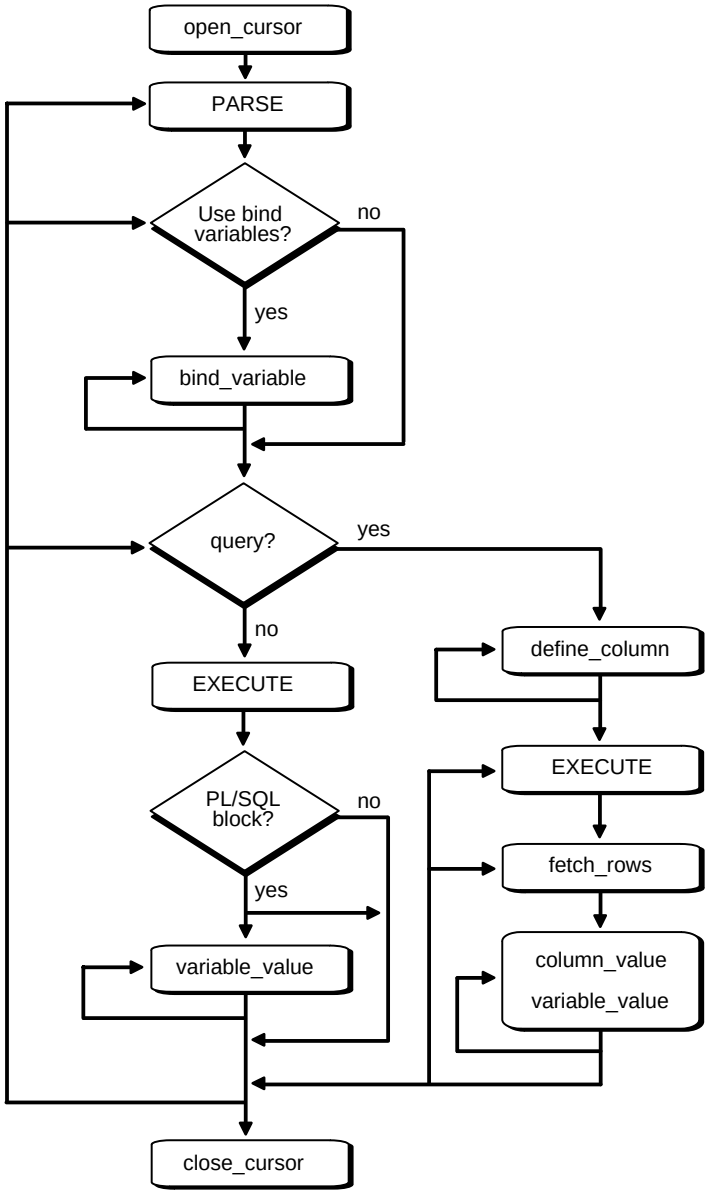
Every SQL statement must be parsed by calling the PARSE procedure. Parsing the statement checks the statement's syntax and associates it with the cursor in your program.

> **See Also:** *Oracle8 Concepts* provides an explanation of how SQL statements are parsed.

You can parse any DML or DDL statement. DDL statements are run on the parse, which performs the implied commit.

Note: When parsing a DDL statement to drop a package or a procedure, a deadlock can occur if you're still using a procedure in the package. After a call to a procedure, that procedure is considered to be in use until execution has returned to the user side. Any such deadlock timeouts after five minutes.

Figure 48–1 DBMS_SQL Execution Flow



BIND_VARIABLE or BIND_ARRAY

Many DML statements require that data in your program be input to Oracle. When you define a SQL statement that contains input data to be supplied at runtime, you must use placeholders in the SQL statement to mark where data must be supplied.

For each placeholder in the SQL statement, you must call one of the bind procedures, BIND_VARIABLE or BIND_ARRAY, to supply the value of a variable in your program (or the values of an array) to the placeholder. When the SQL statement is subsequently run, Oracle uses the data that your program has placed in the output and input, or bind, variables.

DBMS_SQL can run a DML statement multiple times — each time with a different bind variable. The BIND_ARRAY procedure lets you bind a collection of scalars, each value of which is used as an input variable once per EXECUTE. This is similar to the array interface supported by the OCI.

DEFINE_COLUMN, DEFINE_COLUMN_LONG, or DEFINE_ARRAY

The columns of the row being selected in a SELECT statement are identified by their relative positions as they appear in the select list, from left to right. For a query, you must call one of the define procedures (DEFINE_COLUMN, DEFINE_COLUMN_LONG, or DEFINE_ARRAY) to specify the variables that are to receive the SELECT values, much the way an INTO clause does for a static query.

Use the DEFINE_COLUMN_LONG procedure to define LONG columns, in the same way that DEFINE_COLUMN is used to define non-LONG columns. You must call DEFINE_COLUMN_LONG before using the COLUMN_VALUE_LONG procedure to fetch from the LONG column.

Use the DEFINE_ARRAY procedure to define a PL/SQL collection into which you want to fetch rows in a single SELECT statement. DEFINE_ARRAY provides an interface to fetch multiple rows at one fetch. You must call DEFINE_ARRAY before using the COLUMN_VALUE procedure to fetch the rows.

EXECUTE

Call the EXECUTE function to run your SQL statement.

FETCH_ROWS or EXECUTE_AND_FETCH

The FETCH_ROWS function retrieves the rows that satisfy the query. Each successive fetch retrieves another set of rows, until the fetch is unable to retrieve anymore rows. Instead of calling EXECUTE and then FETCH_ROWS, you may find it more

efficient to call EXECUTE_AND_FETCH if you are calling EXECUTE for a single execution.

VARIABLE_VALUE, COLUMN_VALUE, or COLUMN_VALUE_LONG

For queries, call COLUMN_VALUE to determine the value of a column retrieved by the FETCH_ROWS call. For anonymous blocks containing calls to PL/SQL procedures or DML statements with `returning` clause, call VARIABLE_VALUE to retrieve the values assigned to the output variables when statements were run.

To fetch just part of a LONG database column (which can be up to two gigabytes in size), use the COLUMN_VALUE_LONG procedure. You can specify the offset (in bytes) into the column value, and the number of bytes to fetch.

CLOSE_CURSOR

When you no longer need a cursor for a session, close the cursor by calling CLOSE_CURSOR. If you are using an Oracle Open Gateway, then you may need to close cursors at other times as well. Consult your *Oracle Open Gateway* documentation for additional information.

If you neglect to close a cursor, then the memory used by that cursor remains allocated even though it is no longer needed.

Security

Definer Rights Modules

Definer rights modules run under the privileges of the owner of the module. DBMS_SQL subprograms called from definer rights modules run with respect to the schema in which the module is defined.

> **Note:** Prior to Oracle 8*i*, all PL/SQL stored procedures and packages were definer rights modules.

Invoker Rights Modules

Invoker rights modules run under the privileges of the invoker of the module. Therefore, DBMS_SQL subprograms called from invoker rights modules run under the privileges of the invoker of the module.

When a module has AUTHID set to `current_user`, the unqualified names are resolved with respect to the invoker's schema.

Example: income is an invoker rights stored procedure in USER1's schema, and USER2 has been granted EXECUTE privilege on it.

```
CREATE PROCEDURE income(amount number)
   AUTHID current_user IS
   c number;
   n number;
BEGIN
   c:= dbms_sql.open_cursor;
   dbms_sql.parse(c, 'insert into accts(''income'', :1)', dbms_sql.native);
   dbms_sql.bind_variable(c, '1', amount);
   n := dbms_sql.execute(c);
   dbms_sql.close_cursor(c);
END;
```

If USER1 calls USER1.income, then USER1's privileges are used, and name resolution of unqualified names is done with respect to USER1's schema.

If USER2 calls USER1.income, then USER2's privileges are used, and name resolution of unqualified names (such as accts) is done with respect to USER2's schema.

See Also: *PL/SQL User's Guide and Reference*

Anonymous Blocks

Any DBMS_SQL subprograms called from an anonymous PL/SQL block are run using the privileges of the current user.

Summary of Subprograms

Table 48–1 DBMS_SQL Package Subprograms

Subprogram	Description
OPEN_CURSOR function on page 48-10	Returns cursor ID number of new cursor.
PARSE procedure on page 48-10	Parses given statement.
BIND_VARIABLE procedure on page 48-13	Binds a given value to a given variable.
BIND_ARRAY procedure on page 48-13	Binds a given value to a given collection.

Table 48–1 DBMS_SQL Package Subprograms

Subprogram	Description
DEFINE_COLUMN procedure on page 48-18	Defines a column to be selected from the given cursor, used only with SELECT statements.
DEFINE_ARRAY procedure on page 48-19	Defines a collection to be selected from the given cursor, used only with SELECT statements.
DEFINE_COLUMN_LONG procedure on page 48-21	Defines a LONG column to be selected from the given cursor, used only with SELECT statements.
EXECUTE function on page 48-22	Executes a given cursor.
EXECUTE_AND_FETCH function on page 48-22	Executes a given cursor and fetch rows.
FETCH_ROWS function on page 48-23	Fetches a row from a given cursor.
COLUMN_VALUE procedure on page 48-24	Returns value of the cursor element for a given position in a cursor.
COLUMN_VALUE_LONG procedure on page 48-26	Returns a selected part of a LONG column, that has been defined using DEFINE_COLUMN_LONG.
VARIABLE_VALUE procedure on page 48-27	Returns value of named variable for given cursor.
IS_OPEN function on page 48-29	Returns TRUE if given cursor is open.
DESCRIBE_COLUMNS procedure on page 48-30	Describes the columns for a cursor opened and parsed through DBMS_SQL.
CLOSE_CURSOR procedure on page 48-32	Closes given cursor and frees memory.
LAST_ERROR_POSITION function on page 48-33	Returns byte offset in the SQL statement text where the error occurred.
LAST_ROW_COUNT function on page 48-33	Returns cumulative count of the number of rows fetched.
LAST_ROW_ID function on page 48-34	Returns ROWID of last row processed.
LAST_SQL_FUNCTION_CODE function on page 48-34	Returns SQL function code for statement.

OPEN_CURSOR function

This procedure opens a new cursor. When you no longer need this cursor, you must close it explicitly by calling CLOSE_CURSOR.

You can use cursors to run the same SQL statement repeatedly or to run a new SQL statement. When a cursor is reused, the contents of the corresponding cursor data area are reset when the new SQL statement is parsed. It is never necessary to close and reopen a cursor before reusing it.

Syntax

```
DBMS_SQL.OPEN_CURSOR
   RETURN INTEGER;
```

Parameters

None.

Pragmas

```
pragma restrict_references(open_cursor,RNDS,WNDS);
```

Returns

This function returns the cursor ID number of the new cursor.

PARSE procedure

This procedure parses the given statement in the given cursor. All statements are parsed immediately. In addition, DDL statements are run immediately when parsed.

There are two versions of the PARSE procedure: one uses a VARCHAR2 statement as an argument, and the other uses a VARCHAR2S (table of VARCHAR2) as an argument.

Caution: Using DBMS_SQL to dynamically run DDL statements can result in the program hanging. For example, a call to a procedure in a package results in the package being locked until the execution returns to the user side. Any operation that results in a conflicting lock, such as dynamically trying to drop the package before the first lock is released, results in a hang.

The size limit for parsing SQL statements with the syntax above is 32KB.

Syntax

```
DBMS_SQL.PARSE (
   c                IN INTEGER,
   statement        IN VARCHAR2,
   language_flag    IN INTEGER);
```

The PARSE procedure also supports the following syntax for large SQL statements:

> **Note:** The procedure concatenates elements of a PL/SQL table statement and parses the resulting string. You can use this procedure to parse a statement that is longer than the limit for a single VARCHAR2 variable by splitting up the statement.

```
DBMS_SQL.PARSE (
   c                IN INTEGER,
   statement        IN VARCHAR2S,
   lb               IN INTEGER,
   ub               IN INTEGER,
   lfflg            IN BOOLEAN,
   language_flag    IN INTEGER);
```

Parameters

Table 48–2 PARSE Procedure Parameters

Parameter	Description
c	ID number of the cursor in which to parse the statement.

statement	SQL statement to be parsed.
	Unlike PL/SQL statements, your SQL statement should not include a final semicolon. For example:
	`DBMS_SQL.PARSE(cursor1, 'BEGIN proc; END;', 2);`
	`DBMS_SQL.PARSE(cursor1, 'INSERT INTO tab values(1)', 2);`

lb	Lower bound for elements in the statement.
ub	Upper bound for elements in the statement.

Table 48–2 PARSE Procedure Parameters

Parameter	Description
lfflg	If TRUE, then insert a linefeed after each element on concatenation.
language_flag	Determines how Oracle handles the SQL statement. The following options are recognized: ■ V6 (or 0) specifies version 6 behavior. ■ NATIVE (or 1) specifies normal behavior for the database to which the program is connected. ■ V7 (or 2) specifies Oracle7 behavior.

Note: Because client-side code cannot reference remote package variables or constants, you must explicitly use the values of the constants.

For example, the following code does *not* compile on the client:

```
DBMS_SQL.PARSE(cur_hdl, stmt_str, dbms_sql.V7); -- uses constant
dbms_sql.V7
```

The following code works on the client, because the argument is explicitly provided:

```
DBMS_SQL.PARSE(cur_hdl, stmt_str, 2); -- compiles on the client
```

VARCHAR2S Datatype for Parsing Large SQL Strings To parse SQL statements larger than 32 KB, DBMS_SQL makes use of PL/SQL tables to pass a table of strings to the PARSE procedure. These strings are concatenated and then passed on to the Oracle server.

You can declare a local variable as the VARCHAR2S table-item type, and then use the PARSE procedure to parse a large SQL statement as VARCHAR2S.

The definition of the VARCHAR2S datatype is:

```
TYPE varchar2s IS TABLE OF VARCHAR2(256) INDEX BY BINARY_INTEGER;
```

Exceptions

If you create a type/procedure/function/package using DBMS_SQL that has compilation warnings, an ORA-24344 exception is raised, and the procedure is still created.

BIND_VARIABLE procedure

BIND_ARRAY procedure

These two procedures bind a given value or set of values to a given variable in a cursor, based on the name of the variable in the statement. If the variable is an IN or IN/OUT variable or an IN collection, then the given bind value must be valid for the variable or array type. Bind values for OUT variables are ignored.

The bind variables or collections of a SQL statement are identified by their names. When binding a value to a bind variable or bind array, the string identifying it in the statement must contain a leading colon, as shown in the following example:

```
SELECT emp_name FROM emp WHERE SAL > :X;
```

For this example, the corresponding bind call would look similar to

```
BIND_VARIABLE(cursor_name, ':X', 3500);
```

or

```
BIND_VARIABLE (cursor_name, 'X', 3500);
```

Syntax

```
DBMS_SQL.BIND_VARIABLE (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN <datatype>);
```

Where <datatype> can be any one of the following types:

```
NUMBER
DATE
VARCHAR2 CHARACTER SET ANY_CS
BLOB
CLOB CHARACTER SET ANY_CS
BFILE
```

Notice that BIND_VARIABLE is overloaded to accept different datatypes.

> **See Also:** *Oracle8i Application Developer's Guide - Large Objects (LOBs)*

Pragmas

```
pragma restrict_references(bind_variable,WNDS);
```

Usage Notes

The following syntax is also supported for BIND_VARIABLE. The square brackets [] indicate an optional parameter for the BIND_VARIABLE function.

```
DBMS_SQL.BIND_VARIABLE (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN VARCHAR2 CHARACTER SET ANY_CS [,out_value_size IN
INTEGER]);
```

To bind CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.BIND_VARIABLE_CHAR (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN CHAR CHARACTER SET ANY_CS [,out_value_size IN INTEGER]);

DBMS_SQL.BIND_VARIABLE_RAW (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN RAW [,out_value_size IN INTEGER]);

DBMS_SQL.BIND_VARIABLE_ROWID (
   c              IN INTEGER,
   name           IN VARCHAR2,
   value          IN ROWID);
```

Parameters

Table 48–3 BIND_VARIABLE Procedure Parameters

Parameter	Description
c	ID number of the cursor to which you want to bind a value.
name	Name of the variable in the statement.
value	Value that you want to bind to the variable in the cursor. For IN and IN/OUT variables, the value has the same type as the type of the value being passed in for this parameter.
out_value_size	Maximum expected OUT value size, in bytes, for the VARCHAR2, RAW, CHAR OUT or IN/OUT variable. If no size is given, then the length of the current value is used. This parameter must be specified if the value parameter is not initialized.

Bulk Array Binds

Bulk selects, inserts, updates, and deletes can enhance the performance of applications by bundling many calls into one. The DBMS_SQL package lets you work on collections of data using the PL/SQL table type.

Table items are unbounded homogeneous collections. In persistent storage, they are like other relational tables and have no intrinsic ordering. But when a table item is brought into the workspace (either by querying or by navigational access of persistent data), or when it is created as the value of a PL/SQL variable or parameter, its elements are given subscripts that can be used with array-style syntax to get and set the values of elements.

The subscripts of these elements need not be dense, and can be any number including negative numbers. For example, a table item can contain elements at locations -10, 2, and 7 only.

When a table item is moved from transient workspace to persistent storage, the subscripts are not stored; the table item is unordered in persistent storage.

At bind time the table is copied out from the PL/SQL buffers into local DBMS_SQL buffers (the same as for all scalar types) and then the table is manipulated from the local DBMS_SQL buffers. Therefore, if you change the table after the bind call, then that change does not affect the way the execute acts.

Types for Scalar and LOB Collections

You can declare a local variable as one of the following table-item types, which are defined as public types in DBMS_SQL.

```
type Number_Table   IS TABLE OF NUMBER         INDEX BY BINARY_INTEGER;
type Varchar2_Table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
type Date_Table     IS TABLE OF DATE           INDEX BY BINARY_INTEGER;
type Blob_Table     IS TABLE OF BLOB           INDEX BY BINARY_INTEGER;
type Clob_Table     IS TABLE OF CLOB           INDEX BY BINARY_INTEGER;
type Bfile_Table    IS TABLE OF BFILE          INDEX BY BINARY_INTEGER;
```

Syntax

```
DBMS_SQL.BIND_ARRAY (
   c                  IN INTEGER,
   name               IN VARCHAR2,
   <table_variable>   IN <datatype>
 [,index1             IN INTEGER,
   index2             IN INTEGER)] );
```

Where the <table_variable> and its corresponding <datatype> can be any one of the following matching pairs:

```
<num_tab>      Number_Table
<vchr2_tab>    Varchar2_Table
<date_tab>     Date_Table
<blob_tab>     Blob_Table
<clob_tab>     Clob_Table
<bfile_tab>    Bfile_Table
```

Notice that the BIND_ARRAY procedure is overloaded to accept different datatypes.

Parameters

Table 48–4 BIND_ARRAY Procedure Parameters

Parameter	Description
c	ID number of the cursor to which you want to bind a value.
name	Name of the collection in the statement.
table_variable	Local variable that has been declared as <datatype>.
index1	Index for the table element that marks the lower bound of the range.
index2	Index for the table element that marks the upper bound of the range.

Usage Notes

For binding a range, the table must contain the elements that specify the range — tab(index1) and tab(index2) — but the range does not have to be dense. Index1 must be less than or equal to index2. All elements between tab(index1) and tab(index2) are used in the bind.

If you do not specify indexes in the bind call, and two different binds in a statement specify tables that contain a different number of elements, then the number of elements actually used is the minimum number between all tables. This is also the case if you specify indexes — the minimum range is selected between the two indexes for all tables.

Not all bind variables in a query have to be array binds. Some can be regular binds and the same value are used for each element of the collections in expression evaluations (and so forth).

> **See Also:** "Examples 3, 4, and 5: Bulk DML" on page 48-38 for examples of how to bind collections.

Processing Queries

If you are using dynamic SQL to process a query, then you must perform the following steps:

1. Specify the variables that are to receive the values returned by the `SELECT` statement by calling `DEFINE_COLUMN`, `DEFINE_COLUMN_LONG`, or `DEFINE_ARRAY`.
2. Run your `SELECT` statement by calling `EXECUTE`.
3. Call `FETCH_ROWS` (or `EXECUTE_AND_FETCH`) to retrieve the rows that satisfied your query.
4. Call `COLUMN_VALUE` or `COLUMN_VALUE_LONG` to determine the value of a column retrieved by the `FETCH_ROWS` call for your query. If you used anonymous blocks containing calls to PL/SQL procedures, then you must call `VARIABLE_VALUE` to retrieve the values assigned to the output variables of these procedures.

DEFINE_COLUMN procedure

This procedure defines a column to be selected from the given cursor. This procedure is only used with SELECT cursors.

The column being defined is identified by its relative position in the SELECT list of the statement in the given cursor. The type of the COLUMN value determines the type of the column being defined.

Syntax

```
DBMS_SQL.DEFINE_COLUMN (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN <datatype>);
```

Where <datatype> can be any one of the following types:

```
NUMBER
DATE
BLOB
CLOB CHARACTER SET ANY_CS
BFILE
```

Notice that DEFINE_COLUMN is overloaded to accept different datatypes.

> **See Also:** *Oracle8i Application Developer's Guide - Large Objects (LOBs)*

Pragmas

```
pragma restrict_references(define_column,RNDS,WNDS);
```

The following syntax is also supported for the DEFINE_COLUMN procedure:

```
DBMS_SQL.DEFINE_COLUMN (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN VARCHAR2 CHARACTER SET ANY_CS,
   column_size    IN INTEGER);
```

To define columns with CHAR, RAW, and ROWID data, you can use the following variations on the procedure syntax:

```
DBMS_SQL.DEFINE_COLUMN_CHAR (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN CHAR CHARACTER SET ANY_CS,
   column_size    IN INTEGER);

DBMS_SQL.DEFINE_COLUMN_RAW (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN RAW,
   column_size    IN INTEGER);

DBMS_SQL.DEFINE_COLUMN_ROWID (
   c              IN INTEGER,
   position       IN INTEGER,
   column         IN ROWID);
```

Parameters

Table 48–5 DEFINE_COLUMN Procedure Parameters

Parameter	Description
c	ID number of the cursor for the row being defined to be selected.
position	Relative position of the column in the row being defined. The first column in a statement has position 1.
column	Value of the column being defined. The type of this value determines the type for the column being defined.
column_size	Maximum expected size of the column value, in bytes, for columns of type VARCHAR2, CHAR, and RAW.

DEFINE_ARRAY procedure

This procedure defines the collection for column into which you want to fetch rows (with a FETCH_ROWS call). This procedure lets you do batch fetching of rows from a single SELECT statement. A single fetch call brings over a number of rows into the PL/SQL aggregate object.

When you fetch the rows, they are copied into DBMS_SQL buffers until you run a COLUMN_VALUE call, at which time the rows are copied into the table that was passed as an argument to the COLUMN_VALUE call.

Scalar and LOB Types for Collections

You can declare a local variable as one of the following table-item types, and then fetch any number of rows into it using DBMS_SQL. (These are the same types as you can specify for the BIND_ARRAY procedure.)

```
type Number_Table   IS TABLE OF NUMBER         INDEX BY BINARY_INTEGER;
type Varchar2_Table IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
type Date_Table     IS TABLE OF DATE           INDEX BY BINARY_INTEGER;
type Blob_Table     IS TABLE OF BLOB           INDEX BY BINARY_INTEGER;
type Clob_Table     IS TABLE OF CLOB           INDEX BY BINARY_INTEGER;
type Bfile_Table    IS TABLE OF BFILE          INDEX BY BINARY_INTEGER;
```

Syntax

```
DBMS_SQL.DEFINE_ARRAY (
   c           IN INTEGER,
   position    IN INTEGER,
   bf_tab      IN Bfile_Table,
   cnt         IN INTEGER,
   lower_bound IN INTEGER);
```

Pragmas

```
pragma restrict_references(define_array,RNDS,WNDS);
```

The subsequent FETCH_ROWS call fetch "count" rows. When the COLUMN_VALUE call is made, these rows are placed in positions indx, indx+1, indx+2, and so on. While there are still rows coming, the user keeps issuing FETCH_ROWS/COLUMN_VALUE calls. The rows keep accumulating in the table specified as an argument in the COLUMN_VALUE call.

Parameters

Table 48–6 DEFINE_ARRAY Procedure Parameters

Parameter	Description
c	ID number of the cursor to which you want to bind an array.
position	Relative position of the column in the array being defined. The first column in a statement has position 1.

Table 48–6 DEFINE_ARRAY Procedure Parameters

Parameter	Description
column	Type of the value being passed in for this parameter is the type of the column to be defined.
column_size	Maximum expected size of the value in bytes for the VARCHAR2 column.

The count has to be an integer greater than zero, otherwise an exception is raised. The indx can be positive, negative, or zero. A query on which a DEFINE_ARRAY call was issued cannot contain array binds.

> **See Also:** "Examples 6 and 7: Defining an Array" on page 48-40 for examples of how to define collections.

DEFINE_COLUMN_LONG procedure

This procedure defines a LONG column for a SELECT cursor. The column being defined is identified by its relative position in the SELECT list of the statement for the given cursor. The type of the COLUMN value determines the type of the column being defined.

Syntax

```
DBMS_SQL.DEFINE_COLUMN_LONG (
   c             IN INTEGER,
   position      IN INTEGER);
```

Parameters

Table 48–7 DEFINE_COLUMN_LONG Procedure Parameters

Parameter	Description
c	ID number of the cursor for the row being defined to be selected.
position	Relative position of the column in the row being defined. The first column in a statement has position 1.

EXECUTE function

This function executes a given cursor. This function accepts the ID number of the cursor and returns the number of rows processed. The return value is only valid for INSERT, UPDATE, and DELETE statements; for other types of statements, including DDL, the return value is undefined and should be ignored.

Syntax

```
DBMS_SQL.EXECUTE (
   c    IN INTEGER)
  RETURN INTEGER;
```

Parameters

Table 48–8 EXECUTE Function Parameters

Parameter	Description
c	Cursor ID number of the cursor to execute.

EXECUTE_AND_FETCH function

This function executes the given cursor and fetch rows. This function provides the same functionality as calling EXECUTE and then calling FETCH_ROWS. Calling EXECUTE_AND_FETCH instead, however, may cut down on the number of network round-trips when used against a remote database.

The EXECUTE_AND_FETCH function returns the number of rows actually fetched.

Syntax

```
DBMS_SQL.EXECUTE_AND_FETCH (
   c              IN INTEGER,
   exact          IN BOOLEAN DEFAULT FALSE)
  RETURN INTEGER;
```

Pragmas

```
pragma restrict_references(execute_and_fetch,WNDS);
```

Parameters

Table 48–9 EXECUTE_AND_FETCH Function Parameters

Parameter	Description
c	ID number of the cursor to execute and fetch.
exact	Set to TRUE to raise an exception if the number of rows actually matching the query differs from one. Even if an exception is raised, the rows are still fetched and available.

FETCH_ROWS function

This function fetches a row from a given cursor. You can call FETCH_ROWS repeatedly as long as there are rows remaining to be fetched. These rows are retrieved into a buffer, and must be read by calling COLUMN_VALUE, for each column, after each call to FETCH_ROWS.

The FETCH_ROWS function accepts the ID number of the cursor to fetch, and returns the number of rows actually fetched.

Syntax

```
DBMS_SQL.FETCH_ROWS (
   c              IN INTEGER)
  RETURN INTEGER;
```

Parameters

Table 48–10 FETCH_ROWS Function Parameters

Parameter	Description
c	ID number.

Pragmas

```
pragma restrict_references(fetch_rows,WNDS);
```

COLUMN_VALUE procedure

This procedure returns the value of the cursor element for a given position in a given cursor. This procedure is used to access the data fetched by calling FETCH_ROWS.

Syntax

```
DBMS_SQL.COLUMN_VALUE (
   c                IN  INTEGER,
   position         IN  INTEGER,
   value            OUT <datatype>
 [,column_error     OUT NUMBER]
 [,actual_length    OUT INTEGER]);
```

Where <datatype> can be any one of the following types:

```
NUMBER
DATE
VARCHAR2 CHARACTER SET ANY_CS
BLOB
CLOB CHARACTER SET ANY_CS
BFILE
```

Note: The square brackets [ ] indicate optional parameters.

See Also: *Oracle8i Application Developer's Guide - Large Objects (LOBs)*

Pragmas

```
pragma restrict_references(column_value,RNDS,WNDS);
```

The following syntax is also supported for the COLUMN_VALUE procedure:

```
DBMS_SQL.COLUMN_VALUE(
   c                IN  INTEGER,
   position         IN  INTEGER,
   <table_variable> IN  <datatype>);
```

Where the <table_variable> and its corresponding <datatype> can be any one of these matching pairs:

```
<num_tab>      Number_Table
<vchr2_tab>    Varchar2_Table
<date_tab>     Date_Table
<blob_tab>     Blob_Table
<clob_tab>     Clob_Table
<bfile_tab>    Bfile_Table
```

For columns containing CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.COLUMN_VALUE_CHAR (
   c               IN  INTEGER,
   position        IN  INTEGER,
   value           OUT CHAR CHARACTER SET ANY_CS
 [,column_error    OUT NUMBER]
 [,actual_length   OUT INTEGER]);

DBMS_SQL.COLUMN_VALUE_RAW (
   c               IN  INTEGER,
   position        IN  INTEGER,
   value           OUT RAW
 [,column_error    OUT NUMBER]
 [,actual_length   OUT INTEGER]);

DBMS_SQL.COLUMN_VALUE_ROWID (
   c               IN  INTEGER,
   position        IN  INTEGER,
   value           OUT ROWID
 [,column_error    OUT NUMBER]
 [,actual_length   OUT INTEGER]);
```

Parameters

Table 48–11 COLUMN_VALUE Procedure Parameters

Parameter	Description
c	ID number of the cursor from which you are fetching the values.
position	Relative position of the column in the cursor. The first column in a statement has position 1.
value	Returns the value at the specified column and row. If the row number specified is greater than the total number of rows fetched, then you receive an error message. Oracle raises exception ORA-06562, inconsistent_type, if the type of this output parameter differs from the actual type of the value, as defined by the call to DEFINE_COLUMN.
table_variable	Local variable that has been declared <datatype>.
column_error	Returns any error code for the specified column value.
actual_length	The actual length, before any truncation, of the value in the specified column.

Exceptions:

inconsistent_type (ORA-06562) is raised if the type of the given OUT parameter value is different from the actual type of the value. This type was the given type when the column was defined by calling procedure DEFINE_COLUMN.

COLUMN_VALUE_LONG procedure

This procedure gets part of the value of a long column.

Syntax

```
DBMS_SQL.COLUMN_VALUE_LONG (
   c            IN  INTEGER,
   position     IN  INTEGER,
   length       IN  INTEGER,
   offset       IN  INTEGER,
   value        OUT VARCHAR2,
   value_length OUT INTEGER);
```

Pragmas

```
pragma restrict_references(column_value_long,RNDS,WNDS);
```

Parameters

Table 48–12 COLUMN_VALUE_LONG Procedure Parameters

Parameter	Description
c	Cursor ID number of the cursor from which to get the value.
position	Position of the column of which to get the value.
length	Number of bytes of the long value to fetch.
offset	Offset into the long field for start of fetch.
value	Value of the column as a VARCHAR2.
value_length	Number of bytes actually returned in value.

VARIABLE_VALUE procedure

This procedure returns the value of the named variable for a given cursor. It is used to return the values of bind variables inside PL/SQL blocks or DML statements with `returning` clause.

Syntax

```
DBMS_SQL.VARIABLE_VALUE (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT <datatype>);
```

Where <datatype> can be any one of the following types:

```
NUMBER
DATE
VARCHAR2 CHARACTER SET ANY_CS
BLOB
CLOB CHARACTER SET ANY_CS
BFILE
```

Pragmas

```
pragma restrict_references(variable_value,RNDS,WNDS);
```

The following syntax is also supported for the VARIABLE_VALUE procedure:

```
DBMS_SQL.VARIABLE_VALUE (
   c                IN  INTEGER,
   name             IN  VARCHAR2,
   <table_variable>  IN  <datatype>);
```

Where the <table_variable> and its corresponding <datatype> can be any one of these matching pairs:

```
<num_tab>      Number_Table
<vchr2_tab>    Varchar2_Table
<date_tab>     Date_Table
<blob_tab>     Blob_Table
<clob_tab>     Clob_Table
<bfile_tab>    Bfile_Table
```

For variables containing CHAR, RAW, and ROWID data, you can use the following variations on the syntax:

```
DBMS_SQL.VARIABLE_VALUE_CHAR (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT CHAR CHARACTER SET ANY_CS);

DBMS_SQL.VARIABLE_VALUE_RAW (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT RAW);

DBMS_SQL.VARIABLE_VALUE_ROWID (
   c               IN  INTEGER,
   name            IN  VARCHAR2,
   value           OUT ROWID);
```

Parameters

Table 48–13 VARIABLE_VALUE Procedure Parameters

Parameter	Description
c	ID number of the cursor from which to get the values.
name	Name of the variable for which you are retrieving the value.
value	Returns the value of the variable for the specified position. Oracle raises exception ORA-06562, inconsistent_type, if the type of this output parameter differs from the actual type of the value, as defined by the call to BIND_VARIABLE.
position	Relative position of the column in the cursor. The first column in a statement has position 1.

Processing Updates, Inserts and Deletes

If you are using dynamic SQL to process an INSERT, UPDATE, or DELETE, then you must perform the following steps:

1. You must first run your INSERT, UPDATE, or DELETE statement by calling EXECUTE.
2. If statements have the returning clause, then you must call VARIABLE_VALUE to retrieve the values assigned to the output variables.

IS_OPEN function

This function checks to see if the given cursor is currently open.

Syntax

```
DBMS_SQL.IS_OPEN (
   c              IN INTEGER)
  RETURN BOOLEAN;
```

Pragmas

```
pragma restrict_references(is_open,RNDS,WNDS);
```

Parameters

Table 48–14 IS_OPEN Function Parameters

Parameter	Description
c	Cursor ID number of the cursor to check.

Returns

Table 48–15 IS_OPEN Function Return Values

Return Value	Description
TRUE	Given cursor is currently open.
FALSE	Given cursor is currently not open.

DESCRIBE_COLUMNS procedure

This procedure describes the columns for a cursor opened and parsed through DBMS_SQL.

The DESC_REC Type

The DBMS_SQL package declares the DESC_REC record type as follows:

```
type desc_rec is record (
   col_type            BINARY_INTEGER := 0,
   col_max_len         BINARY_INTEGER := 0,
   col_name            VARCHAR2(32)   := '',
   col_name_len        BINARY_INTEGER := 0,
   col_schema_name     VARCHAR2(32)   := '',
   col_schema_name_len BINARY_INTEGER := 0,
   col_precision       BINARY_INTEGER := 0,
   col_scale           BINARY_INTEGER := 0,
   col_charsetid       BINARY_INTEGER := 0,
   col_charsetform     BINARY_INTEGER := 0,
   col_null_ok         BOOLEAN        := TRUE);
```

Parameters

Table 48–16 DESC_REC Type Parameters

Parameter	Description
col_type	Type of the column being described.
col_max_len	Maximum length of the column.
col_name	Name of the column.
col_name_len	Length of the column name.
col_schema_name	Name of the schema the column type was defined in, if an object type.
col_schema_name_len	Length of the schema.
col_precision	Column precision, if a number.
col_scale	Column scale, if a number.
col_charsetid	Column character set identifier.
col_charsetform	Column character set form.
col_null_ok	True if column can be null.

The DESC_TAB Type

The DESC_TAB type is a PL/SQL table of DESC_REC records:

```
type desc_tab is table of desc_rec index by BINARY_INTEGER;
```

You can declare a local variable as the PL/SQL table type DESC_TAB, and then call the DESCRIBE_COLUMNS procedure to fill in the table with the description of each column. All columns are described; you cannot describe a single column.

Syntax

```
DBMS_SQL.DESCRIBE_COLUMNS (
   c             IN  INTEGER,
   col_cnt       OUT INTEGER,
   desc_t        OUT DESC_TAB);
```

Parameters

Table 48–17 DBMS_SQL.DESCRIBE_COLUMNS Procedure Parameters

Parameter	Description
c	ID number of the cursor for the columns being described.
col_cnt	Number of columns in the select list of the query.
desc_t	Table of DESC_REC, each DESC_REC describing a column in the query.

See Also: "Example 8: Describe Columns" on page 48-42 illustrates how to use DESCRIBE_COLUMNS.

CLOSE_CURSOR procedure

This procedure closes a given cursor.

Syntax

```
DBMS_SQL.CLOSE_CURSOR (
   c    IN OUT INTEGER);
```

Pragmas

```
pragma restrict_references(close_cursor,RNDS,WNDS);
```

Parameters

Table 48–18 CLOSE_CURSOR Procedure Parameters

Parameter	Mode	Description
c	IN	ID number of the cursor that you want to close.
c	OUT	Cursor is set to null. After you call CLOSE_CURSOR, the memory allocated to the cursor is released and you can no longer fetch from that cursor.

Locating Errors

There are additional functions in the DBMS_SQL package for obtaining information about the last referenced cursor in the session. The values returned by these functions are only meaningful immediately after a SQL statement is run. In addition, some error-locating functions are only meaningful after certain DBMS_SQL calls. For example, you call LAST_ERROR_POSITION immediately after a PARSE.

LAST_ERROR_POSITION function

This function returns the byte offset in the SQL statement text where the error occurred. The first character in the SQL statement is at position 0.

Syntax

```
DBMS_SQL.LAST_ERROR_POSITION
   RETURN INTEGER;
```

Parameters

None.

Pragmas

```
pragma restrict_references(last_error_position,RNDS,WNDS);
```

Usage Notes

Call this function after a PARSE call, before any other DBMS_SQL procedures or functions are called.

LAST_ROW_COUNT function

This function returns the cumulative count of the number of rows fetched.

Syntax

```
DBMS_SQL.LAST_ROW_COUNT
   RETURN INTEGER;
```

Parameters

None.

Pragmas

```
pragma restrict_references(last_row_count,RNDS,WNDS);
```

Usage Notes

Call this function after a FETCH_ROWS or an EXECUTE_AND_FETCH call. If called after an EXECUTE call, then the value returned is zero.

LAST_ROW_ID function

This function returns the ROWID of the last row processed.

Syntax

```
DBMS_SQL.LAST_ROW_ID
   RETURN ROWID;
```

Parameters

None.

Pragmas

```
pragma restrict_references(last_row_id,RNDS,WNDS);
```

Usage Notes

Call this function after a FETCH_ROWS or an EXECUTE_AND_FETCH call.

LAST_SQL_FUNCTION_CODE function

This function returns the SQL function code for the statement. These codes are listed in the *Oracle Call Interface Programmer's Guide*.

Syntax

```
DBMS_SQL.LAST_SQL_FUNCTION_CODE
   RETURN INTEGER;
```

Parameters

None.

Pragmas

```
pragma restrict_references(last_sql_function_code,RNDS,WNDS);
```

Usage Notes

You should call this function immediately after the SQL statement is run; otherwise, the return value is undefined.

Examples

This section provides example procedures that make use of the DBMS_SQL package.

Example 1 The following sample procedure is passed a SQL statement, which it then parses and runs:

```
CREATE OR REPLACE PROCEDURE exec(STRING IN varchar2) AS
    cursor_name INTEGER;
    ret INTEGER;
BEGIN
   cursor_name := DBMS_SQL.OPEN_CURSOR;
```

DDL statements are run by the parse call, which performs the implied commit.

```
   DBMS_SQL.PARSE(cursor_name, string, DBMS_SQL.native);
   ret := DBMS_SQL.EXECUTE(cursor_name);
   DBMS_SQL.CLOSE_CURSOR(cursor_name);
END;
```

Creating such a procedure enables you to perform the following operations:

- The SQL statement can be dynamically generated at runtime by the calling program.
- The SQL statement can be a DDL statement or a DML without binds.

For example, after creating this procedure, you could make the following call:

```
exec('create table acct(c1 integer)');
```

You could even call this procedure remotely, as shown in the following example. This lets you perform remote DDL.

```
exec@hq.com('CREATE TABLE acct(c1 INTEGER)');
```

Example 2 The following sample procedure is passed the names of a source and a destination table, and copies the rows from the source table to the destination table. This sample procedure assumes that both the source and destination tables have the following columns:

```
id        of type NUMBER
name      of type VARCHAR2(30)
birthdate of type DATE
```

This procedure does not specifically require the use of dynamic SQL; however, it illustrates the concepts of this package.

```
CREATE OR REPLACE PROCEDURE copy (
     source      IN VARCHAR2,
     destination IN VARCHAR2) IS
     id_var              NUMBER;
     name_var            VARCHAR2(30);
     birthdate_var       DATE;
     source_cursor       INTEGER;
     destination_cursor INTEGER;
     ignore              INTEGER;
  BEGIN

  -- Prepare a cursor to select from the source table:
     source_cursor := dbms_sql.open_cursor;
     DBMS_SQL.PARSE(source_cursor,
         'SELECT id, name, birthdate FROM ' || source,
          DBMS_SQL.native);
     DBMS_SQL.DEFINE_COLUMN(source_cursor, 1, id_var);
     DBMS_SQL.DEFINE_COLUMN(source_cursor, 2, name_var, 30);
     DBMS_SQL.DEFINE_COLUMN(source_cursor, 3, birthdate_var);
     ignore := DBMS_SQL.EXECUTE(source_cursor);

  -- Prepare a cursor to insert into the destination table:
     destination_cursor := DBMS_SQL.OPEN_CURSOR;
     DBMS_SQL.PARSE(destination_cursor,
                 'INSERT INTO ' || destination ||
                 ' VALUES (:id_bind, :name_bind, :birthdate_bind)',
                  DBMS_SQL.native);

  -- Fetch a row from the source table and insert it into the destination table:
     LOOP
       IF DBMS_SQL.FETCH_ROWS(source_cursor)>0 THEN
         -- get column values of the row
         DBMS_SQL.COLUMN_VALUE(source_cursor, 1, id_var);
```

```
          DBMS_SQL.COLUMN_VALUE(source_cursor, 2, name_var);
          DBMS_SQL.COLUMN_VALUE(source_cursor, 3, birthdate_var);

    -- Bind the row into the cursor that inserts into the destination table. You
    -- could alter this example to require the use of dynamic SQL by inserting an
    -- if condition before the bind.
          DBMS_SQL.BIND_VARIABLE(destination_cursor, ':id_bind', id_var);
          DBMS_SQL.BIND_VARIABLE(destination_cursor, ':name_bind', name_var);
          DBMS_SQL.BIND_VARIABLE(destination_cursor, ':birthdate_bind',
birthdate_var);
          ignore := DBMS_SQL.EXECUTE(destination_cursor);
        ELSE

    -- No more rows to copy:
          EXIT;
        END IF;
      END LOOP;

    -- Commit and close all cursors:
      COMMIT;
      DBMS_SQL.CLOSE_CURSOR(source_cursor);
      DBMS_SQL.CLOSE_CURSOR(destination_cursor);
    EXCEPTION
      WHEN OTHERS THEN
        IF DBMS_SQL.IS_OPEN(source_cursor) THEN
          DBMS_SQL.CLOSE_CURSOR(source_cursor);
        END IF;
        IF DBMS_SQL.IS_OPEN(destination_cursor) THEN
          DBMS_SQL.CLOSE_CURSOR(destination_cursor);
        END IF;
        RAISE;
    END;
/
```

Examples 3, 4, and 5: Bulk DML This series of examples shows how to use bulk array binds (table items) in the SQL DML statements DELETE, INSERT, and UPDATE.

In a DELETE statement, for example, you could bind in an array in the WHERE clause and have the statement be run for each element in the array:

```
declare
  stmt varchar2(200);
  dept_no_array dbms_sql.Number_Table;
  c number;
  dummy number;
begin
  dept_no_array(1) := 10; dept_no_array(2) := 20;
  dept_no_array(3) := 30; dept_no_array(4) := 40;
  dept_no_array(5) := 30; dept_no_array(6) := 40;
  stmt := 'delete from emp where deptno = :dept_array';
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, stmt, dbms_sql.native);
  dbms_sql.bind_array(c, ':dept_array', dept_no_array, 1, 4);
  dummy := dbms_sql.execute(c);
  dbms_sql.close_cursor(c);

  exception when others then
    if dbms_sql.is_open(c) then
      dbms_sql.close_cursor(c);
    end if;
    raise;
end;
/
```

In the example above, only elements 1 through 4 are used as specified by the bind_ array call. Each element of the array potentially deletes a large number of employees from the database.

Here is an example of a bulk INSERT statement:

```
declare
  stmt varchar2(200);
  empno_array dbms_sql.Number_Table;
  empname_array dbms_sql.Varchar2_Table;
  c number;
  dummy number;
begin
  for i in 0..9 loop
    empno_array(i) := 1000 + i;
    empname_array(I) := get_name(i);
```

```
  end loop;
  stmt := 'insert into emp values(:num_array, :name_array)';
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, stmt, dbms_sql.native);
  dbms_sql.bind_array(c, ':num_array', empno_array);
  dbms_sql.bind_array(c, ':name_array', empname_array);
  dummy := dbms_sql.execute(c);
  dbms_sql.close_cursor(c);

  exception when others then
    if dbms_sql.is_open(c) then
      dbms_sql.close_cursor(c);
    end if;
    raise;
end;
/
```

When the execute takes place, all 10 of the employees are inserted into the table.

Finally, here is an example of an bulk UPDATE statement.

```
declare
  stmt varchar2(200);
  emp_no_array dbms_sql.Number_Table;
  emp_addr_array dbms_sql.Varchar2_Table;
  c number;
  dummy number;
begin
  for i in 0..9 loop
    emp_no_array(i) := 1000 + i;
    emp_addr_array(I) := get_new_addr(i);
  end loop;
  stmt := 'update emp set ename = :name_array
    where empno = :num_array';
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, stmt, dbms_sql.native);
  dbms_sql.bind_array(c, ':num_array', empno_array);
  dbms_sql.bind_array(c, ':name_array', empname_array);
  dummy := dbms_sql.execute(c);
  dbms_sql.close_cursor(c);

  exception when others then
    if dbms_sql.is_open(c) then
      dbms_sql.close_cursor(c);
    end if;
    raise;
```

```
end;
/
```

When the EXECUTE call happens, the addresses of all employees are updated at once. The two collections are always stepped in unison. If the WHERE clause returns more than one row, then all those employees get the address the addr_array happens to be pointing to at that time.

Examples 6 and 7: Defining an Array The following examples show how to use the DEFINE_ARRAY procedure:

```
declare
  c       number;
  d       number;
  n_tab   dbms_sql.Number_Table;
  indx    number := -10;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'select n from t order by 1', dbms_sql);

  dbms_sql.define_array(c, 1, n_tab, 10, indx);

  d := dbms_sql.execute(c);
  loop
    d := dbms_sql.fetch_rows(c);

    dbms_sql.column_value(c, 1, n_tab);

    exit when d != 10;
  end loop;

  dbms_sql.close_cursor(c);

  exception when others then
    if dbms_sql.is_open(c) then
      dbms_sql.close_cursor(c);
    end if;
    raise;
end;
/
```

Each time the example above does a FETCH_ROWS call, it fetches 10 rows that are kept in DBMS_SQL buffers. When the COLUMN_VALUE call is run, those rows move into the PL/SQL table specified (in this case n_tab), at positions -10 to -1, as

specified in the DEFINE statements. When the second batch is fetched in the loop, the rows go to positions 0 to 9; and so on.

A current index into each array is maintained automatically. This index is initialized to "indx" at EXECUTE and keeps getting updated every time a COLUMN_VALUE call is made. If you re-execute at any point, then the current index for each DEFINE is re-initialized to "indx".

In this way the entire result of the query is fetched into the table. When FETCH_ROWS cannot fetch 10 rows, it returns the number of rows actually fetched (if no rows could be fetched, then it returns zero) and exits the loop.

Here is another example of using the DEFINE_ARRAY procedure:

Consider a table MULTI_TAB defined as:

```
create table multi_tab (num number,
                        dat1 date,
                        var varchar2(24),
                        dat2 date)
```

To select everything from this table and move it into four PL/SQL tables, you could use the following simple program:

```
declare
  c       number;
  d       number;
  n_tab  dbms_sql.Number_Table;
  d_tab1 dbms_sql.Date_Table;
  v_tab  dbms_sql.Varchar2_Table;
  d_tab2 dbms_sql.Date_Table;
  indx number := 10;
begin

  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'select * from multi_tab order by 1', dbms_sql);

  dbms_sql.define_array(c, 1, n_tab,  5, indx);
  dbms_sql.define_array(c, 2, d_tab1, 5, indx);
  dbms_sql.define_array(c, 3, v_tab,  5, indx);
  dbms_sql.define_array(c, 4, d_tab2, 5, indx);

  d := dbms_sql.execute(c);

  loop
    d := dbms_sql.fetch_rows(c);
```

```
    dbms_sql.column_value(c, 1, n_tab);
    dbms_sql.column_value(c, 2, d_tab1);
    dbms_sql.column_value(c, 3, v_tab);
    dbms_sql.column_value(c, 4, d_tab2);

    exit when d != 5;
  end loop;

  dbms_sql.close_cursor(c);

/*
```

The four tables can be used for anything. One usage might be to use BIND_ARRAY to move the rows to another table by using a query such as 'INSERT into SOME_T values (:a, :b, :c, :d);

```
*/

exception when others then
    if dbms_sql.is_open(c) then
      dbms_sql.close_cursor(c);
    end if;
    raise;
end;
/
```

Example 8: Describe Columns This can be used as a substitute to the SQL*Plus DESCRIBE call by using a SELECT * query on the table that you want to describe.

```
declare
  c number;
  d number;
  col_cnt integer;
  f boolean;
  rec_tab dbms_sql.desc_tab;
  col_num number;
  procedure print_rec(rec in dbms_sql.desc_rec) is
  begin
    dbms_output.new_line;
    dbms_output.put_line('col_type            =    '
                         || rec.col_type);
    dbms_output.put_line('col_maxlen          =    '
                         || rec.col_max_len);
    dbms_output.put_line('col_name            =    '
```

```
                        || rec.col_name);
    dbms_output.put_line('col_name_len        =    '
                        || rec.col_name_len);
    dbms_output.put_line('col_schema_name     =    '
                        || rec.col_schema_name);
    dbms_output.put_line('col_schema_name_len =    '
                        || rec.col_schema_name_len);
    dbms_output.put_line('col_precision       =    '
                        || rec.col_precision);
    dbms_output.put_line('col_scale           =    '
                        || rec.col_scale);
    dbms_output.put('col_null_ok         =    ');
    if (rec.col_null_ok) then
      dbms_output.put_line('true');
    else
      dbms_output.put_line('false');
    end if;
  end;
begin
  c := dbms_sql.open_cursor;

  dbms_sql.parse(c, 'select * from scott.bonus', dbms_sql);

  d := dbms_sql.execute(c);

  dbms_sql.describe_columns(c, col_cnt, rec_tab);

/*
 * Following loop could simply be for j in 1..col_cnt loop.
 * Here we are simply illustrating some of the PL/SQL table
 * features.
 */
  col_num := rec_tab.first;
  if (col_num is not null) then
    loop
      print_rec(rec_tab(col_num));
      col_num := rec_tab.next(col_num);
      exit when (col_num is null);
    end loop;
  end if;

  dbms_sql.close_cursor(c);
end;
/
```

Example 9: RETURNING clause The RETURNING clause was added to DML statements in Oracle 8.0.3. With this clause, INSERT, UPDATE, and DELETE statements can return values of expressions. These values are returned in bind variables.

DBMS_SQL.BIND_VARIABLE is used to bind these outbinds if a single row is inserted, updated, or deleted. If multiple rows are inserted, updated, or deleted, then DBMS_SQL.BIND_ARRAY is used. DBMS_SQL.VARIABLE_VALUE must be called to get the values in these bind variables.

> **Note:** This is similar to DBMS_SQL.VARIABLE_VALUE, which must be called after running a PL/SQL block with an out-bind inside DBMS_SQL.

i) Single row insert

```
create or replace procedure single_Row_insert
     (c1 number, c2 number, r out number) is
c number;
n number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'insert into tab values (:bnd1, :bnd2) ' ||
                    'returning c1*c2 into :bnd3', 2);
dbms_sql.bind_variable(c, 'bnd1', c1);
  dbms_sql.bind_variable(c, 'bnd2', c2);
  dbms_sql.bind_variable(c, 'bnd3', r);
  n := dbms_sql.execute(c);
  dbms_sql.variable_value(c, 'bnd3', r); -- get value of outbind variable
  dbms_Sql.close_Cursor(c);
end;
/
```

ii) Single row update

```
create or replace procedure single_Row_update
     (c1 number, c2 number, r out number) is
c number;
n number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'update tab set c1 = :bnd1, c2 = :bnd2 ' ||
                    'where rownum < 2' ||
                    'returning c1*c2 into :bnd3', 2);
  dbms_sql.bind_variable(c, 'bnd1', c1);
```

```
  dbms_sql.bind_variable(c, 'bnd2', c2);
  dbms_sql.bind_variable(c, 'bnd3', r);
  n := dbms_sql.execute(c);
  dbms_sql.variable_value(c, 'bnd3', r);-- get value of outbind variable
  dbms_Sql.close_Cursor(c);
end;
/
```

iii) Single row delete

```
create or replace procedure single_Row_Delete
     (c1 number, c2 number, r out number) is
c number;
n number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'delete from tab ' ||
                    'where rownum < 2 ' ||
                   'returning c1*c2 into :bnd3', 2);
  dbms_sql.bind_variable(c, 'bnd1', c1);
  dbms_sql.bind_variable(c, 'bnd2', c2);
  dbms_sql.bind_variable(c, 'bnd3', r);
  n := dbms_sql.execute(c);
  dbms_sql.variable_value(c, 'bnd3', r);-- get value of outbind variable
  dbms_Sql.close_Cursor(c);
end;
/
```

iv) Multi-row insert

```
create or replace procedure multi_Row_insert
     (c1 dbms_sql.number_table, c2 dbms_sql.number_table,
      r out dbms_sql.number_table) is
c number;
n number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'insert into tab values (:bnd1, :bnd2) ' ||
                    'returning c1*c2 into :bnd3', 2);
  dbms_sql.bind_array(c, 'bnd1', c1);
  dbms_sql.bind_array(c, 'bnd2', c2);
  dbms_sql.bind_array(c, 'bnd3', r);
  n := dbms_sql.execute(c);
  dbms_sql.variable_value(c, 'bnd3', r);-- get value of outbind variable
  dbms_Sql.close_Cursor(c);
end;
```

```
/
```

v) Multi row Update.

```
create or replace procedure multi_Row_update
     (c1 number, c2 number, r out dbms_Sql.number_table) is
c number;
n number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'update tab set c1 = :bnd1 where c2 = :bnd2 ' ||
                    'returning c1*c2 into :bnd3', 2);
  dbms_sql.bind_variable(c, 'bnd1', c1);
  dbms_sql.bind_variable(c, 'bnd2', c2);
  dbms_sql.bind_array(c, 'bnd3', r);
  n := dbms_sql.execute(c);
  dbms_sql.variable_value(c, 'bnd3', r);-- get value of outbind variable
  dbms_Sql.close_Cursor(c);
end;
/
```

Note: bnd1 and bnd2 can be array as well. The value of the expression for all the rows updated will be in bnd3. There is no way of differentiating which rows got updated of each value of bnd1 and bnd2.

vi) Multi-row delete

```
create or replace procedure multi_row_delete
     (c1 dbms_Sql.number_table,
      r out dbms_sql.number_table) is
c number;
n number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'delete from tab where c1 = :bnd1' ||
                    'returning c1*c2 into :bnd2', 2);
  dbms_sql.bind_array(c, 'bnd1', c1);
  dbms_sql.bind_array(c, 'bnd2', r);
  n := dbms_sql.execute(c);
  dbms_sql.variable_value(c, 'bnd2', r);-- get value of outbind variable
  dbms_Sql.close_Cursor(c);
end;
/
```

vii) Out-bind in bulk PL/SQL

```
create or replace foo (n number, square out number) is
begin square := n * n; end;/

create or replace procedure bulk_plsql
   (n dbms_sql.number_Table, square out dbms_sql.number_table) is
c number;
r number;
begin
  c := dbms_sql.open_cursor;
  dbms_sql.parse(c, 'begin foo(:bnd1, :bnd2); end;', 2);
  dbms_sql.bind_array(c, 'bnd1', n);
  dbms_Sql.bind_Array(c, 'bnd2', square);
  r := dbms_sql.execute(c);
  dbms_Sql.variable_Value(c, 'bnd2', square);
end;
/
```

Note: DBMS_SQL.BIND_ARRAY of number_Table internally binds a number. The number of times statement is run depends on the number of elements in an inbind array.

49

DBMS_STATS

DBMS_STATS provides a mechanism for you to view and modify optimizer statistics gathered for database objects.The statistics can reside in two different locations:

1. The dictionary.
2. A table created in the user's schema for this purpose.

Only statistics stored in the dictionary itself have an impact on the cost-based optimizer.

This package also facilitates the gathering of some statistics in parallel. The package is divided into three main sections:

- Setting or Getting Statistics
- Transferring Statistics
- Gathering Optimizer Statistics

Using DBMS_STATS

Most of the DBMS_STATS procedures include the three parameters statown, stattab, and statid. These parameters allow you to store statistics in your own tables (outside of the dictionary), which does not affect the optimizer. Therefore, you can maintain and experiment with *sets* of statistics.

The stattab parameter specifies the name of a table in which to hold statistics, and it is assumed that it resides in the same schema as the object for which statistics are collected (unless the statown parameter is specified). Users may create multiple tables with different stattab identifiers to hold separate sets of statistics.

Additionally, users can maintain different sets of statistics within a single stattab by using the statid parameter, which can help avoid cluttering the user's schema.

For all of the SET or GET procedures, if stattab is not provided (i.e., NULL), then the operation works directly on the dictionary statistics; therefore, users do not need to create these statistics tables if they only plan to modify the dictionary directly. However, if stattab is not NULL, then the SET or GET operation works on the specified user statistics table, and not the dictionary.

Types

Types for minimum / maximum values and histogram endpoints:

```
TYPE numarray  IS VARRAY(256) OF NUMBER;
TYPE datearray IS VARRAY(256) OF DATE;
TYPE chararray IS VARRAY(256) OF VARCHAR2(4000);
TYPE rawarray  IS VARRAY(256) OF RAW(2000);

type StatRec is record (
  epc    NUMBER,
  minval RAW(2000),
  maxval RAW(2000),
  bkvals NUMARRAY,
  novals NUMARRAY);
```

Types for listing stale tables:

```
type ObjectElem is record (
  ownname     VARCHAR2(30),     -- owner
  objtype     VARCHAR2(6),      -- 'TABLE' or 'INDEX'
  objname     VARCHAR2(30),     -- table/index
  partname    VARCHAR2(30),     -- partition
  subpartname VARCHAR2(30),     -- subpartition
  confidence  NUMBER);          -- not used
type ObjectTab is TABLE of ObjectElem;
```

Summary of Subprograms

Table 49–1 DBMS_STATS Package Subprograms

Subprogram	Description
PREPARE_COLUMN_VALUES procedure on page 49-6	Converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage via SET_COLUMN_STATS.
SET_COLUMN_STATS procedure on page 49-8	Sets column-related information.
SET_INDEX_STATS procedure on page 49-10	Sets index-related information.
SET_TABLE_STATS procedure on page 49-11	Sets table-related information.
CONVERT_RAW_VALUE procedure on page 49-12	Convert the internal representation of a minimum or maximum value into a datatype-specific value.
GET_COLUMN_STATS procedure on page 49-14	Gets all column-related information.
GET_INDEX_STATS procedure on page 49-15	Gets all index-related information.
GET_TABLE_STATS procedure on page 49-17	Gets all table-related information.
DELETE_COLUMN_STATS procedure on page 49-18	Deletes column-related statistics.
DELETE_INDEX_STATS procedure on page 49-19	Deletes index-related statistics.

Table 49–1 DBMS_STATS Package Subprograms

Subprogram	Description
DELETE_TABLE_STATS procedure on page 49-20	Deletes table-related statistics.
DELETE_SCHEMA_STATS procedure on page 49-21	Deletes schema-related statistics.
DELETE_DATABASE_STATS procedure on page 49-22	Deletes statistics for the entire database.
CREATE_STAT_TABLE procedure on page 49-23	Creates a table with name stattab in ownname's schema which is capable of holding statistics.
DROP_STAT_TABLE procedure on page 49-24	Drops a user stat table created by CREATE_STAT_TABLE.
EXPORT_COLUMN_STATS procedure on page 49-25	Retrieves statistics for a particular column and stores them in the user stat table identified by stattab.
EXPORT_INDEX_STATS procedure on page 49-26	Retrieves statistics for a particular index and stores them in the user stat table identified by stattab.
EXPORT_TABLE_STATS procedure on page 49-27	Retrieves statistics for a particular table and stores them in the user stat table.
EXPORT_SCHEMA_STATS procedure on page 49-28	Retrieves statistics for all objects in the schema identified by ownname and stores them in the user stat table identified by stattab.
EXPORT_DATABASE_STATS procedure on page 49-28	Retrieves statistics for all objects in the database and stores them in the user stat table identified by statown.stattab.
IMPORT_COLUMN_STATS procedure on page 49-29	Retrieves statistics for a particular column from the user stat table identified by stattab and stores them in the dictionary.
IMPORT_INDEX_STATS procedure on page 49-30	Retrieves statistics for a particular index from the user stat table identified by stattab and stores them in the dictionary.
IMPORT_TABLE_STATS procedure on page 49-31	Retrieves statistics for a particular table from the user stat table identified by stattab and stores them in the dictionary.
IMPORT_SCHEMA_STATS procedure on page 49-32	Retrieves statistics for all objects in the schema identified by ownname from the user stat table and stores them in the dictionary.

Table 49–1 DBMS_STATS Package Subprograms

Subprogram	Description
IMPORT_DATABASE_STATS procedure on page 49-33	Retrieves statistics for all objects in the database from the user stat table and stores them in the dictionary.
GATHER_INDEX_STATS procedure on page 49-34	Gathers index statistics.
GATHER_TABLE_STATS procedure on page 49-35	Gathers table and column (and index) statistics.
GATHER_SCHEMA_STATS procedure on page 49-37	Gathers statistics for all objects in a schema.
GATHER_DATABASE_STATS procedure on page 49-39	Gathers statistics for all objects in the database.
GENERATE_STATS procedure on page 49-41	Generates object statistics from previously collected statistics of related objects.

Setting or Getting Statistics

The following procedures enable the storage and retrieval of individual column-, index-, and table-related statistics:

```
PREPARE_COLUMN_VALUES
SET_COLUMN_STATS
SET_INDEX_STATS
SET_TABLE_STATS

CONVERT_RAW_VALUE
GET_COLUMN_STATS
GET_INDEX_STATS
GET_TABLE_STATS

DELETE_COLUMN_STATS
DELETE_INDEX_STATS
DELETE_TABLE_STATS
DELETE_SCHEMA_STATS
DELETE_DATABASE_STATS
```

PREPARE_COLUMN_VALUES procedure

This procedure converts user-specified minimum, maximum, and histogram endpoint datatype-specific values into Oracle's internal representation for future storage via SET_COLUMN_STATS.

Syntax

```
DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec     IN OUT StatRec,
   charvals        CHARARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec      IN OUT StatRec,
   datevals         DATEARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec     IN OUT StatRec,
   numvals         NUMARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES (
   srec     IN OUT StatRec,
   rawvals         RAWARRAY);

DBMS_STATS.PREPARE_COLUMN_VALUES_NVARCHAR (
   srec  IN OUT StatRec,
   nvmin        NVARCHAR2,
   nvmax        NVARCHAR2);

DBMS_STATS.PREPARE_COLUMN_VALUES_ROWID (
   srec  IN OUT StatRec,
   rwmin        ROWID,
   rwmax        ROWID);
```

Pragmas

```
pragma restrict_references(prepare_column_values, WNDS, RNDS, WNPS, RNPS);
pragma restrict_references(prepare_column_values_nvarchar, WNDS, RNDS, WNPS,
RNPS);
pragma restrict_references(prepare_column_values_rowid, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 49–2 PREPARE_COLUMN_VALUES Procedure Parameters

Parameter	Description
srec.epc	Number of values specified in charvals, datevals, numvals, or rawvals. This value must be between 2 and 256, inclusive, and it should be set to 2 for procedures which do not allow histogram information (nvarchar and rowid). The first corresponding array entry should hold the minimum value for the column, and the last entry should hold the maximum. If there are more than two entries, then all the others hold the remaining height-balanced or frequency histogram endpoint values (with in-between values ordered from next-smallest to next-largest). This value may be adjusted to account for compression, so the returned value should be left as is for a call to SET_COLUMN_STATS.
srec.bkvals	If you want a frequency distribution, then this array contains the number of occurrences of each distinct value specified in charvals, datevals, numvals, or rawvals. Otherwise, it is merely an output parameter, and it must be set to NULL when this procedure is called.

Datatype specific input parameters (one of the following):

charvals	The array of values when the column type is character-based. Up to the first 32 bytes of each string should be provided. Arrays must have between 2 and 256 entries, inclusive.
datevals	The array of values when the column type is date-based.
numvals	The array of values when the column type is numeric-based.
rawvals	The array of values when the column type is RAW. Up to the first 32 bytes of each strings should be provided.
nvmin, nvmax	The minimum and maximum values when the column type is national character set based (NLS). No histogram information can be provided for a column of this type.
rwmin, rwmax	The minimum and maximum values when the column type is rowid. No histogram information can be provided for a column of this type.

Output parameters

Table 49–3 PREPARE_COLUMN_VALUES Procedure Output Parameters

Parameter	Description
srec.minval	Internal representation of the minimum which is suitable for use in a call to SET_COLUMN_STATS.
srec.maxval	Internal representation of the maximum which is suitable for use in a call to SET_COLUMN_STATS.
srec.bkvals	Array suitable for use in a call to SET_COLUMN_STATS.
srec.novals	Array suitable for use in a call to SET_COLUMN_STATS.

Exceptions

ORA-20001: Invalid or inconsistent input values.

SET_COLUMN_STATS procedure

This procedure sets column-related information.

Syntax

```
DBMS_STATS.SET_COLUMN_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   colname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2 DEFAULT NULL,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL,
   distcnt  NUMBER   DEFAULT NULL,
   density  NUMBER   DEFAULT NULL,
   nullcnt  NUMBER   DEFAULT NULL,
   srec     StatRec  DEFAULT NULL,
   avgclen  NUMBER   DEFAULT NULL,
   flags    NUMBER   DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–4 SET_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column.
partname	Name of the table partition in which to store the statistics. If the table is partitioned and partname is NULL, then the statistics are stored at the global table level.
stattab	User stat table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
distcnt	Number of distinct values.
density	Column density. If this value is NULL and if distcnt is not NULL, then density is derived from distcnt.
nullcnt	Number of NULLs.
srec	StatRec structure filled in by a call to PREPARE_COLUMN_VALUES or GET_COLUMN_STATS.
avgclen	Average length for the column (in bytes).
flags	For internal Oracle use (should be left as NULL).
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent input values.

SET_INDEX_STATS procedure

This procedure sets index-related information.

Syntax

```
DBMS_STATS.SET_INDEX_STATS (
   ownname  VARCHAR2,
   indname  VARCHAR2,
   partname VARCHAR2  DEFAULT NULL,
   stattab  VARCHAR2  DEFAULT NULL,
   statid   VARCHAR2  DEFAULT NULL,
   numrows  NUMBER    DEFAULT NULL,
   numlblks NUMBER    DEFAULT NULL,
   numdist  NUMBER    DEFAULT NULL,
   avglblk  NUMBER    DEFAULT NULL,
   avgdblk  NUMBER    DEFAULT NULL,
   clstfct  NUMBER    DEFAULT NULL,
   indlevel NUMBER    DEFAULT NULL,
   flags    NUMBER    DEFAULT NULL,
   statown  VARCHAR2  DEFAULT NULL);
```

Parameters

Table 49–5 SET_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
indname	Name of the index.
partname	Name of the index partition in which to store the statistics. If the index is partitioned and if partname is NULL, then the statistics are stored at the global index level.
stattab	User stat table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
numrows	Number of rows in the index (partition).
numlblks	Number of leaf blocks in the index (partition).
numdist	Number of distinct keys in the index (partition).

Table 49–5 SET_INDEX_STATS Procedure Parameters

Parameter	Description
avglblk	Average integral number of leaf blocks in which each distinct key appears for this index (partition). If not provided, then this value is derived from numlblks and numdist.
avgdblk	Average integral number of data blocks in the table pointed to by a distinct key for this index (partition). If not provided, then this value is derived from clstfct and numdist.
clstfct	See clustering_factor column of the user_indexes view for a description.
indlevel	Height of the index (partition).
flags	For internal Oracle use (should be left as NULL).
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid input value.

SET_TABLE_STATS procedure

This procedure sets table-related information.

Syntax

```
DBMS_STATS.SET_TABLE_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2 DEFAULT NULL,
   statid   VARCHAR2 DEFAULT NULL,
   numrows  NUMBER   DEFAULT NULL,
   numblks  NUMBER   DEFAULT NULL,
   avgrlen  NUMBER   DEFAULT NULL,
   flags    NUMBER   DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–6 SET_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table.
partname	Name of the table partition in which to store the statistics. If the table is partitioned and partname is NULL, then the statistics are stored at the global table level.
stattab	User stat table identifier describing where to store the statistics. If stattab is NULL, then the statistics are stored directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
numrows	Number of rows in the table (partition).
numblks	Number of blocks the table (partition) occupies.
avgrlen	Average row length for the table (partition).
flags	For internal Oracle use (should be left as NULL).
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid input value.

CONVERT_RAW_VALUE procedure

This procedure converts the internal representation of a minimum or maximum value into a datatype-specific value. The minval and maxval fields of the StatRec structure as filled in by GET_COLUMN_STATS or PREPARE_COLUMN_VALUES are appropriate values for input.

Syntax

```
DBMS_STATS.CONVERT_RAW_VALUE (
   rawval     RAW,
   resval OUT VARCHAR2);

DBMS_STATS.CONVERT_RAW_VALUE (
   rawval     RAW,
   resval OUT DATE);

DBMS_STATS.CONVERT_RAW_VALUE (
   rawval     RAW,
   resval OUT NUMBER);

DBMS_STATS.CONVERT_RAW_VALUE_NVARCHAR (
   rawval     RAW,
   resval OUT NVARCHAR2);

DBMS_STATS.CONVERT_RAW_VALUE_ROWID (
   rawval     RAW,
   resval OUT ROWID);
```

Pragmas

```
pragma restrict_references(convert_raw_value, WNDS, RNDS, WNPS, RNPS);
pragma restrict_references(convert_raw_value_nvarchar, WNDS, RNDS, WNPS, RNPS);
pragma restrict_references(convert_raw_value_rowid, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 49–7 CONVERT_RAW_VALUE Procedure Parameters

Parameter	Description
rawval	The raw representation of a column minimum or maximum datatype-specific output parameters.
resval	The converted, type-specific value.

Exceptions

None.

GET_COLUMN_STATS procedure

This procedure gets all column-related information.

Syntax

```
DBMS_STATS.GET_COLUMN_STATS (
   ownname     VARCHAR2,
   tabname     VARCHAR2,
   colname     VARCHAR2,
   partname    VARCHAR2 DEFAULT NULL,
   stattab     VARCHAR2 DEFAULT NULL,
   statid      VARCHAR2 DEFAULT NULL,
   distcnt OUT NUMBER,
   density OUT NUMBER,
   nullcnt OUT NUMBER,
   srec    OUT StatRec,
   avgclen OUT NUMBER,
   statown     VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–8 GET_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column.
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if partname is NULL, then the statistics are retrieved from the global table level.
stattab	User stat table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
distcnt	Number of distinct values.
density	Column density.
nullcnt	Number of NULLs.

Table 49–8 GET_COLUMN_STATS Procedure Parameters

Parameter	Description
srec	Structure holding internal representation of column minimum, maximum, and histogram values.
avgclen	Average length of the column (in bytes).
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object.

GET_INDEX_STATS procedure

This procedure of gets all index-related information.

Syntax

```
DBMS_STATS.GET_INDEX_STATS (
   ownname      VARCHAR2,
   indname      VARCHAR2,
   partname     VARCHAR2 DEFAULT NULL,
   stattab      VARCHAR2 DEFAULT NULL,
   statid       VARCHAR2 DEFAULT NULL,
   numrows  OUT NUMBER,
   numlblks OUT NUMBER,
   numdist  OUT NUMBER,
   avglblk  OUT NUMBER,
   avgdblk  OUT NUMBER,
   clstfct  OUT NUMBER,
   indlevel OUT NUMBER,
   statown      VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–9 GET_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
indname	Name of the index.
partname	Name of the index partition for which to get the statistics. If the index is partitioned and if partname is NULL, then the statistics are retrieved for the global index level.
stattab	User stat table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
numrows	Number of rows in the index (partition).
numlblks	Number of leaf blocks in the index (partition).
numdist	Number of distinct keys in the index (partition).
avglblk	Average integral number of leaf blocks in which each distinct key appears for this index (partition).
avgdblk	Average integral number of data blocks in the table pointed to by a distinct key for this index (partition).
clstfct	Clustering factor for the index (partition).
indlevel	Height of the index (partition).
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object.

GET_TABLE_STATS procedure

This procedure gets all table-related information.

Syntax

```
DBMS_STATS.GET_TABLE_STATS (
   ownname     VARCHAR2,
   tabname     VARCHAR2,
   partname    VARCHAR2 DEFAULT NULL,
   stattab     VARCHAR2 DEFAULT NULL,
   statid      VARCHAR2 DEFAULT NULL,
   numrows OUT NUMBER,
   numblks OUT NUMBER,
   avgrlen OUT NUMBER,
   statown     VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–10 GET_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if partname is NULL, then the statistics are retrieved from the global table level.
stattab	User stat table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
numrows	Number of rows in the table (partition).
numblks	Number of blocks the table (partition) occupies.
avgrlen	Average row length for the table (partition).
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges or no statistics have been stored for requested object

DELETE_COLUMN_STATS procedure

This procedure deletes column-related statistics.

Syntax

```
DBMS_STATS.DELETE_COLUMN_STATS (
   ownname       VARCHAR2,
   tabname       VARCHAR2,
   colname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   cascade_parts BOOLEAN  DEFAULT TRUE,
   statown       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–11 DELETE_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column.
partname	Name of the table partition for which to delete the statistics. If the table is partitioned and if partname is NULL, then global column statistics are deleted.
stattab	User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
cascade_parts	If the table is partitioned and if partname is NULL, then setting this to true causes the deletion of statistics for this column for all underlying partitions as well.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges

DELETE_INDEX_STATS procedure

This procedure deletes index-related statistics.

Syntax

```
DBMS_STATS.DELETE_INDEX_STATS (
   ownname       VARCHAR2,
   indname       VARCHAR2,
   partname      VARCHAR2 DEFAULT NULL,
   stattab       VARCHAR2 DEFAULT NULL,
   statid        VARCHAR2 DEFAULT NULL,
   cascade_parts BOOLEAN  DEFAULT TRUE,
   statown       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–12 DELETE_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
indname	Name of the index.
partname	Name of the index partition for which to delete the statistics. If the index is partitioned and if partname is NULL, then index statistics are deleted at the global level.
stattab	User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
cascade_parts	If the index is partitioned and if partname is NULL, then setting this to TRUE causes the deletion of statistics for this index for all underlying partitions as well.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

DELETE_TABLE_STATS procedure

This procedure deletes table-related statistics.

Syntax

```
DBMS_STATS.DELETE_TABLE_STATS (
   ownname          VARCHAR2,
   tabname          VARCHAR2,
   partname         VARCHAR2  DEFAULT NULL,
   stattab          VARCHAR2  DEFAULT NULL,
   statid           VARCHAR2  DEFAULT NULL,
   cascade_parts    BOOLEAN   DEFAULT TRUE,
   cascade_columns  BOOLEAN   DEFAULT TRUE,
   cascade_indexes  BOOLEAN   DEFAULT TRUE,
   statown          VARCHAR2  DEFAULT NULL);
```

Parameters

Table 49–13 DELETE_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column.
partname	Name of the table partition from which to get the statistics. If the table is partitioned and if partname is NULL, then the statistics are retrieved from the global table level.
stattab	User stat table identifier describing from where to retrieve the statistics. If stattab is NULL, then the statistics are retrieved directly from the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
cascade_parts	If the table is partitioned and if partname is NULL, then setting this to TRUE causes the deletion of statistics for this table for all underlying partitions as well.
cascade_columns	Indicates that DELETE_COLUMN_STATS should be called for all underlying columns (passing the cascade_parts parameter).
cascade_indexes	Indicates that DELETE_INDEX_STATS should be called for all underlying indexes (passing the cascade_parts parameter).

Table 49–13 DELETE_TABLE_STATS Procedure Parameters

Parameter	Description
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

DELETE_SCHEMA_STATS procedure

This procedure deletes statistics for an entire schema.

Syntax

```
DBMS_STATS.DELETE_SCHEMA_STATS (
   ownname VARCHAR2,
   stattab VARCHAR2 DEFAULT NULL,
   statid  VARCHAR2 DEFAULT NULL,
   statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–14 DELETE_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
stattab	User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges

DELETE_DATABASE_STATS procedure

This procedure deletes statistics for an entire database.

Syntax

```
DBMS_STATS.DELETE_DATABASE_STATS (
   stattab VARCHAR2 DEFAULT NULL,
   statid  VARCHAR2 DEFAULT NULL,
   statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–15 DELETE_DATABASE_STATS Procedure Parameters

Parameter	Description
stattab	User stat table identifier describing from where to delete the statistics. If stattab is NULL, then the statistics are deleted directly in the dictionary.
statid	Identifier (optional) to associate with these statistics within stattab (Only pertinent if stattab is not NULL).
statown	Schema containing stattab. If stattab is not NULL and if statown is NULL, then it is assumed that every schema in the database contains a user statistics table with the name stattab.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

Transferring Statistics

The following procedures enable the transference of statistics from the dictionary to a user stat table (`export_*`) and from a user stat table to the dictionary (`import_*`):

```
CREATE_STAT_TABLE
DROP_STAT_TABLE

EXPORT_COLUMN_STATS
EXPORT_INDEX_STATS
EXPORT_TABLE_STATS
EXPORT_SCHEMA_STATS
EXPORT_DATABASE_STATS

IMPORT_COLUMN_STATS
IMPORT_INDEX_STATS
IMPORT_TABLE_STATS
IMPORT_SCHEMA_STATS
IMPORT_DATABASE_STATS
```

CREATE_STAT_TABLE procedure

This procedure creates a table with name `stattab` in `ownname`'s schema which is capable of holding statistics. The columns and types that compose this table are not relevant as it should be accessed solely through the procedures in this package.

Syntax

```
DBMS_STATS.CREATE_STAT_TABLE (
   ownname  VARCHAR2,
   stattab  VARCHAR2,
   tblspace VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–16 CREATE_STAT_TABLE Procedure Parameters

Parameter	Description
`ownname`	Name of the schema.
`stattab`	Name of the table to create. This value should be passed as the `stattab` parameter to other procedures when the user does not want to modify the dictionary statistics directly.

Table 49–16 CREATE_STAT_TABLE Procedure Parameters

Parameter	Description
tblspace	Tablespace in which to create the stat tables. If none is specified, then they are created in the user's default tablespace.

Exceptions

ORA-20000: Table already exists or insufficient privileges.

ORA-20001: Tablespace does not exist.

DROP_STAT_TABLE procedure

This procedure drops a user stat table.

Syntax

```
DBMS_STATS.DROP_STAT_TABLE (
   ownname VARCHAR2,
   stattab VARCHAR2);
```

Parameters

Table 49–17 DROP_STAT_TABLE Procedure Parameters

Parameter	Description
ownname	Name of the schema.
stattab	User stat table identifier.

Exceptions

ORA-20000: Table does not exists or insufficient privileges.

EXPORT_COLUMN_STATS procedure

This procedure retrieves statistics for a particular column and stores them in the user stat table identified by `stattab`.

Syntax

```
DBMS_STATS.EXPORT_COLUMN_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   colname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–18 EXPORT_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column.
partname	Name of the table partition. If the table is partitioned and if `partname` is `NULL`, then global and partition column statistics are exported.
stattab	User stat table identifier describing where to store the statistics.
statid	Identifier (optional) to associate with these statistics within `stattab`.
statown	Schema containing `stattab` (if different than `ownname`).

Exceptions

`ORA-20000`: Object does not exist or insufficient privileges.

EXPORT_INDEX_STATS procedure

This procedure retrieves statistics for a particular index and stores them in the user stat table identified by stattab.

Syntax

```
DBMS_STATS.EXPORT_INDEX_STATS (
   ownname  VARCHAR2,
   indname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–19 EXPORT_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
indname	Name of the index.
partname	Name of the index partition. If the index is partitioned and if partname is NULL, then global and partition index statistics are exported.
stattab	User stat table identifier describing where to store the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

EXPORT_TABLE_STATS procedure

This procedure retrieves statistics for a particular table and stores them in the user stat table. Cascade results in all index and column stats associated with the specified table being exported as well.

Syntax

```
DBMS_STATS.EXPORT_TABLE_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   cascade  BOOLEAN  DEFAULT TRUE,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–20 EXPORT_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table.
partname	Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition table statistics are exported.
stattab	User stat table identifier describing where to store the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
cascade	If true, then column and index statistics for this table are also exported.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

EXPORT_SCHEMA_STATS procedure

This procedure retrieves statistics for all objects in the schema identified by ownname and stores them in the user stat tables identified by stattab.

Syntax

```
DBMS_STATS.EXPORT_SCHEMA_STATS (
   ownname VARCHAR2,
   stattab VARCHAR2,
   statid  VARCHAR2 DEFAULT NULL,
   statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–21 EXPORT_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
stattab	User stat table identifier describing where to store the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

EXPORT_DATABASE_STATS procedure

This procedure retrieves statistics for all objects in the database and stores them in the user stat tables identified by statown.stattab

Syntax

```
DBMS_STATS.EXPORT_DATABASE_STATS (
   stattab VARCHAR2,
   statid  VARCHAR2 DEFAULT NULL,
   statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–22 EXPORT_DATABASE_STATS Procedure Parameters

Parameter	Description
stattab	User stat table identifier describing where to store the statistics
statid	Identifier (optional) to associate with these statistics within stattab
statown	Schema containing stattab. If statown is NULL, then it is assumed that every schema in the database contains a user statistics table with the name stattab.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

IMPORT_COLUMN_STATS procedure

This procedure retrieves statistics for a particular column from the user stat table identified by stattab and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_COLUMN_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   colname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–23 IMPORT_COLUMN_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table to which this column belongs.
colname	Name of the column.

Table 49–23 IMPORT_COLUMN_STATS Procedure Parameters

Parameter	Description
partname	Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition column statistics are imported.
stattab	User stat table identifier describing from where to retrieve the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user stat table.

IMPORT_INDEX_STATS procedure

This procedure retrieves statistics for a particular index from the user stat table identified by stattab and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_INDEX_STATS (
   ownname  VARCHAR2,
   indname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–24 IMPORT_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
indname	Name of the index.

Table 49–24 IMPORT_INDEX_STATS Procedure Parameters

Parameter	Description
partname	Name of the index partition. If the index is partitioned and if partname is NULL, then global and partition index statistics are imported.
stattab	User stat table identifier describing from where to retrieve the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user stat table.

IMPORT_TABLE_STATS procedure

This procedure retrieves statistics for a particular table from the user stat table identified by stattab and stores them in the dictionary. Cascade results in all index and column stats associated with the specified table being imported as well.

Syntax

```
DBMS_STATS.IMPORT_TABLE_STATS (
   ownname  VARCHAR2,
   tabname  VARCHAR2,
   partname VARCHAR2 DEFAULT NULL,
   stattab  VARCHAR2,
   statid   VARCHAR2 DEFAULT NULL,
   cascade  BOOLEAN  DEFAULT TRUE,
   statown  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–25 IMPORT_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
tabname	Name of the table.

Table 49–25 IMPORT_TABLE_STATS Procedure Parameters

Parameter	Description
partname	Name of the table partition. If the table is partitioned and if partname is NULL, then global and partition table statistics are imported.
stattab	User stat table identifier describing from where to retrieve the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
cascade	If true, then column and index statistics for this table are also imported.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user stat table.

IMPORT_SCHEMA_STATS procedure

This procedure retrieves statistics for all objects in the schema identified by ownname from the user stat table and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_SCHEMA_STATS (
   ownname VARCHAR2,
   stattab VARCHAR2,
   statid  VARCHAR2 DEFAULT NULL,
   statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–26 IMPORT_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	Name of the schema.
stattab	User stat table identifier describing from where to retrieve the statistics.

Table 49–26 IMPORT_SCHEMA_STATS Procedure Parameters

Parameter	Description
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user stat table.

IMPORT_DATABASE_STATS procedure

This procedure retrieves statistics for all objects in the database from the user stat table(s) and stores them in the dictionary.

Syntax

```
DBMS_STATS.IMPORT_DATABASE_STATS (
   stattab VARCHAR2,
   statid  VARCHAR2 DEFAULT NULL,
   statown VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–27 IMPORT_DATABASE_STATS Procedure Parameters

Parameter	Description
stattab	User stat table identifier describing from where to retrieve the statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab. If statown is NULL, then it is assumed that every schema in the database contains a user statistics table with the name stattab.

Exceptions

ORA-20000: Object does not exist or insufficient privileges.

ORA-20001: Invalid or inconsistent values in the user stat table.

Gathering Optimizer Statistics

The following procedures enable the gathering of certain classes of optimizer statistics, with possible performance improvements over the ANALYZE command:

GATHER_INDEX_STATS
GATHER_TABLE_STATS
GATHER_SCHEMA_STATS
GATHER_DATABASE_STATS

The statown, stattab, and statid parameters instruct the package to backup current statistics in the specified table before gathering new statistics.

Oracle also provides the following procedure for generating some statistics for derived objects when we have sufficient statistics on related objects:

GENERATE_STATS

GATHER_INDEX_STATS procedure

This procedure gathers index statistics. It is equivalent to running ANALYZE INDEX [ownname.]indname [PARTITION partname] COMPUTE STATISTICS | ESTIMATE STATISTICS SAMPLE estimate_percent PERCENT

It does not execute in parallel.

Syntax

```
DBMS_STATS_GATHER_INDEX_STATS (
   ownname          VARCHAR2,
   indname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT NULL,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–28 GATHER_INDEX_STATS Procedure Parameters

Parameter	Description
ownname	Schema of index to analyze.
indname	Name of index.

Table 49–28 GATHER_INDEX_STATS Procedure Parameters

Parameter	Description
partname	Name of partition.
estimate_percent	Percentage of rows to estimate (NULL means compute). The valid range is [0.000001,100). This value may be increased automatically to achieve better results.
stattab	User stat table identifier describing where to save the current statistics.
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Index does not exist or insufficient privileges.

ORA-20001: Bad input value.

GATHER_TABLE_STATS procedure

This procedure gathers table and column (and index) statistics. It attempts to parallelize as much of the work as possible, but there are some restrictions as described in the individual parameters. This operation does not parallelize if the user does not have select privilege on the table being analyzed.

Syntax

```
DBMS_STATS.GATHER_TABLE_STATS (
   ownname          VARCHAR2,
   tabname          VARCHAR2,
   partname         VARCHAR2 DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT NULL,
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
   degree           NUMBER   DEFAULT NULL,
   granularity      VARCHAR2 DEFAULT 'DEFAULT',
   cascade          BOOLEAN  DEFAULT FALSE,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   statown          VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–29 GATHER_TABLE_STATS Procedure Parameters

Parameter	Description
ownname	Schema of table to analyze.
tabname	Name of table.
partname	Name of partition.
estimate_percent	Percentage of rows to estimate (NULL means compute) The valid range is [0.000001,100). This value may be increased automatically to achieve better results.
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.
method_opt	Method options of the following format (the phrase 'SIZE 1' is required to ensure gathering statistics in parallel and for use with the phrase hidden): FOR ALL [INDEXED | HIDDEN] COLUMNS [SIZE integer] FOR COLUMNS [SIZE integer] column|attribute [,column|attribute ...] Optimizer related table statistics are always gathered.
degree	Degree of parallelism (NULL means use table default value).
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). DEFAULT: Gather global- and partition-level statistics. SUBPARTITION: Gather subpartition-level statistics. PARTITION: Gather partition-level statistics. GLOBAL: Gather global statistics. ALL: Gather all (subpartition, partition, and global) statistics.
cascade	Gather statistics on the indexes for this table. Index statistics gathering is not parallelized. Using this option is equivalent to running the gather_index_stats procedure on each of the table's indexes.
stattab	User stat table identifier describing where to save the current statistics.

Table 49–29 GATHER_TABLE_STATS Procedure Parameters

Parameter	Description
statid	Identifier (optional) to associate with these statistics within stattab.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Table does not exist or insufficient privileges.

ORA-20001: Bad input value.

GATHER_SCHEMA_STATS procedure

This procedure gathers statistics for all objects in a schema.

Syntax

```
DBMS_STATS.GATHER_SCHEMA_STATS (
   ownname          VARCHAR2,
   estimate_percent NUMBER   DEFAULT NULL,
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
   degree           NUMBER   DEFAULT NULL,
   granularity      VARCHAR2 DEFAULT 'DEFAULT',
   cascade          BOOLEAN  DEFAULT FALSE);

DBMS_STATS.GATHER_SCHEMA_STATS (
   ownname          VARCHAR2,
   estimate_percent NUMBER   DEFAULT NULL,
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
   degree           NUMBER   DEFAULT NULL,
   granularity      VARCHAR2 DEFAULT 'DEFAULT',
   cascade          BOOLEAN  DEFAULT FALSE,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   options          VARCHAR2 DEFAULT 'GATHER',
   objlist     OUT  ObjectTab,
   statown          VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–30 GATHER_SCHEMA_STATS Procedure Parameters

Parameter	Description
ownname	Schema to analyze (NULL means current schema).
estimate_percent	Percentage of rows to estimate (NULL means compute): The valid range is [0.000001,100).
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.
method_opt	Method options of the following format (the phrase 'SIZE 1' is required to ensure gathering statistics in parallel and for use with the phrase hidden): FOR ALL [INDEXED | HIDDEN] COLUMNS [SIZE integer] This value is passed to all of the individual tables.
degree	Degree of parallelism (NULL means use table default value).
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). DEFAULT: Gather global- and partition-level statistics. SUBPARTITION: Gather subpartition-level statistics. PARTITION: Gather partition-level statistics. GLOBAL: Gather global statistics. ALL: Gather all (subpartition, partition, and global) statistics.
cascade	Gather statistics on the indexes as well. Index statistics gathering is not parallelized. Using this option is equivalent to running the gather_index_stats procedure on each of the indexes in the schema in addition to gathering table and column statistics.
stattab	User stat table identifier describing where to save the current statistics.
statid	Identifier (optional) to associate with these statistics within stattab.

Table 49–30 GATHER_SCHEMA_STATS Procedure Parameters

Parameter	Description
options	Further specification of which objects to gather statistics for: GATHER: Gather statistics on all objects in the schema. GATHER STALE: Gather statistics on stale objects as determined by looking at the *_tab_modifications views. Also, return a list of objects found to be stale. GATHER EMPTY: Gather statistics on objects which currently have no statistics. also, return a list of objects found to have no statistics. LIST STALE: Return list of stale objects as determined by looking at the *_tab_modifications views. LIST EMPTY: Return list of objects which currently have no statistics.
objlist	List of objects found to be stale or empty.
statown	Schema containing stattab (if different than ownname).

Exceptions

ORA-20000: Schema does not exist or insufficient privileges.

ORA-20001: Bad input value.

GATHER_DATABASE_STATS procedure

This procedure gathers statistics for all objects in the database.

Syntax

```
DBMS_STATS.GATHER_DATABASE_STATS (
   estimate_percent NUMBER   DEFAULT NULL,
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
   degree           NUMBER   DEFAULT NULL,
   granularity      VARCHAR2 DEFAULT 'DEFAULT',
   cascade          BOOLEAN  DEFAULT FALSE);
```

```
DBMS_STATS.GATHER_DATABASE_STATS (
   estimate_percent NUMBER   DEFAULT NULL,
   block_sample     BOOLEAN  DEFAULT FALSE,
   method_opt       VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
   degree           NUMBER   DEFAULT NULL,
   granularity      VARCHAR2 DEFAULT 'DEFAULT',
   cascade          BOOLEAN  DEFAULT FALSE,
   stattab          VARCHAR2 DEFAULT NULL,
   statid           VARCHAR2 DEFAULT NULL,
   options          VARCHAR2 DEFAULT 'GATHER',
   objlist     OUT  ObjectTab,
   statown          VARCHAR2 DEFAULT NULL);
```

Parameters

Table 49–31 GATHER_DATABASE_STATS Procedure Parameters

Parameter	Description
estimate_percent	Percentage of rows to estimate (NULL means compute): The valid range is [0.000001,100).
block_sample	Whether or not to use random block sampling instead of random row sampling. Random block sampling is more efficient, but if the data is not randomly distributed on disk, then the sample values may be somewhat correlated. Only pertinent when doing an estimate statistics.
method_opt	Method options of the following format (the phrase 'SIZE 1' is required to ensure gathering statistics in parallel and for use with the phrase hidden): FOR ALL [INDEXED | HIDDEN] COLUMNS [SIZE integer] This value is passed to all of the individual tables.
degree	Degree of parallelism (NULL means use table default value).
granularity	Granularity of statistics to collect (only pertinent if the table is partitioned). DEFAULT: Gather global- and partition-level statistics. SUBPARTITION: Gather subpartition-level statistics. PARTITION: Gather partition-level statistics. GLOBAL: Gather global statistics. ALL: Gather all (subpartition, partition, and global) statistics.

Table 49–31 GATHER_DATABASE_STATS Procedure Parameters

Parameter	Description
cascade	Gather statistics on the indexes as well. Index statistics gathering is not parallelized. Using this option is equivalent to running the `gather_index_stats` procedure on each of the indexes in the database in addition to gathering table and column statistics.
stattab	User stat table identifier describing where to save the current statistics. The statistics table is assumed to reside in the same schema as the object being analyzed, so there must be one such table in each schema to use this option.
statid	Identifier (optional) to associate with these statistics within `stattab`.
options	Further specification of which objects to gather statistics for: `GATHER STALE`: Gather statistics on stale objects as determined by looking at the `*_tab_modifications` views. Also, return a list of objects found to be stale. `GATHER EMPTY`: Gather statistics on objects which currently have no statistics. also, return a list of objects found to have no statistics. `LIST STALE`: Return list of stale objects as determined by looking at the `*_tab_modifications` views. `LIST EMPTY`: Return list of objects which currently have no statistics.
objlist	List of objects found to be stale or empty.
statown	Schema containing `stattab` (if different than `ownname`).

Exceptions

`ORA-20000`: Insufficient privileges.

`ORA-20001`: Bad input value.

GENERATE_STATS procedure

This procedure generates object statistics from previously collected statistics of related objects. For fully populated schemas, the gather procedures should be used instead when more accurate statistics are desired.The currently supported objects are b-tree and bitmap indexes.

Syntax

```
DBMS_STATS.GENERATE_STATS (
   ownname   VARCHAR2,
   objname   VARCHAR2,
   organized NUMBER DEFAULT 7);
```

Parameters

Table 49–32 GENERATE_STATS Procedure Parameters

Parameter	Description
ownname	Schema of object.
objname	Name of object.
organized	Amount of ordering associated between the index and its underlying table. A heavily organized index would have consecutive index keys referring to consecutive rows on disk for the table (the same block). A heavily disorganized index would have consecutive keys referencing different table blocks on disk. This parameter is only used for b-tree indexes. The number can be in the range of 0-10, with 0 representing a completely organized index and 10 a completely disorganized one.

Exceptions

ORA-20000: Unsupported object type of object does not exist.

ORA-20001: Invalid option or invalid statistics.

Example

Scenario There has been a lot of modification against the emp table since the last time statistics were gathered. To ensure that the cost-based optimizer is still picking the best plan, statistics should be gathered once again; however, the user is concerned that new statistics will cause the optimizer to choose bad plans when the current ones are acceptable. The user can do the following:

```
BEGIN
   DBMS_STATS.CREATE_STAT_TABLE ('scott', 'savestats');
   DBMS_STATS.GATHER_TABLE_STATS ('scott', 'emp', 5, stattab => 'savestats');
END;
```

This operation gathers new statistics on emp, but first saves the original statistics in a user stat table: emp.savestats.

If the user believes that the new statistics are causing the optimizer to generate poor plans, then the original stats can be restored as follows:

```
BEGIN
   DBMS_STATS.DELETE_TABLE_STATS ('scott', 'emp');
   DBMS_STATS.IMPORT_TABLE_STATS ('scott', 'emp', stattab => 'savestats');
END;
```

50

DBMS_TRACE

Oracle8*i* PL/SQL provides an API for tracing the execution of PL/SQL programs on the server. You can use the Trace API, implemented on the server as the DBMS_TRACE package, to trace PL/SQL functions, procedures, and exceptions.

DBMS_TRACE provides subprograms to start and stop PL/SQL tracing in a session. The trace data gets collected as the program executes, and it is written out to the Oracle Server trace file.

A typical session involves:

- Starting PL/SQL tracing in session (DBMS_TRACE.SET_PLSQL_TRACE).
- Running application which is to be traced.
- Stopping PL/SQL tracing in session (DBMS_TRACE.CLEAR_PLSQL_TRACE).

Requirements

This package must be created under SYS.

Restrictions

You cannot use PL/SQL tracing with the multi-threaded server (MTS).

Constants

```
trace_all_calls          constant INTEGER := 1;
trace_enabled_calls      constant INTEGER := 2;
trace_all_exceptions     constant INTEGER := 4;
trace_enabled_exceptions constant INTEGER := 8;
/*
 * The version of the trace package. These constants will change as the
 * package evolves. Use the PLSQL_TRACE_VERSION procedure (described below)
 * to get the current version of the package.
 */
trace_major_version      constant BINARY_INTEGER := 1;
trace_minor_version      constant BINARY_INTEGER := 0;
```

Using DBMS_TRACE

Controlling Data Volume

Profiling large applications may produce a huge volume of data. You can control the volume of data collected by *enabling* specific program units for trace data collection.

You can enable a program unit by compiling it debug. This can be done in one of two ways:

```
alter session set plsql_debug=true;
create or replace ... /* create the library units - debug information will be
generated */
```

or:

```
/* recompile specific library unit with debug option */
alter [PROCEDURE | FUNCTION | PACKAGE BODY]  <libunit-name> compile debug;
```

Note: This second method cannot be used for anonymous blocks.

Collecting Trace Data

Tracing Calls Two levels of call tracing are available:

- Level 1: Trace all calls. This corresponds to the constant `trace_all_calls`.
- Level 2: Trace calls to enabled program units only. This corresponds to the constant `trace_enabled_calls`.

Enabling cannot be detected for remote procedure calls (RPCs); hence, RPCs are only traced with level 1.

Tracing Exceptions Two levels of exception tracing are available:

- Level 1: Trace all exceptions. This corresponds to `trace_all_exceptions`.
- Level 2: Trace exceptions raised in enabled program units only. This corresponds to `trace_enabled_exceptions`.

Note: For both call and exception tracing, level 1 overrides level 2. For example, if both level 1 and level 2 are enabled, then level 1 takes precedence.

Collected Data

If tracing is requested only for enabled program units, and if the current program unit is not enabled, then no trace data is written.

If the current program unit is enabled, then call tracing writes out the program unit type, name, and stack depth.

If the current program unit is *not* enabled, then call tracing writes out the program unit type, line number, and the stack depth.

Exception tracing writes out the line number. Raising the exception shows information on whether the exception is user-defined or pre-defined and the exception number, in the case of pre-defined exceptions.

Summary of Subprograms

Table 50–1 DBMS_TRACE Package Subprograms

Subprogram	Description
SET_PLSQL_TRACE procedure on page 50-4	Starts tracing in the current session.
CLEAR_PLSQL_TRACE procedure on page 50-4	Stops trace data dumping in session.
PLSQL_TRACE_VERSION procedure on page 50-5	Gets the version number of the trace package.

SET_PLSQL_TRACE procedure

This procedure enables PL/SQL trace data collection.

Syntax

```
DBMS_TRACE.SET_PLSQL_TRACE (
   trace_level INTEGER);
```

Parameters

Table 50–2 SET_PLSQL_TRACE Procedure Parameters

Parameter	Description
trace_level	This must be one of the constants trace_all_calls, trace_enabled_calls, trace_all_exceptions, or trace_enabled_exceptions. See "Collecting Trace Data" on page 50-3 for more information.

CLEAR_PLSQL_TRACE procedure

This procedure disables trace data collection.

Syntax

```
DBMS_TRACE.CLEAR_PLSQL_TRACE;
```

Parameters

None.

PLSQL_TRACE_VERSION procedure

This procedure gets the version number of the trace package. It returns the major and minor version number of the DBMS_TRACE package.

Syntax

```
DBMS_TRACE.PLSQL_TRACE_VERSION (
   major OUT BINARY_INTEGER,
   minor OUT BINARY_INTEGER);
```

Parameters

Table 50–3 PLSQL_TRACE_VERSION Procedure Parameters

Parameter	Description
major	Major version number of DBMS_TRACE.
minor	Minor version number of DBMS_TRACE.

51

DBMS_TRANSACTION

This package provides access to SQL transaction statements from stored procedures.

See Also: *Oracle8i SQL Reference*

Requirements

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

Summary of Subprograms

Table 51–1 DBMS_TRANSACTION Package Subprograms

Subprogram
READ_ONLY procedure on page 51-3
READ_WRITE procedure on page 51-3
ADVISE_ROLLBACK procedure on page 51-3
ADVISE_NOTHING procedure on page 51-4
ADVISE_COMMIT procedure on page 51-4
USE_ROLLBACK_SEGMENT procedure on page 51-4
COMMIT_COMMENT procedure on page 51-5
COMMIT_FORCE procedure on page 51-5
COMMIT procedure on page 51-6
SAVEPOINT procedure on page 51-6
ROLLBACK procedure on page 51-7
ROLLBACK_SAVEPOINT procedure on page 51-7
ROLLBACK_FORCE procedure on page 51-8
BEGIN_DISCRETE_TRANSACTION procedure on page 51-8
PURGE_MIXED procedure on page 51-9
PURGE_LOST_DB_ENTRY procedure on page 51-10
LOCAL_TRANSACTION_ID function on page 51-12
STEP_ID function on page 51-12

READ_ONLY procedure

This procedure is equivalent to following SQL statement:

```
SET TRANSACTION READ ONLY
```

Syntax

```
DBMS_TRANSACTION.READ_ONLY;
```

Parameters

None.

READ_WRITE procedure

This procedure is equivalent to following SQL statement:

```
SET TRANSACTION READ WRITE
```

Syntax

```
DBMS_TRANSACTION.READ_WRITE;
```

Parameters

None.

ADVISE_ROLLBACK procedure

This procedure is equivalent to following SQL statement:

```
ALTER SESSION ADVISE ROLLBACK
```

Syntax

```
DBMS_TRANSACTION.ADVISE_ROLLBACK;
```

Parameters

None.

ADVISE_NOTHING procedure

This procedure is equivalent to following SQL statement:

```
ALTER SESSION ADVISE NOTHING
```

Syntax

```
DBMS_TRANSACTION.ADVISE_NOTHING;
```

Parameters

None.

ADVISE_COMMIT procedure

This procedure is equivalent to following SQL statement:

```
ALTER SESSION ADVISE COMMIT
```

Syntax

```
DBMS_TRANSACTION.ADVISE_COMMIT;
```

Parameters

None.

USE_ROLLBACK_SEGMENT procedure

This procedure is equivalent to following SQL statement:

```
SET TRANSACTION USE ROLLBACK SEGMENT <rb_seg_name>
```

Syntax

```
DBMS_TRANSACTION.USE_ROLLBACK_SEGMENT (
   rb_name VARCHAR2);
```

Parameters

Table 51–2 USE_ROLLBACK_SEGMENT Procedure Parameters

Parameter	Description
rb_name	Name of rollback segment to use.

COMMIT_COMMENT procedure

This procedure is equivalent to following SQL statement:

```
COMMIT COMMENT <text>
```

Syntax

```
DBMS_TRANSACTION.COMMIT_COMMENT (
   cmnt VARCHAR2);
```

Parameters

Table 51–3 COMMIT_COMMENT Procedure Parameters

Parameter	Description
cmnt	Comment to associate with this commit.

COMMIT_FORCE procedure

This procedure is equivalent to following SQL statement:

```
COMMIT FORCE <text>, <number>"
```

Syntax

```
DBMS_TRANSACTION.COMMIT_FORCE (
   xid VARCHAR2,
   scn VARCHAR2 DEFAULT NULL);
```

Parameters

Table 51–4 COMMIT_FORCE Procedure Parameters

Parameter	Description
xid	Local or global transaction ID.
scn	System change number.

COMMIT procedure

This procedure is equivalent to following SQL statement:

```
COMMIT
```

Here for completeness. This is already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.COMMIT;
```

Parameters

None.

SAVEPOINT procedure

This procedure is equivalent to following SQL statement:

```
SAVEPOINT <savepoint_name>
```

Here for completeness. This is already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.SAVEPOINT (
   savept VARCHAR2);
```

Parameters

Table 51–5 SAVEPOINT Procedure Parameters

Parameter	Description
savept	Savepoint identifier.

ROLLBACK procedure

This procedure is equivalent to following SQL statement:

```
ROLLBACK
```

Here for completeness. This is already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.ROLLBACK;
```

Parameters

None.

ROLLBACK_SAVEPOINT procedure

This procedure is equivalent to following SQL statement:

```
ROLLBACK TO SAVEPOINT <savepoint_name>
```

Here for completeness. This is already implemented as part of PL/SQL.

Syntax

```
DBMS_TRANSACTION.ROLLBACK_SAVEPOINT (
   savept VARCHAR2);
```

Parameters

Table 51–6 ROLLBACK_SAVEPOINT Procedure Parameters

Parameter	Description
savept	Savepoint identifier.

ROLLBACK_FORCE procedure

This procedure is equivalent to following SQL statement:

```
ROLLBACK FORCE <text>
```

Syntax

```
DBMS_TRANSACTION.ROLLBACK_FORCE (
   xid VARCHAR2);
```

Parameters

Table 51–7 ROLLBACK_FORCE Procedure Parameters

Parameter	Description
xid	Local or global transaction ID.

BEGIN_DISCRETE_TRANSACTION procedure

This procedure sets "discrete transaction mode" for this transaction.

Syntax

```
DBMS_TRANSACTION.BEGIN_DISCRETE_TRANSACTION;
```

Parameters

None.

Exceptions

Table 51–8 BEGIN_DISCRETE_TRANSACTION Procedure Exceptions

Exception	Description
ORA-08175	A transaction attempted an operation which cannot be performed as a discrete transaction. If this exception is encountered, then rollback and retry the transaction

Table 51–8 BEGIN_DISCRETE_TRANSACTION Procedure Exceptions

Exception	Description
ORA-08176	A transaction encountered data changed by an operation that does not generate rollback data: create index, direct load or discrete transaction. If this exception is encountered, then retry the operation that received the exception.

Example

```
DISCRETE_TRANSACTION_FAILED exception;
    pragma exception_init(DISCRETE_TRANSACTION_FAILED, -8175);
CONSISTENT_READ_FAILURE exception;
    pragma exception_init(CONSISTENT_READ_FAILURE, -8176);
```

PURGE_MIXED procedure

When in-doubt transactions are forced to commit or rollback (instead of letting automatic recovery resolve their outcomes), there is a possibility that a transaction can have a mixed outcome: Some sites commit, and others rollback. Such inconsistency cannot be resolved automatically by Oracle; however, Oracle flags entries in DBA_2PC_PENDING by setting the MIXED column to a value of 'yes'.

Oracle never automatically deletes information about a mixed outcome transaction. When the application or DBA is certain that all inconsistencies that might have arisen as a result of the mixed transaction have been resolved, this procedure can be used to delete the information about a given mixed outcome transaction.

Syntax

```
DBMS_TRANSACTION.PURGE_MIXED (
   xid VARCHAR2);
```

Parameters

Table 51–9 PURGE_MIXED Procedure Parameters

Parameter	Description
xid	Must be set to the value of the LOCAL_TRAN_ID column in the DBA_2PC_PENDING table.

PURGE_LOST_DB_ENTRY procedure

When a failure occurs during commit processing, automatic recovery consistently resolves the results at all sites involved in the transaction. However, if the remote database is destroyed or recreated before recovery completes, then the entries used to control recovery in DBA_2PC_PENDING and associated tables are never removed, and recovery will periodically retry. Procedure PURGE_LOST_DB_ENTRY enables removal of such transactions from the local site.

Syntax

```
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY (
   xid VARCHAR2);
```

WARNING: PURGE_LOST_DB_ENTRY should *only* be used when the other database is lost or has been recreated. Any other use may leave the other database in an unrecoverable or inconsistent state.

Before automatic recovery runs, the transaction may show up in DBA_2PC_PENDING as state "collecting", "committed", or "prepared". If the DBA has forced an in-doubt transaction to have a particular result by using "commit force" or "rollback force", then states "forced commit" or "forced rollback" may also appear. Automatic recovery normally deletes entries in any of these states. The only exception is when recovery finds a forced transaction which is in a state inconsistent with other sites in the transaction; in this case, the entry is left in the table and the MIXED column has the value 'yes'.

However, under certain conditions, it may not be possible for automatic recovery to run. For example, a remote database may have been permanently lost. Even if it is recreated, it gets a new database ID, so that recovery cannot identify it (a possible symptom is ORA-02062). In this case, the DBA may use the procedure PURGE_LOST_DB_ENTRY to clean up the entries in any state other than "prepared". The DBA does not need to be in any particular hurry to resolve these entries, because they are not holding any database resources.

The following table indicates what the various states indicate about the transaction and what the DBA actions should be:

Table 51–10 PURGE_LOST_DB_ENTRY Procedure States

State of Column	State of Global Transaction	State of Local Transaction	Normal DBA Action	Alternative DBA Action
Collecting	Rolled back	Rolled back	None	PURGE_LOST_DB_ENTRY (1)
Committed	Committed	Committed	None	PURGE_LOST_DB_ENTRY (1)
Prepared	Unknown	Prepared	None	FORCE COMMIT or ROLLBACK
Forced commit	Unknown	Committed	None	PURGE_LOST_DB_ENTRY (1)
Forced rollback	Unknown	Rolled back	None	PURGE_LOST_DB_ENTRY (1)
Forced commit (mixed)	Mixed	Committed	(2)	
Forced rollback (mixed)	Mixed	Rolled back	(2)	

Note 1: Use only if significant reconfiguration has occurred so that automatic recovery cannot resolve the transaction. Examples are total loss of the remote database, reconfiguration in software resulting in loss of two-phase commit capability, or loss of information from an external transaction coordinator such as a TP monitor.

Note 2: Examine and take any manual action to remove inconsistencies, then use the procedure PURGE_MIXED.

Parameters

Table 51–11 PURGE_LOST_DB_ENTRY Procedure Parameters

Parameter	Description
xid	Must be set to the value of the LOCAL_TRAN_ID column in the DBA_2PC_PENDING table.

LOCAL_TRANSACTION_ID function

This function returns the local (to instance) unique identifier for current transaction. It returns null if there is no current transaction.

Syntax

```
DBMS_TRANSACTION.LOCAL_TRANSACTION_ID (
   create_transaction BOOLEAN := FALSE)
  RETURN VARCHAR2;
```

Parameters

Table 51–12 LOCAL_TRANSACTION_ID Function Parameters

Parameter	Description
create_transaction	If true, then start a transaction if one is not currently active.

STEP_ID function

This function returns local (to local transaction) unique positive integer that orders the DML operations of a transaction.

Syntax

```
DBMS_TRANSACTION.STEP_ID
   RETURN NUMBER;
```

Parameters

None.

52

DBMS_TTS

This package checks if the transportable set is self-contained. All violations are inserted into a temporary table that can be selected from the view `TRANSPORT_SET_VIOLATIONS`.

See Also: *Oracle8i Administrator's Guide* and *Oracle8i Migration*

Exceptions

```
ts_not_found  EXCEPTION;
PRAGMA exception_init(ts_not_found, -29304);
ts_not_found_num NUMBER := -29304;

invalid_ts_list  EXCEPTION;
PRAGMA exception_init(invalid_ts_list, -29346);
invalid_ts_list_num NUMBER := -29346;

sys_or_tmp_ts     EXCEPTION;
PRAGMA exception_init(sys_or_tmp_ts, -29351);
sys_or_tmp_ts_num NUMBER := -29351;
```

Summary of Subprograms

These two procedures are designed to be called by database administrators.

Table 52–1 DBMS_TTS Package Subprograms

Subprogram	Description
TRANSPORT_SET_CHECK procedure on page 52-2	Checks if a set of tablespaces (to be transported) is self-contained.
DOWNGRADE procedure on page 52-3	Downgrades transportable tablespace related data.

TRANSPORT_SET_CHECK procedure

This procedure checks if a set of tablespaces (to be transported) is self-contained. After calling this procedure, the user may select from a view to see a list of violations, if there are any. If the view does not return any rows, then the set of tablespaces is self-contained. For example,

```
SVRMGR> EXECUTE TRANSPORT_SET_CHECK('foo,bar', TRUE);
SVRMGR> SELECT * FROM TRANSPORT_SET_VIOLATIONS;
```

Syntax

```
DBMS_TTS.TRANSPORT_SET_CHECK (
   ts_list          IN VARCHAR2,
   incl_constraints IN BOOLEAN);
```

Parameters

Table 52–2 TRANSPORT_SET_CHECK Procedure Parameters

Parameter	Description
ts_list	List of tablespace, separated by comma.
incl_constraints	TRUE if you'd like to count in referential integrity constraints when examining if the set of tablespaces is self-contained.

DOWNGRADE procedure

This procedure downgrades transportable tablespace related data.

Syntax

```
DBMS_TTS.DOWNGRADE;
```

Parameters

None.

53

DBMS_UTILITY

This package provides various utility subprograms.

DBMS_UTILITY submits a job for each partition. It is the users responsibility to control the number of concurrent jobs by setting the INIT.ORA parameter JOB_QUEUE_PROCESSES correctly.There is minimal error checking for correct syntax. Any error is reported in SNP trace files.

Requirements

DBMS_UTILITY runs with the privileges of the calling user for the NAME_RESOLVE, COMPILE_SCHEMA, and ANALYZE_SCHEMA procedures. This is necessary so that the SQL works correctly.

This does not run as SYS. The privileges are checked via DBMS_DDL.

Types

```
type uncl_array IS TABLE OF VARCHAR2(227) INDEX BY BINARY_INTEGER;
```
Lists of "USER"."NAME"."COLUMN"@LINK should be stored here.

```
type name_array IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
```
Lists of NAME should be stored here.

```
type dblink_array IS TABLE OF VARCHAR2(128) INDEX BY BINARY_INTEGER;
```
Lists of database links should be stored here.

```
TYPE index_table_type IS TABLE OF BINARY_INTEGER INDEX BY BINARY_INTEGER;
```
The order in which objects should be generated is returned here.

```
TYPE number_array IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```
The order in which objects should be generated is returned here for users.

```
  TYPE instance_record IS RECORD (
       inst_number   NUMBER,
       inst_name     VARCHAR2(60));
  TYPE instance_table IS TABLE OF instance_record INDEX BY BINARY_INTEGER;
```
The list of active instance number and instance name.

The starting index of instance_table is 1; instance_table is dense.

Summary of Subprograms

Table 53–1 DBMS_UTILITY Package Subprograms **(Page 1 of 3)**

Subprogram	Description
COMPILE_SCHEMA procedure on page 53-4	Compiles all procedures, functions, packages, and triggers in the specified schema.
ANALYZE_SCHEMA procedure on page 53-5	Analyzes all the tables, clusters, and indexes in a schema.
ANALYZE_DATABASE procedure on page 53-6	Analyzes all the tables, clusters, and indexes in a database.

Table 53–1 DBMS_UTILITY Package Subprograms **(Page 2 of 3)**

Subprogram	Description
FORMAT_ERROR_STACK function on page 53-7	Formats the current error stack.
FORMAT_CALL_STACK function on page 53-7	Formats the current call stack.
IS_PARALLEL_SERVER function on page 53-8	Finds out if this database is running in parallel server mode.
GET_TIME function on page 53-8	Finds out the current time in 100th's of a second.
GET_PARAMETER_VALUE function on page 53-9	Gets the value of specified init.ora parameter.
NAME_RESOLVE procedure on page 53-10	Resolves the given name.
NAME_TOKENIZE procedure on page 53-12	Calls the parser to parse the given name.
COMMA_TO_TABLE procedure on page 53-12	Converts a comma-separated list of names into a PL/SQL table of names.
TABLE_TO_COMMA procedure on page 53-13	Converts a PL/SQL table of names into a comma-separated list of names.
PORT_STRING function on page 53-14	Returns a string that uniquely identifies the version of Oracle and the operating system.
DB_VERSION procedure on page 53-14	Returns version information for the database.
MAKE_DATA_BLOCK_ADDRESS function on page 53-15	Creates a data block address given a file number and a block number.
DATA_BLOCK_ADDRESS_FILE function on page 53-15	Gets the file number part of a data block address.
DATA_BLOCK_ADDRESS_BLOCK function on page 53-16	Gets the block number part of a data block address.
GET_HASH_VALUE function on page 53-17	Computes a hash value for the given string.
ANALYZE_PART_OBJECT procedure on page 53-18	

Table 53–1 DBMS_UTILITY Package Subprograms **(Page 3 of 3)**

Subprogram	Description
EXEC_DDL_STATEMENT procedure on page 53-19	Executes the DDL statement in parse_string.
CURRENT_INSTANCE function on page 53-19	Returns the current connected instance number.
ACTIVE_INSTANCES procedure on page 53-19	

COMPILE_SCHEMA procedure

This procedure compiles all procedures, functions, packages, and triggers in the specified schema. After calling this procedure, you should select from view ALL_OBJECTS for items with status of INVALID to see if all objects were successfully compiled.

To see the errors associated with INVALID objects, you may use the Enterprise Manager command:

```
SHOW ERRORS <type> <schema>.<name>
```

Syntax

```
DBMS_UTILITY.COMPILE_SCHEMA (
   schema VARCHAR2);
```

Parameters

Table 53–2 COMPILE_SCHEMA Procedure Parameters

Parameter	Description
schema	Name of the schema.

Exceptions

Table 53–3 COMPILE_SCHEMA Procedure Exceptions

Exception	Description
ORA-20000	Insufficient privileges for some object in this schema.

ANALYZE_SCHEMA procedure

This procedure analyzes all the tables, clusters, and indexes in a schema.

Syntax

```
DBMS_UTILITY.ANALYZE_SCHEMA (
   schema           VARCHAR2,
   method           VARCHAR2,
   estimate_rows    NUMBER   DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT NULL,
   method_opt       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 53–4 ANALYZE_SCHEMA Procedure Parameters

Parameter	Description
schema	Name of the schema.
method	One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE, then either estimate_rows or estimate_percent must be non-zero.
estimate_rows	Number of rows to estimate.
estimate_percent	Percentage of rows to estimate. If estimate_rows is specified, then ignore this parameter.
method_opt	Method options of the following format: [FOR TABLE] [FOR ALL [INDEXED] COLUMNS] [SIZE n] [FOR ALL INDEXES]

Exceptions

Table 53–5 ANALYZE_SCHEMA Procedure Exceptions

Exception	Description
ORA-20000	Insufficient privileges for some object in this schema.

ANALYZE_DATABASE procedure

This procedure analyzes all the tables, clusters, and indexes in a database.

Syntax

```
DBMS_UTILITY.ANALYZE_DATABASE (
   method           VARCHAR2,
   estimate_rows    NUMBER   DEFAULT NULL,
   estimate_percent NUMBER   DEFAULT NULL,
   method_opt       VARCHAR2 DEFAULT NULL);
```

Parameters

Table 53–6 ANALYZE_DATABASE Procedure Parameters

Parameter	Description
method	One of ESTIMATE, COMPUTE or DELETE. If ESTIMATE, then either estimate_rows or estimate_percent must be non-zero.
estimate_rows	Number of rows to estimate.
estimate_percent	Percentage of rows to estimate. If estimate_rows is specified, then ignore this parameter.
method_opt	Method options of the following format: [FOR TABLE] [FOR ALL [INDEXED] COLUMNS] [SIZE n] [FOR ALL INDEXES]

Exceptions

Table 53–7 ANALYZE_DATABASE Procedure Exceptions

Exception	Description
ORA-20000	Insufficient privileges for some object in this database.

FORMAT_ERROR_STACK function

This function formats the current error stack. This can be used in exception handlers to look at the full error stack.

Syntax

```
DBMS_UTILITY.FORMAT_ERROR_STACK
  RETURN VARCHAR2;
```

Parameters

None.

Returns

This returns the error stack, up to 2000 bytes.

FORMAT_CALL_STACK function

This function formats the current call stack. This can be used on any stored procedure or trigger to access the call stack. This can be useful for debugging.

Syntax

```
DBMS_UTILITY.FORMAT_CALL_STACK
  RETURN VARCHAR2;
```

Parameters

None.

Pragmas

```
pragma restrict_references(format_call_stack,WNDS);
```

Returns

This returns the call stack, up to 2000 bytes.

IS_PARALLEL_SERVER function

This function finds out if this database is running in parallel server mode.

Syntax

```
DBMS_UTILITY.IS_PARALLEL_SERVER
  RETURN BOOLEAN;
```

Parameters

None.

Returns

This returns TRUE if this instance was started in parallel server mode, FALSE otherwise.

GET_TIME function

This function finds out the current time in 100th's of a second. It is primarily useful for determining elapsed time.

Syntax

```
DBMS_UTILITY.GET_TIME
  RETURN NUMBER;
```

Parameters

None.

Returns

Time is the number of 100th's of a second from some arbitrary epoch.

GET_PARAMETER_VALUE function

This function gets the value of specified `init.ora` parameter.

Syntax

```
DBMS_UTILITY.GET_PARAMETER_VALUE (
   parnam IN     VARCHAR2,
   intval IN OUT BINARY_INTEGER,
   strval IN OUT VARCHAR2)
  RETURN BINARY_INTEGER;
```

Parameters

Table 53–8 GET_PARAMETER_VALUE Function Parameters

Parameter	Description
parnam	Parameter name.
intval	Value of an integer parameter or the value length of a string parameter.
strval	Value of a string parameter.

Returns

Table 53–9 GET_PARAMETER_VALUE Function Returns

Return	Description
partyp	Parameter type: 0 if parameter is an integer / boolean parameter 1 if parameter is a string / file parameter

Example

```
DECLARE
  parnam VARCHAR2(256);
  intval BINARY_INTEGER;
  strval VARCHAR2(256);
  partyp BINARY_INTEGER;
BEGIN
  partyp := dbms_utility.get_parameter_value('max_dump_file_size',
                                             intval, strval);
  dbms_output.put('parameter value is: ');
```

```
  IF partyp = 1 THEN
    dbms_output.put_line(strval);
  ELSE
    dbms_output.put_line(intval);
  END IF;
  IF partyp = 1 THEN
    dbms_output.put('parameter value length is: ');
    dbms_output.put_line(intval);
  END IF;
  dbms_output.put('parameter type is: ');
  IF partyp = 1 THEN
    dbms_output.put_line('string');
  ELSE
    dbms_output.put_line('integer');
  END IF;
END;
```

NAME_RESOLVE procedure

This procedure resolves the given name, including synonym translation and authorization checking as necessary.

Syntax

```
DBMS_UTILITY.NAME_RESOLVE (
   name          IN  VARCHAR2,
   context       IN  NUMBER,
   schema        OUT VARCHAR2,
   part1         OUT VARCHAR2,
   part2         OUT VARCHAR2,
   dblink        OUT VARCHAR2,
   part1_type    OUT NUMBER,
   object_number OUT NUMBER);
```

Parameters

Table 53–10 NAME_RESOLVE Procedure Parameters

Parameter	Description
name	Name of the object. This can be of the form [[a.]b.]c[@d], where a, b, c are SQL identifier and d is a dblink. No syntax checking is performed on the dblink. If a dblink is specified, or if the name resolves to something with a dblink, then object is not resolved, but the schema, part1, part2 and dblink OUT parameters are filled in. a, b and c may be delimited identifiers, and may contain NLS characters (single and multi-byte).
context	Must be an integer between 0 and 8.
schema	Schema of the object: c. If no schema is specified in name, then the schema is determined by resolving the name.
part1	First part of the name. The type of this name is specified part1_type (synonym, procedure or package).
part2	If this is non-NULL, then this is a procedure name within the package indicated by part1.
dblink	If this is non-NULL, then a database link was either specified as part of name or name was a synonym which resolved to something with a database link. In this later case, part1_type indicates a synonym.
part1_type	Type of part1 is: 5 - synonym 7 - procedure (top level) 8 - function (top level) 9 - package If a synonym, then it means that name is a synonym that translates to something with a database link. In this case, if further name translation is desired, then you must call the DBMS_UTILITY.NAME_RESOLVE procedure on this remote node.

Exceptions

All errors are handled by raising exceptions. A wide variety of exceptions are possible, based on the various syntax error that are possible when specifying object names.

NAME_TOKENIZE procedure

This procedure calls the parser to parse the given name as "a [. b [. c]][@ dblink]". It strips double quotes, or converts to uppercase if there are no quotes. It ignores comments of all sorts, and does no semantic analysis. Missing values are left as NULL.

Syntax

```
DBMS_UTILITY.NAME_TOKENIZE (
   name    IN  VARCHAR2,
   a       OUT VARCHAR2,
   b       OUT VARCHAR2,
   c       OUT VARCHAR2,
   dblink  OUT VARCHAR2,
   nextpos OUT BINARY_INTEGER);
```

Parameters

For each of a, b, c, dblink, tell where the following token starts in anext, bnext, cnext, dnext respectively.

COMMA_TO_TABLE procedure

This procedure converts a comma-separated list of names into a PL/SQL table of names. This uses NAME_TOKENIZE to figure out what are names and what are commas.

Syntax

```
DBMS_UTILITY.COMMA_TO_TABLE (
   list   IN  VARCHAR2,
   tablen OUT BINARY_INTEGER,
   tab    OUT UNCL_ARRAY);
```

Parameters

Table 53–11 COMMA_TO_TABLE Procedure Parameters

Parameter	Description
list	Comma separated list of tables.
tablen	Number of tables in the PL/SQL table.
tab	PL/SQL table which contains list of table names.

Returns

A PL/SQL table is returned, with values `1..n` and `n+1 is null`.

Usage Notes

The `list` must be a non-empty comma-separated list: Anything other than a comma-separated list is rejected. Commas inside double quotes do not count.

The values in `tab` are cut from the original list, with no transformations.

TABLE_TO_COMMA procedure

This procedure converts a PL/SQL table of names into a comma-separated list of names. This takes a PL/SQL table, `1..n`, terminated with `n+1 null`.

Syntax

```
DBMS_UTILITY.TABLE_TO_COMMA (
   tab    IN  UNCL_ARRAY,
   tablen OUT BINARY_INTEGER,
   list   OUT VARCHAR2);
```

Parameters

Table 53–12 TABLE_TO_COMMA Procedure Parameters

Parameter	Description
`tab`	PL/SQL table which contains list of table names.
`tablen`	Number of tables in the PL/SQL table.
`list`	Comma separated list of tables.

Returns

Returns a comma-separated list and the number of elements found in the table (`n`). Note that ',,,' || ',' || ',,,' = ',,,,,,,'.

PORT_STRING function

This function returns a string that identifies the operating system and the TWO TASK PROTOCOL version of the database. For example, "VAX/VMX-7.1.0.0"

The maximum length is port-specific.

Syntax

```
DBMS_UTILITY.PORT_STRING
   RETURN VARCHAR2;
```

Parameters

None.

Pragmas

```
pragma restrict_references(port_string, WNDS, RNDS, WNPS, RNPS);
```

DB_VERSION procedure

This procedure returns version information for the database.

Syntax

```
DBMS_UTILITY.DB_VERSION (
   version       OUT VARCHAR2,
   compatibility OUT VARCHAR2);
```

Parameters

Table 53–13 DB_VERSION Procedure Parameters

Parameter	Description
version	A string which represents the internal software version of the database (e.g., 7.1.0.0.0). The length of this string is variable and is determined by the database version.
compatibility	The compatibility setting of the database determined by the "compatible" init.ora parameter. If the parameter is not specified in the init.ora file, then NULL is returned.

MAKE_DATA_BLOCK_ADDRESS function

This function creates a data block address given a file number and a block number. A data block address is the internal structure used to identify a block in the database. This function is useful when accessing certain fixed tables that contain data block addresses.

Syntax

```
DBMS_UTILITY.MAKE_DATA_BLOCK_ADDRESS (
   file  NUMBER,
   block NUMBER)
  RETURN NUMBER;
```

Parameters

Table 53–14 MAKE_DATA_BLOCK_ADDRESS Function Parameters

Parameter	Description
file	File that contains the block.
block	Offset of the block within the file in terms of block increments.

Pragmas

```
pragma restrict_references(make_data_block_address, WNDS, RNDS, WNPS, RNPS);
```

Returns

Table 53–15 MAKE_DATA_BLOCK_ADDRESS Function Returns

Returns	Description
dba	Data block address.

DATA_BLOCK_ADDRESS_FILE function

This function gets the file number part of a data block address.

Syntax

```
DBMS_UTILITY.DATA_BLOCK_ADDRESS_FILE (
   dba NUMBER)
  RETURN NUMBER;
```

Parameters

Table 53–16 DATA_BLOCK_ADDRESS_FILE Function Parameters

Parameter	Description
dba	Data block address.

Pragmas

```
pragma restrict_references(data_block_address_file, WNDS, RNDS, WNPS, RNPS);
```

Returns

Table 53–17 DATA_BLOCK_ADDRESS_FILE Function Returns

Returns	Description
file	File that contains the block.

DATA_BLOCK_ADDRESS_BLOCK function

This function gets the block number part of a data block address.

Syntax

```
DBMS_UTILITY.DATA_BLOCK_ADDRESS_BLOCK (
   dba NUMBER)
  RETURN NUMBER;
```

Parameters

Table 53–18 DATA_BLOCK_ADDRESS_BLOCK Function Parameters

Parameter	Description
dba	Data block address.

Pragmas

```
pragma restrict_references(data_block_address_block, WNDS, RNDS, WNPS, RNPS);
```

Returns

Table 53–19 DATA_BLOCK_ADDRESS_BLOCK Function Returns

Returns	Description
block	Block offset of the block.

GET_HASH_VALUE function

This function computes a hash value for the given string.

Syntax

```
DBMS_UTILITY.GET_HASH_VALUE (
   name      VARCHAR2,
   base      NUMBER,
   hash_size NUMBER)
  RETURN NUMBER;
```

Parameters

Table 53–20 GET_HASH_VALUE Function Parameters

Parameter	Description
name	String to be hashed.
base	Base value for the returned hash value to start at.
hash_size	Desired size of the hash table.

Pragmas

```
pragma restrict_references(get_hash_value, WNDS, RNDS, WNPS, RNPS);
```

Returns

A hash value based on the input string. For example, to get a hash value on a string where the hash value should be between 1000 and 3047, use 1000 as the base value and 2048 as the hash_size value. Using a power of 2 for the hash_size parameter works best.

ANALYZE_PART_OBJECT procedure

This procedure is equivalent to SQL:

"ANALYZE TABLE|INDEX [<schema>.]<object_name> PARTITION <pname> [<command_type>] [<command_opt>] [<sample_clause>]

For each partition of the object, run in parallel using job queues.

Syntax

```
DBMS_UTILITY.ANALYZE_PART_OBJECT (
   schema        IN VARCHAR2 DEFAULT NULL,
   object_name   IN VARCHAR2 DEFAULT NULL,
   object_type   IN CHAR     DEFAULT 'T',
   command_type  IN CHAR     DEFAULT 'E',
   command_opt   IN VARCHAR2 DEFAULT NULL,
   sample_clause IN VARCHAR2 DEFAULT 'SAMPLE 5 PERCENT');
```

Parameters

Table 53–21 ANALYZE_PART_OBJECT Procedure Parameters

Parameter	Description
schema	Schema of the object_name.
object_name	Name of object to be analyzed, must be partitioned.
object_type	Type of object, must be T (table) or I (index).
command_type	Must be one of the following: C (compute statistics) E (estimate statistics) D (delete statistics) V (validate structure)
command_opt	Other options for the command type. For C, E it can be FOR table, FOR all LOCAL indexes, FOR all columns or combination of some of the 'for' options of analyze statistics (table). For V, it can be CASCADE when object_type is T.
sample_clause	The sample clause to use when command_type is 'E'.

EXEC_DDL_STATEMENT procedure

This procedure executes the DDL statement in parse_string.

Syntax

```
DBMS_UTILITY.EXEC_DDL_STATEMENT (
   parse_string IN VARCHAR2);
```

Parameters

Table 53–22 EXEC_DDL_STATEMENT Procedure Parameters

Parameter	Description
parse_string	DDL statement to be executed.

CURRENT_INSTANCE function

This function returns the current connected instance number. It returns NULL when connected instance is down.

Syntax

```
DBMS_UTILITY.CURRENT_INSTANCE
   RETURN NUMBER;
```

Parameters

None.

ACTIVE_INSTANCES procedure

Syntax

```
DBMS_UTILITY.ACTIVE_INSTANCE (
   instance_table   OUT INSTANCE_TABLE,
   instance_count   OUT NUMBER);
```

Parameters

Table 53–23 ACTIVE_INSTANCES Procedure Parameters

Procedure	Description
instance_table	Contains a list of the active instance numbers and names. When no instance is up (or non-OPS setting), the list is empty.
instance_count	Number of active instances; 0 under non-OPS setting.

54

DEBUG_EXTPROC

The DEBUG_EXTPROC package enables you to start up the extproc agent within a session. This utility package can help you debug external procedures.

Requirements

Your Oracle account must have EXECUTE privileges on the package and CREATE LIBRARY privileges.

> **Note:** DEBUG_EXTPROC works only on platforms with debuggers that can attach to a running process.

Installation Notes

To install the package, run the script DBGEXTP.SQL.

- Install/load this package in the Oracle USER where you want to debug the 'extproc' process.
- Ensure that you have execute privileges on package DEBUG_EXTPROC

```
SELECT SUBSTR(OBJECT_NAME, 1, 20)
FROM USER_OBJECTS
WHERE OBJECT_NAME = 'DEBUG_EXTPROC';
```

- You can install this package as any other user, as long as you have EXECUTE privileges on the package.

Using DEBUG_EXTPROC

Usage Assumptions

This assumes that the Listener has been appropriately configured to startup an external procedures 'extproc' agent.

This also assumes that you built your shared library with debug symbols to aid in the debugging process. Please check the C compiler manual pages for the appropriate C compiler switches to build the shared library with debug symbols.

Usage Notes

- Start a brand new oracle session through SQL*Plus or OCI program by connecting to ORACLE.
- Execute procedure DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT to startup the extproc agent in this session; e.g., execute DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT; Do not exit this session, because that terminates the extproc agent.

- Determine the PID of the extproc agent that was started up for this session.
- Using a debugger (e.g., gdb, dbx, or the native system debugger), load the extproc executable and attach to the running process.
- Set a breakpoint on function 'pextproc' and let the debugger continue with its execution.
- Now execute your external procedure in the same session where you first executed DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT
- Your debugger should now break in function 'pextproc'. At this point in time, the shared library referenced by your PL/SQL external function would have been loaded and the function resolved. Now set a breakpoint in your C function and let the debugger continue its execution.

Because PL/SQL loads the shared library at runtime, the debugger you use may or may not automatically be able to track the new symbols from the shared library. You may have to issue some debugger command to load the symbols (for example, 'share' in gdb)

- The debugger should now break in your C function. Its assumed that you had built the shared library with debugging symbols.
- Now proceed with your debugging.

Summary of Subprograms

DEBUG_EXTPROC contains one subprogram: STARTUP_EXTPROC_AGENT procedure. This starts up the extproc agent process in the session

STARTUP_EXTPROC_AGENT procedure

This procedure starts up the extproc agent process in the session.This enables you to get the PID of the executing process. This PID is needed to be able to attach to the running process using a debugger.

Syntax

```
DEBUG_EXTPROC.STARTUP_EXTPROC_AGENT;
```

Parameters

None.

55

OUTLN_PKG

The OUTLN_PKG package contains the functional interface for subprograms associated with the management of stored outlines.

A stored outline is the stored data that pertains to an execution plan for a given SQL statement. It enables the optimizer to repeatedly recreate execution plans that are equivalent to the plan originally generated along with the outline.The data stored in an outline consists, in part, of a set of hints that are used to achieve plan stability.

Requirements

OUTLN_PKG contains management procedures that should only be made available to appropriate users. Execute privilege is not extended to the general user community unless the DBA explicitly does so.

Security

PL/SQL functions that are available for outline management purposes can be executed only by users with EXECUTE privilege on the procedure (or package).

Summary of Subprograms

Table 55–1 OUTLN_PKG Package Subprograms

Subprogram	Description
DROP_UNUSED procedure on page 55-2	Drops all outlines that have not been used since they were created.
DROP_BY_CAT procedure on page 55-3	Drops all outlines that belong to a particular category.
UPDATE_BY_CAT procedure on page 55-3	Changes the category of all outlines in one category to a new category.

DROP_UNUSED procedure

This procedure drops outlines that have never been applied in the compilation of a SQL statement.

Syntax

```
OUTLN_PKG.DROP_UNUSED;
```

Parameters

None.

Usage Notes

You may want to use DROP_UNUSED for outlines created on behalf of dynamic SQL statements, that were generated by an application for one time use only. For these statements, the outlines are never used and are simply taking up valuable disk space.

DROP_BY_CAT procedure

This procedure drops outlines that belong to a particular category.

Syntax

```
OUTLN_PKG.DROP_BY_CAT (
   cat VARCHAR2);
```

Parameters

Table 55–2 DROP_BY_CAT Procedure Parameters

Parameter	Description
cat	Category of outlines to drop.

Usage Notes

You may occasionally want to purge a category of outlines. This procedure accomplishes that in a single call.

Example

This example drops all outlines in the DEFAULT category:

```
OUTLN_PKG.DROP_BY_CAT('DEFAULT');
```

UPDATE_BY_CAT procedure

This procedure changes the category of all outlines in one category to a new category. If the SQL text in an outline already has an outline in the target category, then it is not merged into the new category.

Syntax

```
OUTLN_PKG.UPDATE_BY_CAT (
   oldcat VARCHAR2 DEFAULT 'DEFAULT',
   newcat VARCHAR2 DEFAULT 'DEFAULT');
```

Parameters

Table 55–3 UPDATE_BY_CAT Procedure Parameters

Parameter	Description
oldcat	Current category to be changed.
newcat	Target category to change outline to.

Usage Notes

Once satisfied with a set of outlines, you might chose to move outlines from an *experimental* category to a *production* category. Likewise, you might want to merge a set of outlines from one category into another pre-existing category.

Example

This example changes all outlines in the DEFAULT category to the CAT1 category:

```
OUTLN_PKG.UPDATE_BY_CAT('DEFAULT', 'CAT1');
```

56

UTL_COLL

The UTL_COLL package lets PL/SQL programs use collection locators to query and update.

Summary of Subprograms

There is currently only one function supported in this package:IS_LOCATOR.

IS_LOCATOR function

This function determines whether a collection item is actually a locator or not.

Syntax

```
UTL_COLL.IS_LOCATOR (
   collection IN ANY)
  RETURNS BOOLEAN;
```

Parameters

Table 56–1 IS_LOCATOR Function Parameters

Parameter	Description
collection	Nested table or varray item.

Returns

Table 56–2 IS_LOCATOR Function Returns

Return Value	Description
1	Collection item is indeed a locator.
0	Collection item is not a locator.

Pragmas

Asserts WNDS, WNPS and RNPS pragmas

Exceptions

None.

Example

```
CREATE OR REPLACE TYPE list_t as TABLE OF VARCHAR2(20);
/

CREATE OR REPLACE TYPE phone_book_t AS OBJECT (
  pno  number,
  ph   list_t );
/

CREATE TABLE phone_book OF phone_book_t
      NESTED TABLE ph STORE AS nt_ph;
CREATE TABLE phone_book1 OF phone_book_t
      NESTED TABLE ph STORE AS nt_ph_1 RETURN LOCATOR;

INSERT INTO phone_book VALUES(1, list_t('650-633-5707','650-323-0953'));
INSERT INTO phone_book1 VALUES(1, list_t('415-555-1212'));

CREATE OR REPLACE PROCEDURE chk_coll IS
  plist list_t;
  plist1 list_t;
BEGIN
  SELECT ph INTO plist FROM phone_book WHERE pno=1;

  SELECT ph INTO plist1 FROM phone_book1 WHERE pno=1;

  IF (UTL_COLL.IS_LOCATOR(plist)) THEN
    DBMS_OUTPUT.PUT_LINE('plist is a locator');
  ELSE
    DBMS_OUTPUT.PUT_LINE('plist is not a locator');
  END IF;

  IF (UTL_COLL.IS_LOCATOR(plist1)) THEN
    DBMS_OUTPUT.PUT_LINE('plist1 is a locator');
  ELSE
    DBMS_OUTPUT.PUT_LINE('plist1 is not a locator');
  END IF;

END chk_coll;

SET SERVEROUTPUT ON
EXECUTE chk_coll;
```

57

UTL_FILE

The UTL_FILE package lets your PL/SQL programs read and write operating system (OS) text files. It provides a restricted version of standard OS stream file input/output (I/O).

The file I/O capabilities are similar to those of the standard operating system stream file I/O (OPEN, GET, PUT, CLOSE), with some limitations. For example, call the FOPEN function to return a *file handle*, which you then use in subsequent calls to GET_LINE or PUT to perform stream I/O to a file. When you are done performing I/O on the file, call FCLOSE to complete any output and to free any resources associated with the file.

Security

The PL/SQL file I/O feature is available for both client side and server side PL/SQL. The client implementation (text I/O) is subject to normal operating system file permission checking, and it does not need any additional security constraints. However, the server implementation might be running in a privileged mode, and will need additional security restrictions that limit the power of this feature.

Note: The UTL_FILE package is similar to the client-side TEXT_IO package currently provided by Oracle Procedure Builder. Restrictions for a server implementation require some API differences between UTL_FILE and TEXT_IO. In PL/SQL file I/O, errors are returned to you using PL/SQL exceptions.

Server Security

Server security for PL/SQL file I/O consists of a restriction on the directories that can be accessed. Accessible directories must be specified in the instance parameter initialization file (INIT.ORA).

Specify the accessible directories for the UTL_FILE functions in the initialization file using the UTL_FILE_DIR parameter. For example:

```
UTL_FILE_DIR = <directory name>
```

Note: The directory specification is different on different platforms.

If the initialization file for the instance contains the line *UTL_FILE_DIR = /usr/jsmith/my_app*, then the directory */usr/jsmith/my_app* is accessible to the FOPEN function. Note that a directory named */usr/jsmith/My_App* would not be accessible on case-sensitive operating systems.

The parameter specification UTL_FILE_DIR = * has a special meaning. This entry turns off directory access checking, and it makes any directory accessible to the UTL_FILE functions.

Caution:

The '*' option should be used with great caution. Oracle does not recommend that you use this option in production systems. Also, do not include '.' (the current directory for UNIX) in the accessible directories list.

To ensure security on file systems that enable symbolic links, users must not be allowed WRITE permission to directories accessible by PL/SQL file I/O functions. The symbolic links and PL/SQL file I/O could be used to circumvent normal operating system permission checking and allow users read/write access to directories to which they would not otherwise have access.

File Ownership and Protections

On UNIX systems, a file created by the FOPEN function has as its owner the owner of the shadow process running the instance. In the normal case, this owner is *oracle*. Files created using FOPEN are always writable and readable using the UTL_FILE subprograms, but non-privileged users who need to read these files outside of PL/SQL might nned their system administrator to give them access.

Examples (UNIX-Specific)

If the parameter initialization file contains only:

```
UTL_FILE_DIR=/appl/gl/log
UTL_FILE_DIR=/appl/gl/out
```

Then, the following file locations and filenames are valid:

```
FILE LOCATION            FILENAME
/appl/gl/log             L10324.log
/appl/gl/out             O10324.out
```

But, the following file locations and filename are invalid:

```
FILE LOCATION            FILENAME
/appl/gl/log/backup      L10324.log          # subdirectory
/APPL/gl/log             L10324.log          # uppercase
/appl/gl/log             backup/L10324.log   # dir in name
/usr/tmp                 T10324.tmp          # not in INIT.ORA
```

Caution: There are no user-level file permissions. All file locations specified by the UTL_FILE_DIR parameters are valid, for both reading and writing, for all users of the file I/O procedures. This can override operating system file permissions.

Types

```
TYPE file_type IS RECORD (id BINARY_INTEGER);
```

The contents of FILE_TYPE are private to the UTL_FILE package. Users of the package should not reference or change components of this record.

Exceptions

Table 57–1 UTL_FILE Package Exceptions

Exception Name	Description
INVALID_PATH	File location or filename was invalid.
INVALID_MODE	The open_mode parameter in FOPEN was invalid.
INVALID_FILEHANDLE	File handle was invalid.
INVALID_OPERATION	File could not be opened or operated on as requested.
READ_ERROR	Operating system error occurred during the read operation.
WRITE_ERROR	Operating system error occurred during the write operation.
INTERNAL_ERROR	Unspecified PL/SQL error.

In addition to these package exceptions, procedures in UTL_FILE can also raise predefined PL/SQL exceptions such as NO_DATA_FOUND or VALUE_ERROR.

Summary of Subprograms

Table 57–2 UTL_FILE Subprograms

Subprogram	Description
FOPEN function on page 57-6	Opens a file for input or output with the default line size.
IS_OPEN function on page 57-7	Determines if a file handle refers to an open file.
FCLOSE procedure on page 57-8	Closes a file.
FCLOSE_ALL procedure on page 57-8	Closes all open file handles.
GET_LINE procedure on page 57-9	Reads a line of text from an open file.
PUT procedure on page 57-10	Writes a line to a file. This does not append a line terminator.
NEW_LINE procedure on page 57-11	Writes one or more OS-specific line terminators to a file.
PUT_LINE procedure on page 57-12	Writes a line to a file. This appends an OS-specific line terminator.
PUTF procedure on page 57-12	A PUT procedure with formatting.
FFLUSH procedure on page 57-14	Physically writes all pending output to a file.
FOPEN function on page 57-14	Opens a file with the maximum line size specified.

FOPEN function

This function opens a file for input or output. The file location must be an accessible directory, as defined in the instance's initialization parameter UTL_FILE_DIR. The complete directory path must already exist; it is not created by FOPEN.

FOPEN returns a file handle, which must be used in all subsequent I/O operations on the file.

This version of FOPEN does not take a parameter for the maximum line size. Thus, the default (which is 1023 on most systems) is used. To specify a different maximum line size, use the other, overloaded version of "FOPEN function" on page 57-14.

You can have a maximum of 50 files open simultaneously.

Syntax

```
UTL_FILE.FOPEN (
   location  IN VARCHAR2,
   filename  IN VARCHAR2,
   open_mode IN VARCHAR2)
  RETURN UTL_FILE.FILE_TYPE;
```

Parameters

Table 57–3 FOPEN Function Parameters

Parameters	Description
location	Operating system-specific string that specifies the directory in which to open the file.
filename	Name of the file, including extension (file type), without any directory path information. (Under the UNIX operating system, the filename cannot be terminated with a '/'.).
open_mode	String that specifies how the fie is to be opened (either upper or lower case letters can be used). The supported values, and the UTL_FILE procedures that can be used with them are: 'r' read text (GET_LINE) 'w' write text (PUT, PUT_LINE, NEW_LINE, PUTF, FFLUSH) 'a' append text (PUT, PUT_LINE, NEW_LINE, PUTF, FFLUSH)

Note: If you open a file that does not exist using the 'a' value for open_mode, then the file is created in write ('w') mode.

Returns

FOPEN returns a file handle, which must be passed to all subsequent procedures that operate on that file. The specific contents of the file handle are private to the UTL_FILE package, and individual components should not be referenced or changed by the UTL_FILE user.

Note: The file location and file name parameters are supplied to the FOPEN function as separate strings, so that the file location can be checked against the list of accessible directories as specified in the initialization file. Together, the file location and name must represent a legal filename on the system, and the directory must be accessible. A subdirectory of an accessible directory is not necessarily also accessible; it too must be specified using a complete path name in the initialization file.

Operating system-specific parameters, such as C-shell environment variables under UNIX, cannot be used in the file location or file name parameters.

Exceptions

INVALID_PATH
INVALID_MODE
INVALID_OPERATION

IS_OPEN function

This function tests a file handle to see if it identifies an open file. IS_OPEN reports only whether a file handle represents a file that has been opened, but not yet closed. It does not guarantee that there will be no operating system errors when you attempt to use the file handle.

Syntax

```
UTL_FILE.IS_OPEN (
   file  IN FILE_TYPE)
  RETURN BOOLEAN;
```

Parameters

Table 57–4 IS_OPEN Function Parameters

Parameter	Description
file	Active file handle returned by an FOPEN call.

Returns

TRUE or FALSE

Exceptions

None.

FCLOSE procedure

This procedure closes an open file identified by a file handle. If there is buffered data yet to be written when FCLOSE runs, then you may receive a WRITE_ERROR exception when closing a file.

Syntax

```
UTL_FILE.FCLOSE (
   file IN OUT FILE_TYPE);
```

Parameters

Table 57–5 FCLOSE Procedure Parameters

Parameter	Description
file	Active file handle returned by an FOPEN call.

Exceptions

WRITE_ERROR
INVALID_FILEHANDLE

FCLOSE_ALL procedure

This procedure closes all open file handles for the session. This should be used as an emergency cleanup procedure, for example, when a PL/SQL program exits on an exception.

Note: FCLOSE_ALL does not alter the state of the open file handles held by the user. This means that an IS_OPEN test on a file handle after an FCLOSE_ALL call still returns TRUE, even though the file has been closed. No further read or write operations can be performed on a file that was open before an FCLOSE_ALL.

Syntax

```
UTL_FILE.FCLOSE_ALL;
```

Parameters

None.

Exceptions

WRITE_ERROR

GET_LINE procedure

This procedure reads a line of text from the open file identified by the file handle and places the text in the output buffer parameter. Text is read up to but not including the line terminator, or up to the end of the file.

If the line does not fit in the buffer, then a VALUE_ERROR exception is raised. If no text was read due to "end of file," then the NO_DATA_FOUND exception is raised.

Because the line terminator character is not read into the buffer, reading blank lines returns empty strings.

The maximum size of an input record is 1023 bytes, unless you specify a larger size in the overloaded version of FOPEN.

Syntax

```
UTL_FILE.GET_LINE (
   file        IN  FILE_TYPE,
   buffer      OUT VARCHAR2);
```

Parameters

Table 57–6 GET_LINE Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN call. The file must be open for reading (mode 'r'), otherwise an INVALID_OPERATION exception is raised.
buffer	Data buffer to receive the line read from the file.

Exceptions

INVALID_FILEHANDLE
INVALID_OPERATION
READ_ERROR
NO_DATA_FOUND
VALUE_ERROR

PUT procedure

PUT writes the text string stored in the buffer parameter to the open file identified by the file handle. The file must be open for write operations. No line terminator is appended by PUT; use NEW_LINE to terminate the line or use PUT_LINE to write a complete line with a line terminator.

The maximum size of an input record is 1023 bytes, unless you specify a larger size in the overloaded version of FOPEN.

Syntax

```
UTL_FILE.PUT (
   file      IN FILE_TYPE,
   buffer    IN VARCHAR2);
```

Parameters

Table 57–7 PUT Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN call.
buffer	Buffer that contains the text to be written to the file. You must have opened the file using mode 'w' or mode 'a'; otherwise, an INVALID_OPERATION exception is raised.

Exceptions

INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR

NEW_LINE procedure

This procedure writes one or more line terminators to the file identified by the input file handle. This procedure is separate from PUT because the line terminator is a platform-specific character or sequence of characters.

Syntax

```
UTL_FILE.NEW_LINE (
   file     IN FILE_TYPE,
   lines    IN NATURAL := 1);
```

Parameters

Table 57–8 NEW_LINE Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN call.
lines	Number of line terminators to be written to the file.

Exceptions

INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR

PUT_LINE procedure

This procedure writes the text string stored in the buffer parameter to the open file identified by the file handle. The file must be open for write operations. PUT_LINE terminates the line with the platform-specific line terminator character or characters.

The maximum size for an output record is 1023 bytes, unless you specify a larger value using the overloaded version of FOPEN.

Syntax

```
UTL_FILE.PUT_LINE (
   file    IN FILE_TYPE,
   buffer  IN VARCHAR2);
```

Parameters

Table 57–9 PUT_LINE Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN call.
buffer	Text buffer that contains the lines to be written to the file.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

PUTF procedure

This procedure is a formatted PUT procedure. It works like a limited printf(). The format string can contain any text, but the character sequences '%s' and '\n' have special meaning.

%s	Substitute this sequence with the string value of the next argument in the argument list.
\n	Substitute with the appropriate platform-specific line terminator.

Syntax

```
UTL_FILE.PUTF (
   file    IN FILE_TYPE,
   format  IN VARCHAR2,
   [arg1   IN VARCHAR2  DEFAULT NULL,
   . . .
   arg5    IN VARCHAR2  DEFAULT NULL]);
```

Parameters

Table 57–10 PUTF Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN call.
format	Format string that can contain text as well as the formatting characters '\n' and '%s'.
arg1..arg5	From one to five operational argument strings. Argument strings are substituted, in order, for the '%s' formatters in the format string. If there are more formatters in the format parameter string than there are arguments, then an empty string is substituted for each '%s' for which there is no argument.

Example

The following example writes the lines:

```
Hello, world!
I come from Zork with greetings for all earthlings.

my_world  varchar2(4) := 'Zork';
...
PUTF(my_handle, 'Hello, world!\nI come from %s with %s.\n',
                my_world,
                'greetings for all earthlings');
```

If there are more %S formatters in the format parameter than there are arguments, then an empty string is substituted for each %S for which there is no matching argument.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

FFLUSH procedure

FFLUSH physically writes all pending data to the file identified by the file handle. Normally, data being written to a file is buffered. The FFLUSH procedure forces any buffered data to be written to the file.

Flushing is useful when the file must be read while still open. For example, debugging messages can be flushed to the file so that they can be read immediately.

Syntax

```
UTL_FILE.FFLUSH (
   file  IN FILE_TYPE);
invalid_maxlinesize  EXCEPTION;
```

Parameters

Table 57–11 FFLUSH Procedure Parameters

Parameters	Description
file	Active file handle returned by an FOPEN call.

Exceptions

```
INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR
```

FOPEN function

This function opens a file. You can have a maximum of 50 files open simultaneously.

Note: This version of FOPEN enables you to specify the desired maximum line size. The other version of the "FOPEN function" on page 57-6 uses the default line size.

Syntax

```
UTL_FILE.FOPEN (
   location     IN VARCHAR2,
   filename     IN VARCHAR2,
   open_mode    IN VARCHAR2,
   max_linesize IN BINARY_INTEGER)
  RETURN file_type;
```

Parameters

Table 57–12 FOPEN Function Parameters

Parameter	Description
location	Directory location of file.
filename	File name (including extension).
open_mode	Open mode ('r', 'w', 'a').
max_linesize	Maximum number of characters per line, including the newline character, for this file. (minimum value 1, maximum value 32767).

Returns

Table 57–13 FOPEN Function Returns

Return	Description
file_type	Handle to open file.

Exceptions

INVALID_PATH: File location or name was invalid.
INVALID_MODE: The open_mode string was invalid.
INVALID_OPERATION: File could not be opened as requested.
INVALID_MAXLINESIZE: Specified max_linesize is too large or too small.

58

UTL_HTTP

UTL_HTTP makes hyper-text transfer protocol (HTTP) callouts from PL/SQL and SQL. You can use it to access data on the Internet or to call Oracle Web Server cartridges.

UTL_HTTP contains two similar entrypoints: REQUEST and REQUEST_PIECES. They each take a string universal resource locator (URL), contact that site, and return the data (typically HTML — hyper-text markup language) obtained from that site.

Exceptions

Table 58–1 UTL_HTTP Package Exceptions

Exception	Description
INIT_FAILED	Initialization of the HTTP-callout subsystem fails (for environmental reasons, such as lack of available memory).
REQUEST_FAILED	The HTTP call fails (for example, because of failure of the HTTP daemon, or because the argument to REQUEST or REQUEST_PIECES cannot be interpreted as a URL because it is NULL or has non-HTTP syntax).

The above two exceptions, unless explicitly caught by an exception handler, are reported by this generic message: ORA-06510: PL/SQL: unhandled user-defined exception. This reports them as "user-defined" exceptions, although they are defined in this system package.

If any other exception is raised during the processing of the HTTP request (for example, an out-of-memory error), then function REQUEST or REQUEST_PIECES reraises that exception.

When no response is received from a request to the given URL (for example, because no site corresponding to that URL is contacted), then a formatted HTML error message may be returned. For example:

```
<HTML>
<HEAD>
<TITLE>Error Message</TITLE>
</HEAD>
<BODY>
<H1>Fatal Error 500</H1>
Can't Access Document:  http://home.nothing.comm.
<P>
<B>Reason:</B> Can't locate remote host:  home.nothing.comm.
<P>

<P><HR>
<ADDRESS><A HREF="http://www.w3.org">
CERN-HTTPD3.0A</A></ADDRESS>
</BODY>
</HTML>
```

Usage Notes

You should not expect REQUEST or REQUEST_PIECES to succeed in contacting a URL unless you can contact that URL by using a browser on the same machine (and with the same privileges, environment variables, etc.)

If REQUEST or REQUEST_PIECES fails (for example, if it raises an exception, or if it returns an HTML-formatted error message, yet you believe that the URL argument is correct), then try contacting that same URL with a browser, to verify network availability from your machine. Remember that you may have a proxy server set in your browser that needs to be set with each REQUEST or REQUEST_PIECES call using the optional proxy parameter.

Note: UTL_HTTP can also use environment variables to specify its proxy behavior. For example, on UNIX, setting the environment variable http_proxy to a URL specifies to use that service as the proxy server for HTTP requests. Setting the environment variable no_proxy to a domain name specifies to *not* use the HTTP proxy server for URL's in that domain.

Summary of Subprograms

Table 58–2 UTL_HTTP Package Subprograms

Subprogram	Description
REQUEST function on page 58-3	Returns up to the first 2000 bytes of the data retrieved from the given URL.
REQUEST_PIECES function on page 58-5	Returns a PL/SQL-table of 2000-byte pieces of the data retrieved from the given URL.

REQUEST function

This function returns up to the first 2000 bytes of the data retrieved from the given URL.

Syntax

```
UTL_HTTP.REQUEST (
   url   IN VARCHAR2,
   proxy IN VARCHAR2 DEFAULT NULL)
  RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references (request, wnds, rnds, wnps, rnps);
```

Parameters

Table 58–3 REQUEST Function Parameters

Parameter	Description
url	Universal resource locator.
proxy	(Optional) Specifies a proxy server to use when making the HTTP request.

Returns

Its return-type is a string of length 2000 or less, which contains up to the first 2000 bytes of the HTML result returned from the HTTP request to the argument URL.

Exceptions

```
INIT_FAILED
REQUEST_FAILED
```

Example

```
SVRMGR> SELECT utl_http.request('http://www.oracle.com/') FROM dual;
UTL_HTTP.REQUEST('HTTP://WWW.ORACLE.COM/')
<html>
<head><title>Oracle Corporation Home Page</title>
<!--changed Jan. 16, 19
1 row selected.
```

If you are behind a firewall, include the `proxy` parameter. For example, from within the Oracle firewall, where there might be a proxy server named `www-proxy.us.oracle.com`:

```
SVRMGR> SELECT
utl_http.request('http://www.oracle.com', 'www-proxy.us.oracle.com') FROM dual;
```

REQUEST_PIECES function

This function returns a PL/SQL table of 2000-byte pieces of the data retrieved from the given URL.

Syntax

```
type html_pieces is table of varchar2(2000) index by binary_integer;

UTL_HTTP.REQUEST_PIECES (
   url        IN VARCHAR2,
   max_pieces NATURAL     DEFAULT 32767,
   proxy      IN VARCHAR2 DEFAULT NULL)
  RETURN HTML_PIECES;
```

Pragmas

```
pragma restrict_references (request_pieces, wnds, rnds, wnps, rnps);
```

Parameters

Table 58–4 REQUEST_PIECES Function Parameters

Parameter	Description
url	Universal resource locator.
max_pieces	(Optional) The maximum number of pieces (each 2000 characters in length, except for the last, which may be shorter), that REQUEST_PIECES should return. If provided, then that argument should be a positive integer.
proxy	(Optional) Specifies a proxy server to use when making the HTTP request.

Returns

REQUEST_PIECES returns a PL/SQL table of type UTL_HTTP.HTML_PIECES. Each element of that PL/SQL table is a string of length 2000. The final element may be shorter than 2000 characters.

The elements of the PL/SQL table returned by REQUEST_PIECES are successive pieces of the data obtained from the HTTP request to that URL.

Exceptions

INIT_FAILED
REQUEST_FAILED

Example

A call to REQUEST_PIECES could look like the example below. Note the use of the PL/SQL table method COUNT to discover the number of pieces returned, which may be zero or more:

```
DECLARE pieces utl_http.html_pieces;
BEGIN
  pieces := utl_http.request_pieces('http://www.oracle.com/');
  FOR i in 1 .. pieces.count loop
    .... -- process each piece
  END LOOP;
END;
```

Example

The following block retrieves up to 100 pieces of data (each 2000 bytes, except perhaps the last) from the URL. It prints the number of pieces retrieved and the total length, in bytes, of the data retrieved.

```
SET SERVEROUTPUT ON
/
DECLARE
  x utl_http.html_pieces;
BEGIN
  x := utl_http.request_pieces('http://www.oracle.com/', 100);
  dbms_output.put_line(x.count || ' pieces were retrieved.');
  dbms_output.put_line('with total length ');
  IF x.count < 1
  THEN dbms_output.put_line('0');
  ELSE dbms_output.put_line
  ((2000 * (x.count - 1)) + length(x(x.count)));
  END IF;
END;
/
-- Output
Statement processed.
4 pieces were retrieved.
with total length
7687
```

59

UTL_RAW

The UTL_RAW package provides SQL functions for manipulating RAW datatypes. This package is necessary because normal SQL functions do not operate on RAWs, and PL/SQL does not allow overloading between a RAW and a CHAR datatype. UTL_RAW also includes subprograms that convert various COBOL number formats to, and from, RAWs.

UTL_RAW is not specific to the database environment, and it may actually be used in other environments as it exists here. For this reason, the prefix UTL has been given to the package, instead of DBMS.

Usage Notes

The are many possible uses for the RAW functions. UTL_RAW allows a RAW "record" to be composed of many elements. By using the RAW datatype, character set conversion will not be performed keeping the RAW in its original format when being transferred through remote procedure calls (RPC).

The RAW functions also provide the ability to manipulate binary data which was previously limited to the `hextoraw` and `rawtohex` functions.

Summary of Subprograms

Table 59–1 UTL_RAW Package Subprograms

Subprogram	Description
CONCAT function on page 59-3	Concatenates up to 12 RAWs into a single RAW.
CAST_TO_RAW function on page 59-4	Converts a VARCHAR2 represented using `n` data bytes into a RAW with `n` data bytes.
CAST_TO_VARCHAR2 function on page 59-5	Converts a RAW represented using `n` data bytes into VARCHAR2 with `n` data bytes.
LENGTH function on page 59-6	Returns the length in bytes of a RAW `r`.
SUBSTR function on page 59-7	Returns `len` bytes, starting at `pos` from RAW `r`.
TRANSLATE function on page 59-8	Translates the bytes in the input RAW `r` according to the bytes in the translation RAWs `from_set` and `to_set`.
TRANSLITERATE function on page 59-10	Converts the bytes in the input RAW `r` according to the bytes in the transliteration RAWs `from_set` and `to_set`.
OVERLAY function on page 59-11	Overlays the specified portion of target RAW with overlay RAW, starting from byte position `pos` of target and proceding for `len` bytes.
COPIES function on page 59-13	Returns `n` copies of `r` concatenated together.
XRANGE function on page 59-14	Returns a RAW containing all valid 1-byte encodings in succession, beginning with the value `start_byte` and ending with the value `end_byte`.

Table 59–1 UTL_RAW Package Subprograms

Subprogram	Description
REVERSE function on page 59-15	Reverses a byte sequence in RAW r from end to end.
COMPARE function on page 59-16	Compares RAW r1 against RAW r2.
CONVERT function on page 59-17	Converts RAW r from character set from_charset to character set to_charset and returns the resulting RAW.
BIT_AND function on page 59-19	Performs bitwise logical "and" of the values in RAW r1 with RAW r2 and returns the "anded" result RAW.
BIT_OR function on page 59-20	Performs bitwise logical "or" of the values in RAW r1 with RAW r2 and returns the "or'd" result RAW.
BIT_XOR function on page 59-21	Performs bitwise logical "exclusive or" of the values in RAW r1 with RAW r2 and returns the "xor'd" result RAW.
BIT_COMPLEMENT function on page 59-22	Performs bitwise logical "complement" of the values in RAW r and returns the "complement'ed" result RAW.

CONCAT function

This function concatenates up to 12 RAWs into a single RAW. If the concatenated size exceeds 32K, then an error is returned

Syntax

```
UTL_RAW.CONCAT (
   r1  IN RAW DEFAULT NULL,
   r2  IN RAW DEFAULT NULL,
   r3  IN RAW DEFAULT NULL,
   r4  IN RAW DEFAULT NULL,
   r5  IN RAW DEFAULT NULL,
   r6  IN RAW DEFAULT NULL,
   r7  IN RAW DEFAULT NULL,
   r8  IN RAW DEFAULT NULL,
   r9  IN RAW DEFAULT NULL,
   r10 IN RAW DEFAULT NULL,
   r11 IN RAW DEFAULT NULL,
   r12 IN RAW DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(concat, WNDS, RNDS, WNPS, RNPS);
```

Parameters

r1....r12 are the RAW items to concatenate.

Returns

Table 59–2 CONCAT Function Returns

Return	Description
RAW	Containing the items concatenated.

Errors

There is an error if the sum of the lengths of the inputs exceeds the maximum allowable length for a RAW, which is 32767 bytes.

CAST_TO_RAW function

This function converts a VARCHAR2 represented using n data bytes into a RAW with n data bytes. The data is not modified in any way, only its datatype is recast to a RAW datatype.

Syntax

```
UTL_RAW.CAST_TO_RAW (
   c IN VARCHAR2)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(cast_to_raw, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–3 CAST_TO_RAW Function Parameters

Parameter	Description
c	VARCHAR2 to be changed to a RAW.

Returns

Table 59–4 CAST_TO_RAW Function Returns

Return	Description
RAW	Containing the same data as the input VARCHAR2 and equal byte length as the input VARCHAR2 and without a leading length field.
NULL	If c input parameter was NULL.

Errors

None.

CAST_TO_VARCHAR2 function

This function converts a RAW represented using n data bytes into VARCHAR2 with n data bytes.

> **Note:** When casting to a VARCHAR2, the current NLS character set is used for the characters within that VARCHAR2.

Syntax

```
UTL_RAW.CAST_TO_VARCHAR2 (
   r IN RAW)
  RETURN VARCHAR2;
```

Pragmas

```
pragma restrict_references(cast_to_varchar2, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–5 CAST_TO_VARCHAR2 Function Parameters

Parameter	Description
r	RAW (without leading length field) to be changed to a VARCHAR2).

Returns

Table 59–6 CAST_TO_VARCHAR2 Function Returns

Return	Description
VARCHAR2	Containing having the same data as the input RAW.
NULL	If r input parameter was NULL.

Errors

None.

LENGTH function

This function returns the length in bytes of a RAW r.

Syntax

```
UTL_RAW.LENGTH (
   r IN RAW)
  RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(length, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–7 LENGTH Function Parameters

Parameter	Description
r	The RAW byte stream to be measured.

Returns

Table 59–8 LENGTH Function Returns

Return	Description
NUMBER	Equal to the current length of the RAW.

Errors

None.

SUBSTR function

This function returns len bytes, starting at pos from RAW r.

Syntax

```
UTL_RAW.SUBSTR (
   r   IN RAW,
   pos IN BINARY_INTEGER,
   len IN BINARY_INTEGER DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(substr, WNDS, RNDS, WNPS, RNPS);
```

Parameters

If pos is positive, then SUBSTR counts from the beginning of r to find the first byte. If pos is negative, then SUBSTR counts backwards from the end of the r. The value pos cannot be 0.

If len is omitted, then SUBSTR returns all bytes to the end of r. The value len cannot be less than 1.

Table 59–9 SUBSTR Function Parameters

Parameter	Description
r	The RAW byte-string from which a portion is extracted.
pos	The byte position in r at which to begin extraction.
len	The number of bytes from pos to extract from r (optional).

Defaults and Optional Parameters

Table 59–10 SUBSTR Function Exceptions

Optional Parameter	Description
len	Position pos through to the end of r.

Returns

Table 59–11 SUBSTR Function Returns

Return	Description
portion of r	Beginning at pos for len bytes long.
NULL	R input parameter was NULL.

Errors

Table 59–12 SUBSTR Function Errors

Error	Description
VALUE_ERROR	Either pos = 0 or len < 0

TRANSLATE function

This function translates the bytes in the input RAW r according to the bytes in the translation RAWs from_set and to_set. If a byte in r has a matching byte in from_set, then it is replaced by the byte in the corresponding position in to_set, or deleted.

Bytes in r, but undefined in from_set, are copied to the result. Only the first (leftmost) occurrence of a byte in from_set is used. Subsequent duplicates are not scanned and are ignored. If to_set is shorter than from_set, then the extra from_set bytes have no translation correspondence and any bytes in r matching.

Note: Difference from TRANSLITERATE:

- Translation RAWs have no defaults.

- r bytes undefined in the to_set translation RAW are deleted.

- Result RAW may be shorter than input RAW r.

Syntax

```
UTL_RAW.TRANSLATE (
   r        IN RAW,
   from_set IN RAW,
   to_set   IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(translate, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–13 TRANSLATE Function Parameters

Parameter	Description
r	RAW source byte-string to be translated.
from_set	RAW byte-codes to be translated, if present in r.
to_set	RAW byte-codes to which corresponding from_str bytes are translated.

Returns

Table 59–14 TRANSLATE Function Returns

Return	Description
RAW	Translated byte-string.

Errors

Table 59–15 TRANSLATE Function Errors

Error	Description
VALUE_ERROR	Either: - r is NULL and / or has 0 length - from_set is NULL and / or has 0 length - to_set is NULL and / or has 0 length

TRANSLITERATE function

This function converts the bytes in the input RAW r according to the bytes in the transliteration RAWs from_set and to_set. Successive bytes in r are looked-up in the from_set, and, if not found, copied unaltered to the result RAW. If found, then they are replaced in the result RAW by either corresponding bytes in the to_set, or the pad byte when no correspondence exists.

Bytes in r, but undefined in from_set, are copied to the result. Only the first (leftmost) occurrence of a byte in from_set is used. Subsequent duplicates are not scanned and are ignored. The result RAW is always the same length as r.

If the to_set is shorter than the from_set, then the pad byte is placed in the result RAW when a selected from_set byte has no corresponding to_set byte (as if the to_set were extended to the same length as the from_set with pad bytes).

Note: Difference from TRANSLATE:

- r bytes undefined in to_set are padded.

- Result RAW is always same length as input RAW r.

Syntax

```
UTL_RAW.TRANSLITERATE (
   r        IN RAW,
   to_set   IN RAW DEFAULT NULL,
   from_set IN RAW DEFAULT NULL,
   pad      IN RAW DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(transliterate, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–16 TRANSLITERATE Function Parameters

Parameter	Description
r	RAW input byte-string to be converted.
from_set	RAW byte-codes to be converted, if present in r (any length).

Table 59–16 TRANSLITERATE Function Parameters

Parameter	Description
to_set	RAW byte-codes to which corresponding from_set bytes are converted (any length).
pad	1 byte used when to-set is shorter than the from_set.

Defaults and Optional Parameters

Table 59–17 TRANSLITERATE Function Optional Parameters

Optional Parameter	Description
from_set	x'00 through x'ff.
to_set	To the NULL string and effectively extended with pad to the length of from_set as necessary.
pad	x'00'.

Returns

Table 59–18 TRANSLITERATE Function Returns

Return	Description
RAW	Converted byte-string.

Errors

Table 59–19 TRANSLITERATE Function Errors

Error	Description
VALUE_ERROR	R is NULL and / or has 0 length.

OVERLAY function

This function overlays the specified portion of target RAW with overlay RAW, starting from byte position pos of target and proceeding for len bytes.

If overlay has less than len bytes, then it is extended to len bytes using the pad byte. If overlay exceeds len bytes, then the extra bytes in overlay are ignored. If len bytes beginning at position pos of target exceeds the length of target, then target is extended to contain the entire length of overlay.

len, if specified, must be greater than, or equal to, 0. pos, if specified, must be greater than, or equal to, 1. If pos exceeds the length of target, then target is padded with pad bytes to position pos, and then target is further extended with overlay bytes.

Syntax

```
UTL_RAW.OVERLAY (
   overlay_str IN RAW,
   target      IN RAW,
   pos         IN BINARY_INTEGER DEFAULT 1,
   len         IN BINARY_INTEGER DEFAULT NULL,
   pad         IN RAW            DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(overlay, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–20 OVERLAY Function Parameters

Parameters	Description
overlay_str	Byte-string used to overlay target.
target	Byte-string which is to be overlayed.
pos	Position in target (numbered from 1) to start overlay.
len	The number of target bytes to overlay.
pad	Pad byte used when overlay len exceeds overlay length or pos exceeds target length.

Defaults and Optional Parameters

Table 59–21 OVERLAY Function Optional Parameters

Optional Parameter	Description
pos	1
len	To the length of overlay
pad	x'00'

Returns

Table 59–22 OVERLAY Function Returns

Return	Description
RAW	The target byte_string overlayed as specified.

Errors

Table 59–23 OVERLAY Function Errors

Error	Description
VALUE_ERROR	Either: - Overlay is NULL and / or has 0 length - Target is missing or undefined - Length of target exceeds maximum length of a RAW - len < 0 - pos < 1

COPIES function

This function returns n copies of r concatenated together.

Syntax

```
UTL_RAW.COPIES (
   r IN RAW,
   n IN NUMBER)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(copies, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–24 COPIES Function Parameters

Parameters	Description
r	RAW to be copied

Table 59–24 COPIES Function Parameters

Parameters	Description
n	Number of times to copy the RAW (must be positive).

Returns

This returns the RAW copied n times.

Errors

Table 59–25 COPIES Function Errors

Error	Description
VALUE_ERROR	Either: - r is missing, NULL and / or 0 length - n < 1 - Length of result exceeds maximum length of a RAW

XRANGE function

This function returns a RAW containing all valid 1-byte encodings in succession, beginning with the value start_byte and ending with the value end_byte. If start_byte is greater than end_byte, then the succession of result bytes begin with start_byte, wrap through 'FF'x to '00'x, and end at end_byte. If specified, then start_byte and end_byte must be single byte RAWs.

Syntax

```
UTL_RAW.XRANGE (
   start_byte IN RAW DEFAULT NULL,
   end_byte   IN RAW DEFAULT NULL)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(xrange, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–26 XRANGE Function Parameters

Parameters	Description
start_byte	Beginning byte-code value of resulting sequence.
end_byte	Ending byte-code value of resulting sequence.

Defaults and Optional Parameters

```
start_byte - x'00'
start_byte - x'00'
end_byte   - x'FF'
```

Returns

Table 59–27 XRANGE Function Returns

Return	Description
RAW	Containing succession of 1-byte hexadecimal encodings.

Errors

None.

REVERSE function

This function reverses a byte sequence in RAW r from end to end. For example, x'0102F3' would be reversed into x'F30201', and 'xyz' would be reversed into 'zyx'.The result length is the same as the input RAW length.

Syntax

```
UTL_RAW.REVERSE (
   r IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(reverse, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–28 REVERSE Function Parameters

Parameter	Description
r	RAW to reverse.

Returns

Table 59–29 REVERSE Function Returns

Return	Description
RAW	Containing the "reverse" of r.

Errors

Table 59–30 REVERSE Function Errors

Error	Description
VALUE_ERROR	R is NULL and / or has 0 length.

COMPARE function

This function compares RAW r1 against RAW r2. If r1 and r2 differ in length, then the shorter RAW is extended on the right with pad if necessary.

Syntax

```
UTL_RAW.COMPARE (
   r1  IN RAW,
   r2  IN RAW,
   pad IN RAW DEFAULT NULL)
  RETURN NUMBER;
```

Pragmas

```
pragma restrict_references(compare, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–31 COMPARE Function Parameters

Parameter	Description
r1	1st RAW to be compared, may be NULL and / or 0 length.
r2	2nd RAW to be compared, may be NULL and / or 0 length.
pad	Byte to extend whichever of r1 or r2 is shorter.

Defaults and optional parameters

pad - x'00'

Returns

Table 59–32 COMPARE Function Returns

Return	Description
NUMBER	Equals 0 if RAW byte strings are both NULL or identical; or, Equals position (numbered from 1) of the first mismatched byte.

Errors

None.

CONVERT function

This function converts RAW r from character set from_charset to character set to_charset and returns the resulting RAW.

Both from_charset and to_charset must be supported character sets defined to the Oracle server.

Syntax

```
UTL_RAW.CONVERT (
   r            IN RAW,
   to_charset   IN VARCHAR2,
   from_charset IN VARCHAR2)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(convert, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–33 CONVERT Function Parameters

Parameter	Description
r	RAW byte-string to be converted.
to_charset	Name of NLS character set to which r is converted.
from_charset	Name of NLS character set in which r is supplied.

Returns

Table 59–34 CONVERT Function Returns

Return	Description
RAW	Byte string r converted according to the specified character sets.

Errors

Table 59–35 CONVERT Function Errors

Error	Description
VALUE_ERROR	Either: - r missing, NULL, and / or 0 length - from_charset or to_charset missing, NULL, and / or 0 length - from_charset or to_charset names invalid or unsupported

BIT_AND function

This function performs bitwise logical "and" of the values in RAW r1 with RAW r2 and returns the "anded" result RAW.

If r1 and r2 differ in length, then the "and" operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result. The result length equals the longer of the two input RAWs.

Syntax

```
UTL_RAW.BIT_AND (
   r1 IN RAW,
   r2 IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_and, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–36 BIT_AND Function Parameters

Parameter	Description
r1	RAW to "and" with r2.
r2	RAW to "and" with r1.

Returns

Table 59–37 BIT_AND Function Returns

Return	Description
RAW	Containing the "and" of r1 and r2.
NULL	Either r1 or r2 input parameter was NULL.

Errors

None.

BIT_OR function

This function performs bitwise logical "or" of the values in RAW r1 with RAW r2 and returns the "or'd" result RAW.

If r1 and r2 differ in length, then the "or" operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result.The result length equals the longer of the two input RAWs.

Syntax

```
UTL_RAW.BIT_OR (
   r1 IN RAW,
   r2 IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_or, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–38 BIT_OR Function Parameters

Parameters	Description
r1	RAW to "or" with r2.
r2	RAW to "or" with r1.

Returns

Table 59–39 BIT_OR Function Returns

Return	Description
RAW	Containing the "or" of r1 and r2.
NULL	Either r1 or r2 input parameter was NULL.

Errors

None.

BIT_XOR function

This function performs bitwise logical "exclusive or" of the values in RAW r1 with RAW r2 and returns the "xor'd" result RAW.

If r1 and r2 differ in length, then the "xor" operation is terminated after the last byte of the shorter of the two RAWs, and the unprocessed portion of the longer RAW is appended to the partial result.The result length equals the longer of the two input RAWs.

Syntax

```
UTL_RAW.BIT_XOR (
   r1 IN RAW,
   r2 IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_xor, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–40 BIT_XOR Function Parameters

Parameter	Description
r1	RAW to "xor" with r2.
r2	RAW to "xor" with r1 .

Returns

Table 59–41 BIT_XOR Function Returns

Return	Description
RAW	Containing the "xor" of r1 and r2.
NULL	If either r1 or r2 input parameter was NULL.

Errors

None.

BIT_COMPLEMENT function

This function performs bitwise logical "complement" of the values in RAW r and returns the "complement'ed" result RAW. The result length equals the input RAW r length.

Syntax

```
UTL_RAW.BIT_COMPLEMENT (
   r IN RAW)
  RETURN RAW;
```

Pragmas

```
pragma restrict_references(bit_complement, WNDS, RNDS, WNPS, RNPS);
```

Parameters

Table 59–42 BIT_COMPLEMENT Function Parameters

Parameter	Description
r	RAW to perform "complement" operation.

Returns

Table 59–43 BIT_COMPLEMENT Function Returns

Return	Description
RAW	The "complement" of r1.
NULL	If r input parameter was NULL.

Errors

None.

60

UTL_REF

Oracle8*i* supports user-defined composite type or object type. Any instance of an object type is called an object. An object type can be used as the type of a column or as the type of a table.

In an object table, each row of the table stores an object. You can uniquely identify an object in an object table with an object identifier.

A reference is a persistent pointer to an object, and each reference can contain an object identifier. The reference can be an attribute of an object type, or it can be stored in a column of a table. Given a reference, an object can be retrieved.

The UTL_REF package provides PL/SQL procedures to support reference-based operations. Unlike SQL, UTL_REF procedures enable you to write generic type methods without knowing the object table name.

Requirements

The procedural option is needed to use this package. This package must be created under SYS (connect internal). Operations provided by this package are performed under the current calling user, not under the package owner SYS.

Datatypes

An object type is a composite datatype defined by the user or supplied as a library type. You can create the object type `employee_type` using the following syntax:

```
CREATE TYPE employee_type AS OBJECT (
    name    VARCHAR2(20),
    id      NUMBER,

    member function GET_ID
        (name VARCHAR2)
       RETURN MEMBER);
```

The object type `employee_type` is a user-defined type that contains two attributes, `name` and `id`, and a member function, `GET_ID`().

You can create an object table using the following SQL syntax:

```
CREATE TABLE employee_table OF employee_type;
```

Exceptions

Exceptions can be returned during execution of `UTL_REF` functions for various reasons. For example, the following scenarios would result in exceptions:

- The object selected does not exist. This could be because either:
  1. The object has been deleted, or the given reference is dangling (invalid).
  2. The object table was dropped or does not exist.
- The object cannot be modified or locked in a serializable transaction. The object was modified by another transaction after the serializable transaction started.
- You do not have the privilege to select or modify the object. The caller of the `UTL_REF` subprogram must have the proper privilege on the object that is being selected or modified.

Table 60–1 UTL_REF Exceptions

Exceptions	Description
errnum == 942	Insufficient privileges.
errnum == 1031	Insufficient privileges.
errnum == 8177	Unable to serialize, if in a serializable transaction.
errnum == 60	Deadlock detected.
errnum == 1403	No data found (if the REF is null, etc.).

The UTL_REF package does not define any named exceptions. You may define exception handling blocks to catch specific exceptions and to handle them appropriately.

Security

You can use the UTL_REF package from stored PL/SQL procedures/packages on the server, as well as from client/side PL/SQL code.

When invoked from PL/SQL procedures/packages on the server, UTL_REF verifies that the invoker has the appropriate privileges to access the object pointed to by the REF.

> **Note:** This is in contrast to PL/SQL packages/procedures on the server which operate with definer's privileges, where the package owner must have the appropriate privileges to perform the desired operations.

Thus, if UTL_REF is defined under user SYS, and user A invokes UTL_REF.SELECT to select an object from a reference, then user A (the invoker) requires the privileges to check.

When invoked from client-side PL/SQL code, UTL_REF operates with the privileges of the client session under which the PL/SQL execution is being done.

Summary of Subprograms

Table 60–2 UTL_REF Subprograms

Subprogram	Description
SELECT_OBJECT procedure on page 60-4	Selects an object given a reference.
LOCK_OBJECT procedure on page 60-5	Locks an object given a reference.
UPDATE_OBJECT procedure on page 60-6	Updates an object given a reference.
DELETE_OBJECT procedure on page 60-7	Deletes an object given a reference.

SELECT_OBJECT procedure

This procedure selects an object given its reference. The selected object is retrieved from the database and its value is put into the PL/SQL variable 'object'. The semantic of this subprogram is similar to the following SQL statement:

```
SELECT VALUE(t)
INTO object
FROM object_table t
WHERE REF(t) = reference;
```

Unlike the above SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

```
UTL_REF.SELECT_OBJECT (
   reference IN REF "<typename>",
   object    IN OUT "<typename>");
```

Parameters

Table 60–3 SELECT_OBJECT Procedure Parameters

Parameter	Description
reference	Reference to the object to select or retrieve.

Table 60–3 SELECT_OBJECT Procedure Parameters

Parameter	Description
object	The PL/SQL variable that stores the selected object; this variable should be of the same object type as the referenced object.

Returns

None.

Pragmas

None.

Exceptions

May be raised.

LOCK_OBJECT procedure

This procedure locks an object given a reference. In addition, this procedure lets the program select the locked object. The semantic of this subprogram is similar to the following SQL statement:

```
SELECT VALUE(t)
  INTO object
  FROM object_table t
  WHERE REF(t) = reference
  FOR UPDATE;
```

Unlike the above SQL statement, this subprogram does not require you to specify the object table name where the object resides. It is not necessary to lock an object before updating/deleting it.

Syntax

```
UTL_REF.LOCK_OBJECT (
   reference IN REF "<typename>");

UTL_REF.LOCK_OBJECT (
   reference IN REF "<typename>",
   object    IN OUT "<typename>");
```

Parameters

Table 60–4 LOCK_OBJECT Procedure Parameters

Parameter	Description
reference	Reference of the object to lock.
object	The PL/SQL variable that stores the locked object. This variable should be of the same object type as the locked object.

Returns

None.

Pragmas

None.

Exceptions

May be raised.

UPDATE_OBJECT procedure

This procedure updates an object given a reference. The referenced object is updated with the value contained in the PL/SQL variable 'object'. The semantic of this subprogram is similar to the following SQL statement:

```
UPDATE object_table t
SET VALUE(t) = object
WHERE REF(t) = reference;
```

Unlike the above SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

```
UTL_REF.UPDATE_OBJECT (
   reference IN REF "<typename>",
   object    IN     "<typename>");
```

Parameters

Table 60–5 UPDATE_OBJECT Procedure Parameters

Parameter	Description
reference	Reference of the object to update.
object	The PL/SQL variable that contains the new value of the object. This variable should be of the same object type as the object to update.

Returns

None.

Pragmas

None.

Exceptions

May be raised.

DELETE_OBJECT procedure

This procedure deletes an object given a reference. The semantic of this subprogram is similar to the following SQL statement:

```
DELETE FROM object_table
WHERE REF(t) = reference;
```

Unlike the above SQL statement, this subprogram does not require you to specify the object table name where the object resides.

Syntax

```
UTL_REF.DELETE_OBJECT (
   reference IN REF "<typename>");
```

Parameters

Table 60–6 DELETE_OBJECT Procedure Parameters

Parameter	Description
reference	Reference of the object to delete.

Returns

None.

Pragmas

None.

Exceptions

May be raised.

Example

The following example illustrates usage of the UTL_REF package to implement this scenario: if an employee of a company changes their address, their manager should be notified.

... declarations of Address_t and others...

```
CREATE OR REPLACE TYPE Person_t (
    name    VARCHAR2(64),
    gender  CHAR(1),
    address Address_t,
    MEMBER PROCEDURE setAddress(addr IN Address_t)
);

CREATE OR REPLACE TYPE BODY Person_t (
    MEMBER PROCEDURE setAddress(addr IN Address_t) IS
    BEGIN
        address := addr;
    END;
);

CREATE OR REPLACE TYPE Employee_t (
```

Under Person_t: Simulate implementation of inheritance using a REF to Person_t and delegation of setAddress to it.

```
    thePerson  REF Person_t,
    empno      NUMBER(5),
    deptREF    Department_t,
    mgrREF     Employee_t,
    reminders  StringArray_t,
    MEMBER PROCEDURE setAddress(addr IN Address_t),
    MEMBER procedure addReminder(reminder VARCHAR2);
```

```
);

CREATE TYPE BODY Employee_t (
    MEMBER PROCEDURE setAddress(addr IN Address_t) IS
        myMgr Employee_t;
        meAsPerson Person_t;
    BEGIN
```

Update the address by delegating the responsibility to `thePerson`. Lock the Person object from the reference, and also select it:

```
        UTL_REF.LOCK_OBJECT(thePerson, meAsPerson);
        meAsPerson.setAddress(addr);
```

Delegate to `thePerson`:

```
        UTL_REF.UPDATE_OBJECT(thePerson, meAsPerson);
        if mgr is NOT NULL THEN
```

Give the manager a reminder:

```
            UTL_REF.LOCK_OBJECT(mgr);
            UTL_REF.SELECT_OBJECT(mgr, myMgr);
            myMgr.addReminder
            ('Update address in the employee directory for' ||
            thePerson.name || ', new address: ' || addr.asString);
            UTL_REF.UPDATE_OBJECT(mgr, myMgr);
        END IF;
    EXCEPTION
        WHEN OTHERS THEN
        errnum := SQLCODE;
        errmsg := SUBSTR(SQLERRM, 1, 200);
```

Index

A

C

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U

V