Oracle8*i*

Replication Management API Reference

Release 2 (8.1.6)

December 1999

A76958-01

ORACLE

Oracle8i Replication Management API Reference, Release 2 (8.1.6)

A76958-01

Primary Author: William Creekbaum and Randy Urbano

Graphic Artist: Valarie Moore

Contributors: Nimar Arora, Alan Downing, Al Demers, Ira Greenberg, Denis Goddard, Maria Pratt, Jim Stamos, Curt Elsbernd, Jairaj Galagali, Viswanathan Krishnamurthy, Jing Liu, Pat McElroy, Arvind Rajaram, Wayne Smith, Eric Vandevelde, Lik Wong

Contents

8 Replication Management API Reference

9 Data Dictionary Views

Send Us Your Comments

Oracle8i Replication Management API Reference, Release 2 (8.1.6)

A76958-01

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:

- E-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 City, CA 94065
  U.S.A.

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

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This Preface contains the following topics:

The *Oracle8i Replication Management API Reference* contains information that describes the features and functionality of the Oracle8*i* and the Oracle8*i* Enterprise Edition products. Oracle8*i* and the Oracle8*i* Enterprise Edition have the same basic features. However, several advanced features are available only with the Enterprise Edition, and some of these are optional. For example, to use partitioning, you must have the Enterprise Edition and the partitioning option.

> **See Also:** *Getting to Know Oracle8i* for information about the differences between Oracle8*i* and the Oracle8*i* Enterprise Edition and the features and options that are available to you.

Overview of This Reference

This reference describes the replication management API. This reference assumes that you are familiar with the replication concepts described in *Oracle8i Replication*.

The emphasis of this reference is to illustrate how the replication management API is used and to serve as a quick reference source for the replication management API.

Information in this reference applies to the Oracle8*i* server running on all operating systems. Topics include the following:

- Setup Replication Site
- Create Master Group
- Create Deployment Template
- Create Snapshot Group
- Conflict Resolution
- Managing Replication Environment
- Replication Management API Reference
- Data Dictionary Views
- Security Options

Audience

This reference is written for database administrators and application developers who develop and maintain Oracle8*i* replication environments.

This reference assumes you are familiar with relational database concepts, distributed database administration, PL/SQL (if using procedural replication), and the operating system under which you run an Oracle replicated environment.

This reference also assumes that you have read and understand the information in the following documents:

- *Oracle8i Concepts*
- *Oracle8i Administrator's Guide*
- *Oracle8i Distributed Database Systems*
- *PL/SQL User's Guide and Reference*
- *Oracle8i Replication*

How This Reference Is Organized

This reference contains the following chapters and appendices:

Chapter 1, "Replication Overview"
Provides an overview of process for building a replicated environment with the replication management API. This chapter also contains some prerequisites for building a replicated environment.

Chapter 2, "Create Replication Site"
Describes in detail the process of setting up both a master and snapshot site. Consult this chapter when building a new replicated environment and when adding either a new master or snapshot site to an established replicated environment.

Chapter 3, "Create a Master Group"
Describes how to build a master group for multimaster replication or as a master for a snapshot site. Chapter 3 builds a master group that replicates data between the three master sites that were set up in Chapter 2.

Chapter 4, "Create Deployment Template"
Describes how to build a snapshot environment with deployment templates, which are the most effective method of distributing a snapshot environment to any number of snapshot sites.

Chapter 5, "Create Snapshot Group"
Describes how to build a snapshot environment with snapshot groups. If deployment templates do not meet your requirements, Chapter 5 describes in detail how to build a snapshot environment at the snapshot site.

Chapter 6, "Conflict Resolution"
Describes the conflict resolution methods that can help your data converge at all sites when a data conflict arises.

Chapter 7, "Manage Replicated Environment with APIs"
Describes many of the management tasks that you may need to perform to manage your replicated environment. Topics discussed include master group management, altering replicated objects, offline instantiation, and more.

Chapter 8, "Replication Management API Reference"
Describes the parameters for the packaged procedures and functions used to implement a replicated environment, as well as exceptions these procedures and functions might raise.

Chapter 9, "Data Dictionary Views"
Describes views of interest to users of deferred transactions, read-only snapshots, and the Oracle replication.

Appendix A, "Security Options"
Describes setting up security for multimaster and snapshot replication using the replication management API.

Appendix B, "User-Defined Conflict Resolution Methods"
Describes building user-defined conflict resolution methods and notification functions using the replication management API.

Changes To This Book

The following major change was made to this book:

- Two appendices were added: Appendix A, "Security Options" and Appendix B, "User-Defined Conflict Resolution Methods".

Conventions Used in This Reference

This reference uses different fonts to represent different types of information.

Special Notes

Special notes alert you to particular information within the body of this reference:

Note: Indicates special or auxiliary information.

See Also: Indicates where to get more information.

Caution: Indicates important information about possible damage to your system or your data.

Text of the Reference

The following sections describe conventions used this reference.

UPPERCASE Characters

Uppercase text is used to call attention to statement keywords, object names, initialization parameters, and data dictionary views.

For example, "If you create a private rollback segment, the name must be included in the ROLLBACK_SEGMENTS initialization parameter".

Italicized **Characters**

Italicized words within text indicate the definition of a word, book titles, or emphasized words.

An example of a definition is the following: "A *database* is a collection of data to be treated as a unit. The general purpose of a database is to store and retrieve related information".

An example of a reference to another book is the following: "For more information, see *Oracle8i Designing and Tuning for Performance*."

An example of an emphasized word is the following: "You *must* back up your database regularly".

Code Examples

SQL, Server Manager line mode, and SQL*Plus commands/statements appear separated from the text of paragraphs in a monospaced font. For example:

```
INSERT INTO emp (empno, ename) VALUES (1000, 'SMITH');

ALTER TABLESPACE users ADD DATAFILE 'users2.ora' SIZE 50K;
```

Example statements may include punctuation, such as commas or quotation marks. All punctuation in example statements is required. All example statements terminate with a semicolon (;). Depending on the application, a semicolon or other terminator may or may not be required to end a statement.

Uppercase words in example statements indicate the keywords within Oracle SQL. When issuing statements, however, keywords are not case sensitive. Lowercase words in example statements indicate words supplied only for the context of the example. For example, lowercase words may indicate the name of a table, column, or file.

Your Comments Are Welcome

We value your comments as an Oracle user and reader of our manuals. As we write, revise, and evaluate, your opinions are the most important input we receive. This manual contains a Reader's Comment Form that we encourage you to use to tell us what you like and dislike about this manual or other Oracle manuals. Please mail comments to:

Server Technologies Documentation Manager

Oracle Corporation

500 Oracle Parkway

Redwood City, CA 94065

U.S.A.

Fax - 650.506.7228 Attn: Server Technologies Documentation Manager

You can send comments and suggestions to the Information Development department at the following e-mail address:

infodev@us.oracle.com

1

Replication Overview

This chapter reviews the process of building a replicated environment with the replication management API. The following topics are discussed:

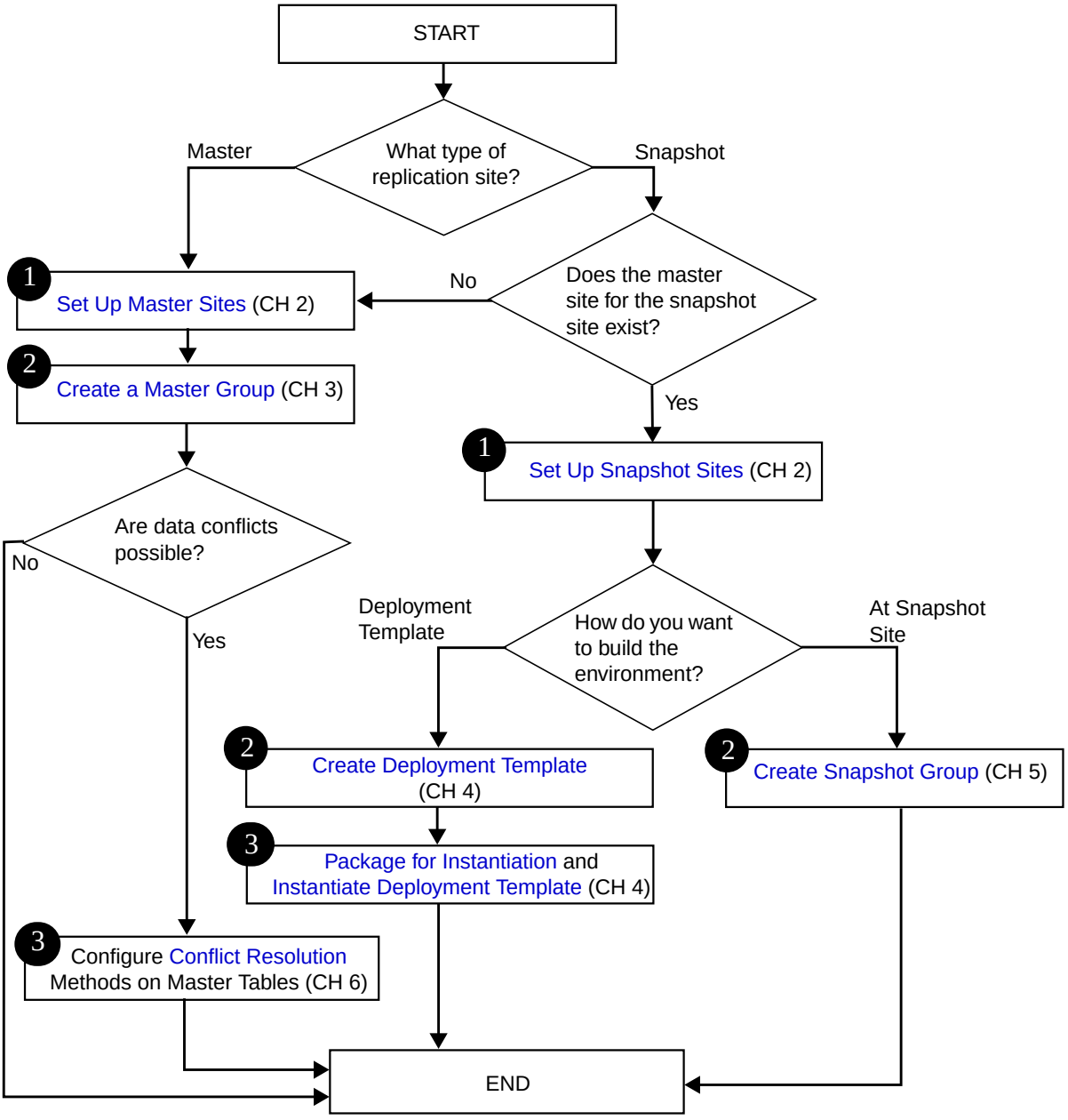
- Creating a Replicated Environment Overview
- Before You Start

Creating a Replicated Environment Overview

Figure 1–1 illustrates the basic steps required to build a replicated environment. Regardless of the type of replication site or sites that you are building, you begin by setting up the replicated site.

After you have set up your replication sites, you are ready to begin building your master and snapshot groups. After you have built your replication environment, make sure that you review Chapter 6 and Chapter 7 to learn about conflict resolution and managing your replicated environment.

Figure 1–1 Create Replicated Environment Process



Before You Start

Before you begin setting up your replication site, there are several items that you need to verify:

- Ensure that GLOBAL_NAMES is set to TRUE in your initialization parameter file.
- Ensure that you have allocated enough job processes at each master site.

Global Names

To ensure that your replicated environment works properly, you must set the GLOBAL_NAMES initialization parameter in the initialization parameter file to TRUE. The following description and setting are in your initialization parameter file:

```
# Global Naming -- enforce that a dblink has same name as the db it connects to
global_names = TRUE
```

This initialization parameter must be set to TRUE for each database that is participating in your replicated environment, including both master and snapshot sites.

Job Processes

It is important that you have allocated sufficient job processes, sometimes referred to as SNP background processes, to handle the automation of your replicated environment. That is, automatically propagating the deferred transaction queue, purging the deferred transaction queue, refreshing snapshots, and so on.

For multimaster replication, each site has a scheduled link to each of the other master sites. For example, if you have six master sites, each site has scheduled links to the other five sites. You typically need one process for each scheduled link. You may also want to add an additional job process for purging the deferred transaction queue and other user-defined jobs.

By the nature of snapshot replication, each snapshot site typically has one scheduled link to the master database and requires at least one job process. Snapshot sites typically require between one and three job processes, depending on purge scheduling, user-defined jobs, and the scheduled link. Alternatively, if your users are responsible for manually refreshing the snapshot through an application interface, you do not need to create a scheduled link and your snapshot site requires one less job process.

In addition to defining the number of job processes, you must also define the job interval. The job interval determines how often your job processes "wake up" to execute any pending operations, such as pushing a queue. While the default value of 60 seconds is adequate for most replicated environments, you may need to adjust this value to maximize performance for you individual requirements. For example, if you want to propagate changes every 20 seconds, a job interval of 60 seconds would not be sufficient. On the other hand, if you need to propagate your changes once a day, you may only want your SNP process to check for a pending operation once an hour.

The job processes are also defined in the initialization parameter file, usually under the "Oracle replication" heading.

```
job_queue_processes = 7
job_queue_interval = 60
```

After you have modified the contents of your initialization parameter file, restart your database with these new settings.

> **See Also:** Chapter 7, "Planning Your Replication Environment" in *Oracle8i Replication* for more information about the initialization parameters that are important for Oracle replication, and see the *Oracle8i Administrator's Guide* for information on restarting your database.

2

Create Replication Site

This chapter illustrates how to setup both a master and a snapshot replication site using the replication management API. The following topics are discussed:

- Overview of Setting Up Replication Sites
- Set Up Master Sites
- Set Up Snapshot Sites

Overview of Setting Up Replication Sites

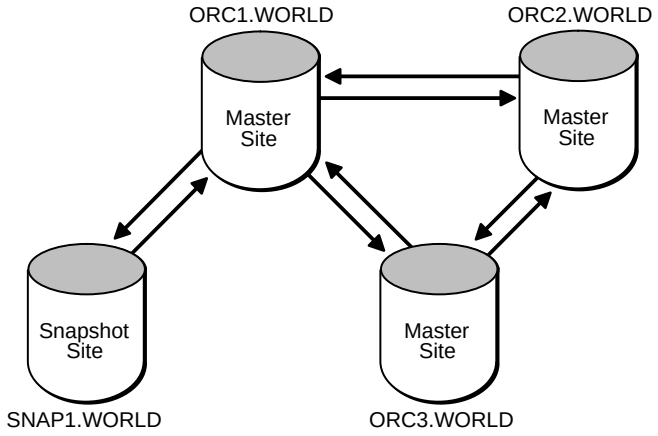
Before you begin building your replicated environment, you need to set up the sites that will participate in the replicated environment. As illustrated in Figure 2–2 and Figure 2–3, there are separate processes for setting up a master site versus setting up a snapshot site.

This chapter assumes that you have the following:

- Four Databases:
  - ORC1.WORLD
  - ORC2.WORLD
  - ORC3.WORLD
  - SNAP1.WORLD

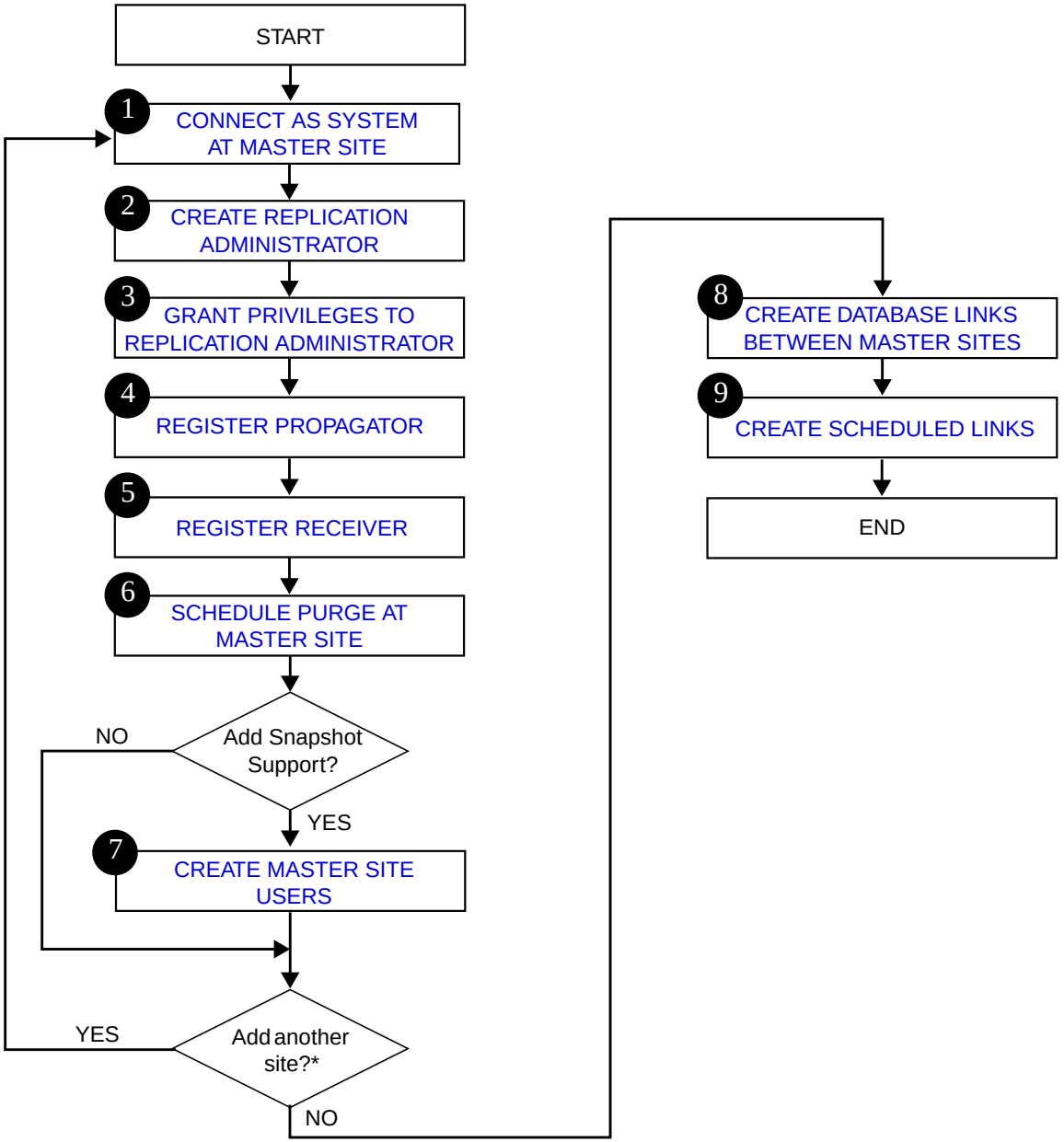
Chapters 2 - 6 work with the replication environment illustrated in Figure 2–1. You start to create this environment in this chapter.

Figure 2–1 Three Master Sites and One Snapshot Site



Follow the procedures identified in Figure 2–2 when you build a new master site or in Figure 2–3 when you build a new snapshot site.

Figure 2–2 Set Up Master Sites



*Multiple master sites (multimaster replication) can be used only with the Enterprise Edition of Oracle.

Set Up Master Sites

```
/*************************************************************************
STEP 1 @ ORC1.WORLD:
CONNECT AS SYSTEM AT MASTER SITE
*************************************************************************/

--Connect as SYSTEM to the database that you want to
--setup for replication. After you setup ORC1.WORLD,
--begin again with STEP 1 for sites ORC2.WORLD on page 2-7 and
--ORC3.WORLD on page 2-10.

CONNECT system/manager@orc1.world

/*************************************************************************
STEP 2 @ ORC1.WORLD:
CREATE REPLICATION ADMINISTRATOR
*************************************************************************/

--The replication administrator must be granted the necessary privileges
--to create and manage a replicated environment. The replication
--administrator must be created at each database that participates
--in the replicated environment.

CREATE USER repadmin IDENTIFIED BY repadmin;

/*************************************************************************
STEP 3 @ ORC1.WORLD:
GRANT PRIVILEGES TO REPLICATION ADMINISTRATOR

For additional information about the GRANT_ADMIN_ANY_SCHEMA API, see "GRANT_
ADMIN_ANY_SCHEMA procedure" on page 8-156.
*************************************************************************/

--Executing the GRANT_ADMIN_ANY_SCHEMA API grants the replication
--administrator powerful privileges to create and manage a replicated
--environment.

BEGIN
   DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
      USERNAME => 'repadmin');
END;
/

--If you want your REPADMIN to be able to create snapshot logs for any
--replicated table, grant COMMENT ANY TABLE and LOCK ANY TABLE to REPADMIN.
```

```
/*************************************************************************
STEP 4 @ ORC1.WORLD:
REGISTER PROPAGATOR

For additional information about the REGISTER_PROPAGATOR API, see
"REGISTER_PROPAGATOR procedure" on page 8-31.
*************************************************************************/

--The propagator is responsible for propagating the deferred transaction
--queue to other master sites.

BEGIN
   DBMS_DEFER_SYS.REGISTER_PROPAGATOR (
      USERNAME => 'repadmin');
END;
/

/*************************************************************************
STEP 5 @ ORC1.WORLD:
REGISTER RECEIVER

For additional information about the REGISTER_USER_REPGROUP API,
see "REGISTER_USER_REPGROUP procedure" on page 8-157.
*************************************************************************/

--The receiver receives the propagated deferred transactions sent
--by the propagator from other master sites.

BEGIN
   DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
      USERNAME => 'repadmin',
      PRIVILEGE_TYPE => 'receiver',
      LIST_OF_GNAMES => NULL);
END;
/
```

```
/*************************************************************************
STEP 6 @ ORC1.WORLD:
SCHEDULE PURGE AT MASTER SITE

For additional information about the SCHEDULE_PURGE API, see "SCHEDULE_PURGE
procedure" on page 8-32.
*************************************************************************/

--In order to keep the size of the deferred transaction queue in check,
--you should purge successfully completed deferred transactions. The
--SCHEDULE_PURGE API automates the purge process for you. You must execute
--this procedure as the replication administrator.

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
      NEXT_DATE => SYSDATE,
      INTERVAL => 'SYSDATE + 1/24',
      DELAY_SECONDS => 0,
      ROLLBACK_SEGMENT => '');
END;
/

/*************************************************************************
STEP 7:
CREATE MASTER SITE USERS
*************************************************************************/

--STEP 7a: CREATE PROXY SNAPSHOT ADMINISTRATOR
--The proxy snapshot administrator performs tasks at the target master
--site on behalf of the snapshot administrator at the snapshot
--site.  See "Security Setup for Snapshot Replication" in
--Oracle8i Replication.

CONNECT system/manager@orc1.world

CREATE USER proxy_snapadmin IDENTIFIED BY proxy_snapadmin;

BEGIN
   DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
      USERNAME => 'proxy_snapadmin',
      PRIVILEGE_TYPE => 'proxy_snapadmin',
      LIST_OF_GNAMES => NULL);
END;
/
```

```
--STEP 7b: CREATE PROXY REFRESHER
--The proxy refresher performs tasks at the master site on behalf of
--the refresher at the snapshot site.

CREATE USER proxy_refresher IDENTIFIED BY proxy_refresher;

GRANT CREATE SESSION TO proxy_refresher;
GRANT SELECT ANY TABLE TO proxy_refresher;

/*************************************************************************
STEP 1  @ ORC2.WORLD:
CONNECT AS SYSTEM
*************************************************************************/

--NOTE:
--Multiple master sites (multimaster replication) can only be used with
--Oracle8i Enterprise Edition. If you are not using Oracle8i Enterprise
--Edition, skip to step 8 on page 2-12.

--You must connect as SYSTEM to the database that you want to
--set up for replication. After you set up ORC2.WORLD,
--begin again with STEP 1 for site ORC3.WORLD on page 2-10.

CONNECT system/manager@orc2.world

/*************************************************************************
STEP 2  @ ORC2.WORLD:
CREATE REPLICATION ADMINISTRATOR
*************************************************************************/

--The replication administrator must be granted the necessary privileges
--to create and manage a replicated environment. The replication
--administrator must be created at each database that participates
--in the replicated environment.

CREATE USER repadmin IDENTIFIED BY repadmin;
```

```
/*************************************************************************
STEP 3  @ ORC2.WORLD:
GRANT PRIVILEGES TO REPLICATION ADMINISTRATOR

For additional information about the GRANT_ADMIN_ANY_SCHEMA API, see "GRANT_
ADMIN_ANY_SCHEMA procedure" on page 8-156.
*************************************************************************/

--Executing the GRANT_ADMIN_ANY_SCHEMA API grants the replication
--administrator powerful privileges to create and manage a replicated
--environment.

BEGIN
   DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
      USERNAME => 'repadmin');
END;
/

--If you want your REPADMIN to be able to create snapshot logs for any
--replicated table, grant COMMENT ANY TABLE and LOCK ANY TABLE to REPADMIN.
/*************************************************************************
STEP 4  @ ORC2.WORLD:
REGISTER PROPAGATOR

For additional information about the REGISTER_PROPAGATOR API, see "REGISTER_
PROPAGATOR procedure" on page 8-31.
*************************************************************************/

--The propagator is responsible for propagating the deferred transaction
--queue to other master sites.

BEGIN
   DBMS_DEFER_SYS.REGISTER_PROPAGATOR (
      USERNAME => 'repadmin');
END;
/
```

```
/*************************************************************************
STEP 5  @ ORC2.WORLD:
REGISTER RECEIVER

For additional information about the REGISTER_USER_REPGROUP API, see "REGISTER_
USER_REPGROUP procedure" on page 8-157.
*************************************************************************/

--The receiver receives the propagated deferred transactions sent
--by the propagator from the other master sites.

BEGIN
   DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
      USERNAME => 'repadmin',
      PRIVILEGE_TYPE => 'receiver',
      LIST_OF_GNAMES => NULL);
END;
/

/*************************************************************************
STEP 6  @ ORC2.WORLD:
SCHEDULE PURGE AT MASTER SITE

For additional information about the SCHEDULE_PURGE API, see "SCHEDULE_PURGE
procedure" on page 8-32.
*************************************************************************/

--In order to keep the size of the deferred transaction queue in check,
--you should purge successfully completed deferred transactions. The
--SCHEDULE_PURGE API automates the purge process for you. You must execute
--this procedure as the replication administrator.

CONNECT repadmin/repadmin@orc2.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
      NEXT_DATE => SYSDATE,
      INTERVAL => 'SYSDATE + 1/24',
      DELAY_SECONDS => 0,
      ROLLBACK_SEGMENT => '');
END;
/
```

```
/*************************************************************************
STEP 1  @ ORC3.WORLD:
CONNECT AS SYSTEM
*************************************************************************/

--NOTE:
--Multiple master sites (multimaster replication) can be used only with
--Oracle8i Enterprise Edition. If you are not using Oracle8i Enterprise
--Edition, skip to step 8 on page 2-12.

--You must connect as SYSTEM to the database that you want to
--set up for replication.

CONNECT system/manager@orc3.world

/*************************************************************************
STEP 2 @ ORC3.WORLD:
CREATE REPLICATION ADMINISTRATOR
*************************************************************************/

--The replication administrator must be granted the necessary privileges
--to create and manage a replicated environment. The replication
--administrator must be created at each database that participates
--in the replicated environment.

CREATE USER repadmin IDENTIFIED BY repadmin;

/*************************************************************************
STEP 3 @ ORC3.WORLD:
GRANT PRIVILEGES TO REPLICATION ADMINISTRATOR

For additional information about the GRANT_ADMIN_ANY_SCHEMA API, see "GRANT_
ADMIN_ANY_SCHEMA procedure" on page 8-156.
*************************************************************************/

--Executing the GRANT_ADMIN_ANY_SCHEMA API grants the replication
--administrator powerful privileges to create and manage a replicated
--environment.

BEGIN
   DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
      USERNAME => 'repadmin');
END;
/
```

```
--If you want your REPADMIN to be able to create snapshot logs for any
--replicated table, grant COMMENT ANY TABLE and LOCK ANY TABLE to REPADMIN.

/*************************************************************************
STEP 4 @ ORC3.WORLD:
REGISTER PROPAGATOR

For additional information about the REGISTER_PROPAGATOR API, see "REGISTER_
PROPAGATOR procedure" on page 8-31.
*************************************************************************/

--The propagator is responsible for propagating the deferred transaction
--queue to other master sites.

BEGIN
   DBMS_DEFER_SYS.REGISTER_PROPAGATOR (
      USERNAME => 'repadmin');
END;
/

/*************************************************************************
STEP 5 @ ORC3.WORLD:
REGISTER RECEIVER

For additional information about the REGISTER_USER_REPGROUP API, see "REGISTER_
USER_REPGROUP procedure" on page 8-157.
*************************************************************************/

--The receiver receives the propagated deferred transactions sent
--by the propagator from the other master sites.

BEGIN
   DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
      USERNAME => 'repadmin',
      PRIVILEGE_TYPE => 'receiver',
      LIST_OF_GNAMES => NULL);
END;
/
```

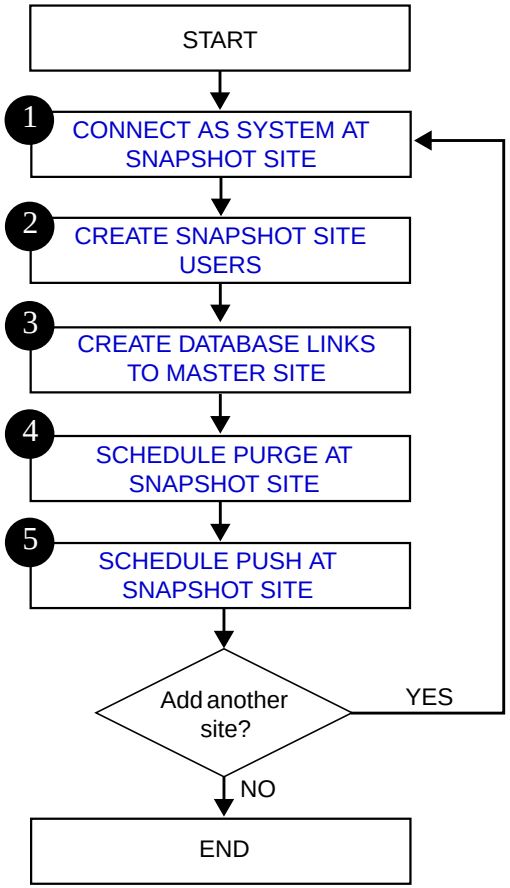
```
/*************************************************************************
STEP 6 @ ORC3.WORLD:
SCHEDULE PURGE AT MASTER SITE

For additional information about the SCHEDULE_PURGE API, see "SCHEDULE_PURGE
procedure" on page 8-32.
*************************************************************************/

--In order to keep the size of the deferred transaction queue in check,
--you should purge successfully completed deferred transactions. The
--SCHEDULE_PURGE API automates the purge process for you. You must execute
--this procedure as the replication administrator.

CONNECT repadmin/repadmin@orc3.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
      NEXT_DATE => SYSDATE,
      INTERVAL => 'SYSDATE + 1/24',
      DELAY_SECONDS => 0,
      ROLLBACK_SEGMENT => '');
END;
/

/*************************************************************************
STEP 7:
CREATE DATABASE LINKS BETWEEN MASTER SITES

The database links provide the necessary distributed mechanisms to allow
the different replication sites to replicate data among themselves. See
Oracle8i Distributed Database Systems for more information.
*************************************************************************/

--Before you create any private database links, you must create the
--public database links that each private database link will use.
--You then must create a database link between all replication
--administrators at each of the master sites that you have set up.

CONNECT system/manager@orc1.world
CREATE PUBLIC DATABASE LINK orc2.world USING 'orc2.world';
CREATE PUBLIC DATABASE LINK orc3.world USING 'orc3.world';

CONNECT repadmin/repadmin@orc1.world
CREATE DATABASE LINK orc2.world CONNECT TO repadmin IDENTIFIED BY repadmin;
CREATE DATABASE LINK orc3.world CONNECT TO repadmin IDENTIFIED BY repadmin;
```

```
CONNECT system/manager@orc2.world
CREATE PUBLIC DATABASE LINK orc1.world USING 'orc1.world';
CREATE PUBLIC DATABASE LINK orc3.world USING 'orc3.world';

CONNECT repadmin/repadmin@orc2.world
CREATE DATABASE LINK orc1.world CONNECT TO repadmin IDENTIFIED BY repadmin;
CREATE DATABASE LINK orc3.world CONNECT TO repadmin IDENTIFIED BY repadmin;

CONNECT system/manager@orc3.world
CREATE PUBLIC DATABASE LINK orc1.world USING 'orc1.world';
CREATE PUBLIC DATABASE LINK orc2.world USING 'orc2.world';

CONNECT repadmin/repadmin@orc3.world
CREATE DATABASE LINK orc1.world CONNECT TO repadmin IDENTIFIED BY repadmin;
CREATE DATABASE LINK orc2.world CONNECT TO repadmin IDENTIFIED BY repadmin;

/*************************************************************************
STEP 8:
CREATE SCHEDULED LINKS

Create a scheduled link by defining a database link when you execute the
SCHEDULE_PUSH procedure (see "SCHEDULE_PUSH procedure" on page 8-34
for more information).
*************************************************************************/

--The scheduled link determines how often your deferred transaction queue is
--propagated to each of the other master sites. You need to execute the
--SCHEDULE_PUSH procedure for each database link that you created
--in STEP 7. The database link is specified in the DESTINATION parameter
--of the SCHEDULE_PUSH procedure.

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc2.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/
```

```
BEGIN
DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc3.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/

CONNECT repadmin/repadmin@orc2.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc1.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc3.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/

CONNECT repadmin/repadmin@orc3.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc1.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc2.world',
      INTERVAL => 'SYSDATE + 10 / (24 * 60)',
      NEXT_DATE => SYSDATE);
END;
/
```

Set Up Snapshot Sites

Figure 2–3 Set Up Snapshot Sites



```
/*************************************************************************
STEP 1:
CONNECT AS SYSTEM AT SNAPSHOT SITE
*************************************************************************/
--You must connect as SYSTEM to the database that you want to
--set up as a snapshot site.

CONNECT system/manager@snap1.world

/*************************************************************************
STEP 2:
CREATE SNAPSHOT SITE USERS
*************************************************************************/

--Several users need to be created at the snapshot site. These users are:
--  SNAPSHOT ADMINISTRATOR
--  PROPAGATOR
--  REFRESHER

--STEP 2a: CREATE SNAPSHOT ADMINISTRATOR
--The snapshot administrator is responsible for creating and managing
--the snapshot site. Execute the GRANT_ADMIN_ANY_SCHEMA
--procedure to grant the snapshot administrator the appropriate privileges.

CREATE USER snapadmin IDENTIFIED BY snapadmin;

BEGIN
   DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
      USERNAME => 'snapadmin');
END;
/

--STEP 2b: CREATE PROPAGATOR
--The propagator is responsible for propagating the deferred transaction
--queue to the target master site.

CREATE USER propagator IDENTIFIED BY propagator;

BEGIN
   DBMS_DEFER_SYS.REGISTER_PROPAGATOR (
      USERNAME => 'propagator');
END;
/

--STEP 2c: CREATE REFRESHER
```

```
--The refresher is responsible for "pulling" changes made to the replicated
--tables at the target master site to the snapshot site.

CREATE USER refresher IDENTIFIED BY refresher;

GRANT CREATE SESSION TO refresher;
GRANT ALTER ANY SNAPSHOT TO refresher;

/*************************************************************************
STEP 3:
CREATE DATABASE LINKS TO MASTER SITE
*************************************************************************/

--STEP 3A: CREATE PUBLIC DATABASE LINK

CONNECT system/manager@snap1.world

CREATE PUBLIC DATABASE LINK orc1.world USING 'orc1.world';

--STEP 3b: CREATE SNAPSHOT ADMINISTRATOR DATABASE LINK
--You need to create a database link from the snapshot administrator at
--the snapshot site to the proxy snapshot administrator at
--the master site.

CONNECT snapadmin/snapadmin@snap1.world;

CREATE DATABASE LINK orc1.world
  CONNECT TO proxy_snapadmin IDENTIFIED BY proxy_snapadmin;

--STEP 3c: CREATE PROPAGATOR/RECEIVER DATABASE LINK
--You need to create a database link from the propagator at the
--snapshot site to the receiver at the master site. The receiver was defined
--when you created the master group - see "REGISTER RECEIVER" on page 2-5
--for more information.

CONNECT propagator/propagator@snap1.world

CREATE DATABASE LINK orc1.world
  CONNECT TO repadmin IDENTIFIED BY repadmin;
```

```
/*************************************************************************
STEP 4:
SCHEDULE PURGE AT SNAPSHOT SITE

For additional information about the SCHEDULE_PURGE API, see "SCHEDULE_PURGE
procedure" on page 8-32.
*************************************************************************/

--In order to keep the size of the deferred transaction queue in check,
--you should purge successfully completed deferred transactions. The
--SCHEDULE_PURGE API automates the purge process for you. If your snapshot
--site only contains "read-only" snapshots, then you do not need to
--execute this procedure.

CONNECT snapadmin/snapadmin@snap1.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
   NEXT_DATE => SYSDATE,
   INTERVAL => 'SYSDATE + 1/24',
   DELAY_SECONDS => 0,
   ROLLBACK_SEGMENT => '');
END;
/
```

```
/*************************************************************************
STEP 5:
SCHEDULE PUSH AT SNAPSHOT SITE

For additional information about the SCHEDULE_PUSH API, see "SCHEDULE_PUSH
procedure" on page 8-34.
*************************************************************************/

--The SCHEDULE_PUSH API schedules when the deferred transaction queue
--should be propagated to the target master site.

CONNECT snapadmin/snapadmin@snap1.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      DESTINATION => 'orc1.world',
      INTERVAL => 'SYSDATE + 1/24',
      NEXT_DATE => SYSDATE,
      STOP_ON_ERROR => FALSE,
      DELAY_SECONDS => 0,
      PARALLELISM => 0);
END;
/
```

3

Create a Master Group

This chapter illustrates how to create a master group at a master replication site. The following topics are discussed:

- Overview of Creating a Master Group
- Create Master Group

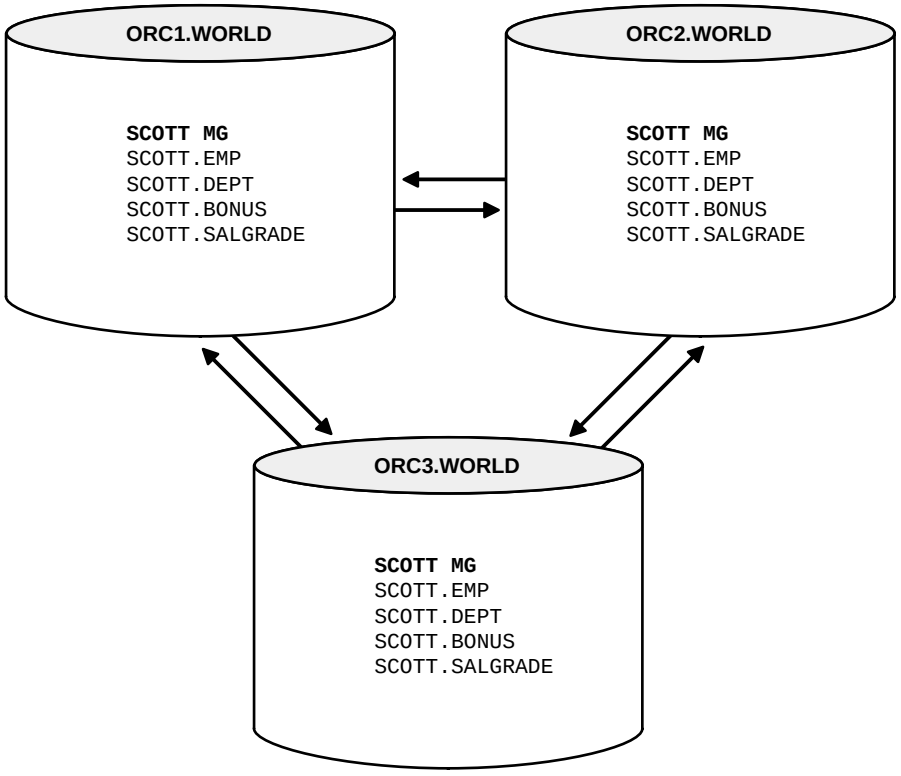
Overview of Creating a Master Group

After you have set up your master sites, you are ready to begin building a master group. As illustrated in Figure 3–2, you need to follow a specific sequence to successfully build a replicated environment.

> **See Also:** "Create Replication Site" on page 2-1 for information about setting up master sites.

In this chapter, you create the SCOTT_MG master group and replicate the objects illustrated in Figure 3–1:

Figure 3–1 Replicate EMP, DEPT, BONUS, and SALGRADE Between All Sites



Before You Start

In order for the script in this chapter to work as designed, it is assumed that the schema SCOTT exists at ORC1.WORLD (and optionally ORC2.WORLD and ORC3.WORLD) and contains the following objects:

- EMP
- DEPT
- BONUS
- SALGRADE

If you do not have the SCOTT schema at ORC1.WORLD or the SCOTT objects do not exist, you can run a script that comes with your Oracle database to create the sample schema SCOTT and the corresponding objects.

Complete the following:

1. Connect to ORC1.WORLD as user SYSTEM.

   ```
   CONNECT system/manager@orc1.world
   ```

2. If the SCOTT schema does exist, skip to Step 3. Otherwise, create the user SCOTT as illustrated below:

   ```
   CREATE USER scott IDENTIFIED BY tiger;
   ```

3. Run the `utlsampl.sql` script that is contained in your *`ORACLE_HOME`*`\rdbms\admin` directory.

   The schema SCOTT must exist, and be IDENTIFIED BY tiger, in order for this script to run properly. If it does not exist, be sure that you complete Step 2.

   Note: If you are running multiple database instances on the same computer, you may need to alter the CONNECT string contained within the `utlsampl.sql` script to contain the target database. For example, you would replace

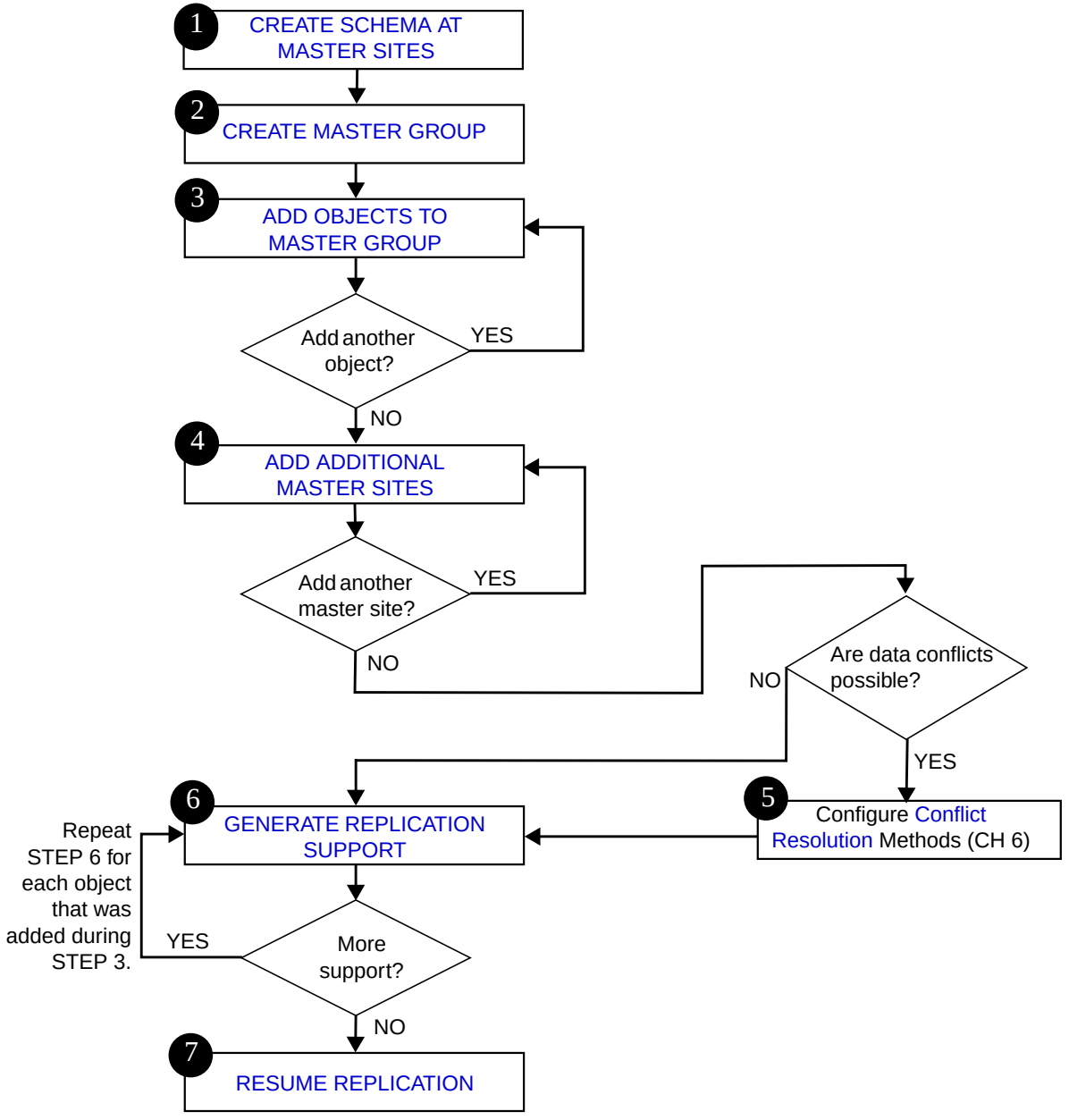
   ```
   CONNECT scott/tiger
   ```

   with

   ```
   CONNECT scott/tiger@orc1.world
   ```

After you have completed the three steps above, you have a "fresh" copy of the EMP, DEPT, BONUS, and SALGRADE tables.

Figure 3–2 Create Master Group



Create Master Group

```
/*************************************************************************
STEP 1:
CREATE SCHEMA AT MASTER SITES
*************************************************************************/

CONNECT system/manager@orc2.world;
CREATE USER scott IDENTIFIED BY tiger;
GRANT CONNECT, RESOURCE TO scott;

CONNECT system/manager@orc3.world;
CREATE USER scott IDENTIFIED BY tiger;
GRANT CONNECT, RESOURCE TO scott;

/*************************************************************************
STEP 2:
CREATE MASTER GROUP
*************************************************************************/

--Use the CREATE_MASTER_REPGROUP API to define a new master group.
--When you add an object to your master group or perform other replication
--administrative tasks, you reference the master group name defined
--during this step. The following must be executed by the replication
--administrator.

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPGROUP (
      GNAME => 'scott_mg');
END;
/

/*************************************************************************
STEP 3:
ADD OBJECTS TO MASTER GROUP
*************************************************************************/

--Use the CREATE_MASTER_REPOBJECT API to add an object to your master group.
--In most cases, you probably will be adding tables to your master group,
--but you can also add indexes, procedures, views, synonyms, and so on.  See
--CREATE_MASTER_REPOBJECT procedure on page 8-104 for additional
--information.
```

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'table',
      ONAME => 'emp',
      SNAME => 'scott',
      USE_EXISTING_OBJECT => TRUE,
      COPY_ROWS => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'table',
      ONAME => 'dept',
      SNAME => 'scott',
      USE_EXISTING_OBJECT => TRUE,
      COPY_ROWS => TRUE);
END;
/

--The tables EMP and DEPT have a primary key, but BONUS and SALGRADE do not have
--a primary key. For replication to work properly, each replicated table either
--needs a primary key or to have a "set column." The
--DBMS_REPCAT.SET_COLUMNS procedure is sufficient for multimaster replication
--only, but if you also want to support fast refreshable snapshots, you need a
--primary key. It is easier to alter your object before you add it to your
--master group.

ALTER TABLE scott.bonus ADD (CONSTRAINT bonus_pk PRIMARY KEY(ename));

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'table',
      ONAME => 'bonus',
      SNAME => 'scott',
      USE_EXISTING_OBJECT => TRUE,
      COPY_ROWS => TRUE);
END;
/
```

```
--You must modify the SCOTT.SALGRADE object just as you altered the
--SCOTT.BONUS object in the previous step.

ALTER TABLE scott.salgrade ADD (CONSTRAINT salgrade_pk PRIMARY KEY(grade));

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'table',
      ONAME => 'salgrade',
      SNAME => 'scott',
      USE_EXISTING_OBJECT => TRUE,
      COPY_ROWS => TRUE);
END;
/

/*****************************************************************************
STEP 4:
ADD ADDITIONAL MASTER SITES
*****************************************************************************/

--After you have defined your master group at the MASTERDEF site (the
--site where the master group was created becomes the MASTER DEFINITION
--site by default), you can define the other sites that will participate
--in the replicated environment. You might have guessed that you will be
--adding the ORC2.WORLD and ORC3.WORLD sites to our replicated environment.

BEGIN
   DBMS_REPCAT.ADD_MASTER_DATABASE (
      GNAME => 'scott_mg',
      MASTER => 'orc2.world',
      USE_EXISTING_OBJECTS => TRUE,
      COPY_ROWS => TRUE,
      PROPAGATION_MODE => 'ASYNCHRONOUS');
END;
/

/*****************************************************************************
NOTE: You should wait until ORC2.WORLD appears in the DBA_REPSITES view
before continuing. Execute the following SELECT statement in another
SQL*Plus session to make sure that ORC2.WORLD has appeared):

SELECT * FROM dba_repsites WHERE gname = 'scott_mg';
*****************************************************************************/
PAUSE Press <RETURN> to continue.
```

```
BEGIN
   DBMS_REPCAT.ADD_MASTER_DATABASE (
      GNAME => 'scott_mg',
      MASTER => 'orc3.world',
      USE_EXISTING_OBJECTS => TRUE,
      COPY_ROWS => TRUE,
      PROPAGATION_MODE => 'ASYNCHRONOUS');
END;
/

/*************************************************************************
NOTE: You should wait until ORC3.WORLD appears in the DBA_REPSITES view
before continuing. Execute the following SELECT statement in another
SQL*Plus session to make sure that ORC3.WORLD has appeared):

SELECT * FROM dba_repsites WHERE gname = 'scott_mg';
*************************************************************************/
PAUSE Press <RETURN> to continue.

/*************************************************************************
CAUTION:  If you added one or more tables to a master group during creation
of the group, do not resume replication activity immediately. First consider
the possibility of replication conflicts, and configure conflict resolution
for the replicated tables in the group. See Chapter 6, "Conflict Resolution"
for more information about configuring conflict resolution for master group
objects.
*************************************************************************/

/*************************************************************************
STEP 5:
GENERATE REPLICATION SUPPORT
*************************************************************************/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'dept',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'bonus',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'salgrade',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

/*************************************************************************
NOTE: You should wait until the DBA_REPCATLOG view is empty before
resuming master activity. Execute the following SELECT statement
to monitor your DBA_REPCATLOG view:

SELECT * FROM dba_repcatlog WHERE gname = 'scott_mg';
*************************************************************************/
PAUSE Press <RETURN> to continue.
```

```
/*****************************************************************************
STEP 6:
RESUME REPLICATION
*****************************************************************************/

--After you have completed creating your master group, adding replication
--objects, generating replication support, and adding additional master
--databases, you need to resume replication activity. The
--RESUME_MASTER_ACTIVITY procedure API "turns on" replication for
--the specified master group.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

4

Create Deployment Template

This chapter illustrates how to build a deployment template using the replication managment API. The following topics are discussed:

- Oracle Deployment Templates Concepts
- Build Deployment Template
- Package for Instantiation
- Instantiate Deployment Template

Before You Instantiate Deployment Templates:

- If you want to perform fast refreshes of the snapshots that are created by your deployment templates, make sure the master tables for the snapshots have a snapshot log. You also need to create a snapshot log for the master tables specified in a subquery of a snapshot. The following example shows how to create a snapshot log on the SCOTT.EMP master table:

  ```
  CREATE SNAPSHOT LOG ON scott.emp;
  ```

- If conflicts are possible at the master site due to activity at the snapshot sites you are creating, configure conflict resolution for the master tables of the snapshots before you instantiate a deployment template. See Chapter 6, "Conflict Resolution" for information about configuring conflict resolution.

Oracle Deployment Templates Concepts

Oracle offers deployment templates to allow the database administrator to package a snapshot environment for easy, custom, and secure distribution and installation. A deployment template can be simple (for example, it can contain a single snapshot with a fixed data set), or complex (for example, it can contain hundreds of snapshots with a dynamic data set based on one or more variables). The goal is to define the environment once and deploy the deployment template as often as necessary. Oracle deployment templates feature:

- Central control
- Repeated deployment of a snapshot environment
- Data subsetting at remote sites using template parameters
- Authorized user list to control template instantiation and data access

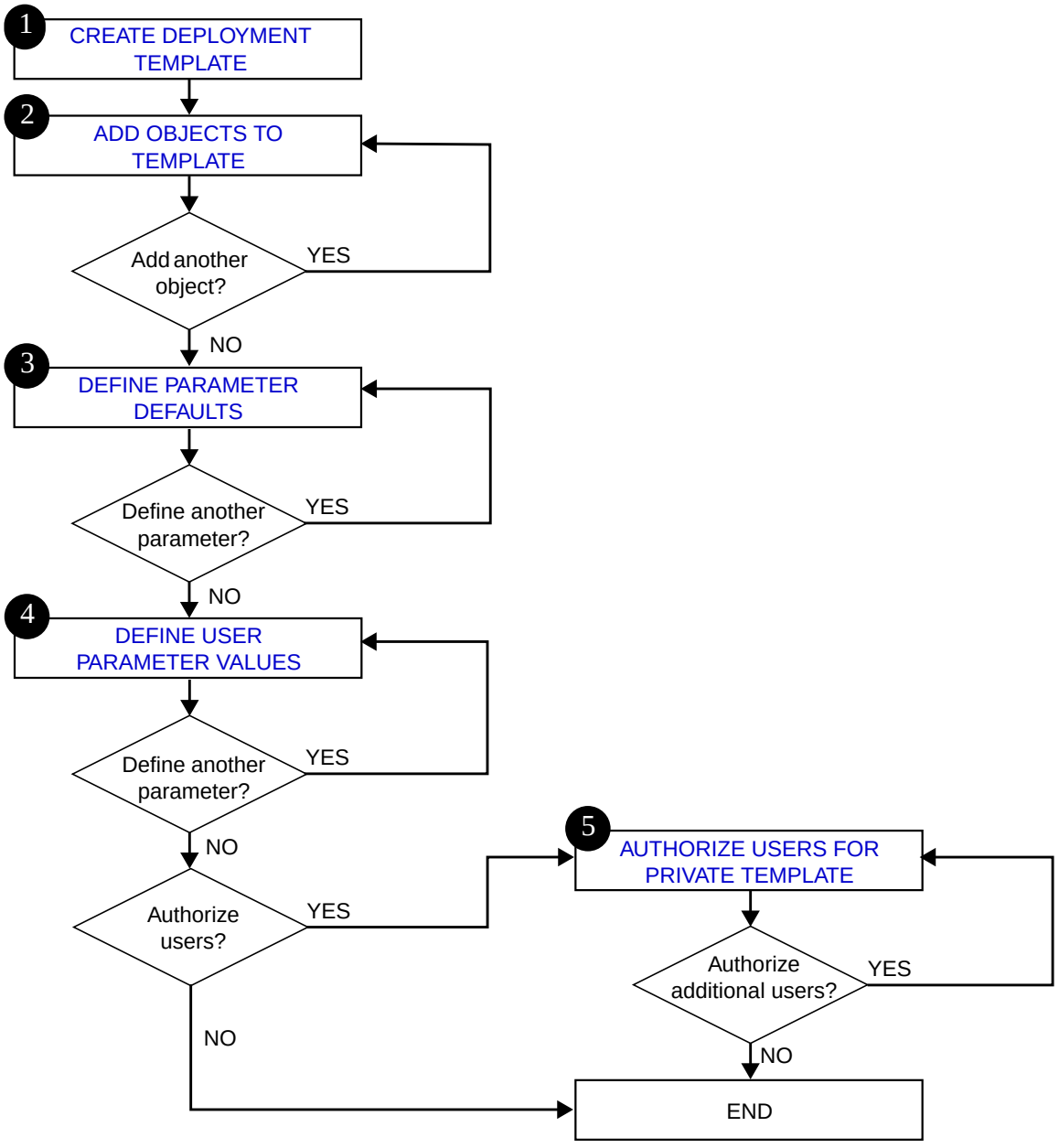
To prepare a snapshot environment for deployment, the DBA creates a *deployment template* at the master site. This template stores all of the information needed to deploy a snapshot environment, including the DDL to create the objects at the remote site and the target refresh group. This template also maintains links to user security information and template parameters for custom snapshot creation.

Build Deployment Template

This section contains a complete script example of how to construct a deployment template using the replication management API.

> **See Also:** Chapter 4, "Deployment Templates Concepts & Architecture" in *Oracle8i Replication* for conceptual and architectural information about deployment templates.

Figure 4–1 Create Deployment Template



Be sure to read the comments contained within the scripts, as they contain important and useful information about building templates with the replication management API.

> **Note:** Vertical partitioning is not supported using the replication management API. See "Vertical Partitioning" in *Oracle8i Replication* for more information.

```
--This script creates a private deployment template that contains
--four template objects, two template parameters, a set of user
--parameter values, and an authorized user.  A template is
--built in the following order:
--
--STEP 1: Define Refresh Group Template
--STEP 2: Add template objects to DT_PERSONNEL
--STEP 3: Define Parameter Defaults and Prompt Text
--STEP 4: Define User Parameter Values
--STEP 5: Authorize Users for Private Template


CONNECT repadmin/repadmin@orc3.world

/*************************************************************************
STEP 1:
CREATE DEPLOYMENT TEMPLATE
*************************************************************************/

--Before you begin assembling the components of your deployment
--template, use the CREATE_RERESH_TEMPLATE procedure to define the name of
--your deployment template, along with several other template characteristics
--(Public/Private status, target refresh group, and owner).

DECLARE
   a NUMBER;
BEGIN
   a := DBMS_REPCAT_RGT.CREATE_REFRESH_TEMPLATE (
           OWNER => 'scott',
           REFRESH_GROUP_NAME => 'personnel',
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           TEMPLATE_COMMENT => 'personnel deployment template',
           PUBLIC_TEMPLATE => 'N');
END;
/
```

```
/*************************************************************************
STEP 2:
ADD OBJECTS TO TEMPLATE
*************************************************************************/

--STEP 2a: Create EMP Snapshot

--The following procedure uses the DBMS_LOB package. This package is required
--to insert values into the DDL_TEXT parameter of the CREATE_TEMPLATE_OBJECT
--function, which has a CLOB datatype.  You will see the DBMS_LOB package
--used whenever a value must be inserted into a CLOB parameter. For more
--information about using the DBMS_LOB package and LOBs in general, see
--Oracle8i Application Developer's Guide - Fundamentals.

DECLARE
   tempstring VARCHAR2(300);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'CREATE SNAPSHOT scott.snap_emp AS SELECT
      empno, ename, job, mgr, hiredate, sal, comm, deptno
      FROM scott.emp@:dblink WHERE deptno = :dept';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           OBJECT_NAME => 'snap_emp',
           OBJECT_TYPE => 'SNAPSHOT',
           DDL_TEXT => templob,
           master_rollback_seg => 'RBS');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/

--Whenever you create a snapshot, always specify the schema name of the table
--owner in the query for the snapshot. In the example above, SCOTT is specified
--as the owner of the EMP table.
```

```
--STEP 2b: Create DEPT Snapshot

DECLARE
   tempstring VARCHAR2(300);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'CREATE SNAPSHOT scott.snap_dept AS SELECT
      deptno, dname, loc
      FROM scott.dept@:dblink';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           OBJECT_NAME => 'snap_dept',
           OBJECT_TYPE => 'SNAPSHOT',
           DDL_TEXT => templob,
           MASTER_ROLLBACK_SEG => 'RBS');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/

--STEP 2c: Create SALGRADE Snapshot

DECLARE
   tempstring VARCHAR2(300);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'CREATE SNAPSHOT scott.snap_salgrade AS SELECT
      grade, losal, hisal
      FROM scott.salgrade@:dblink';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           OBJECT_NAME => 'snap_salgrade',
           OBJECT_TYPE => 'SNAPSHOT',
           DDL_TEXT => templob,
           MASTER_ROLLBACK_SEG => 'RBS');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

```
--STEP 2d: Create BONUS Snapshot

DECLARE
   tempstring VARCHAR2(300);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'CREATE SNAPSHOT scott.snap_bonus AS SELECT
      ename, job, sal, comm
      FROM scott.bonus@:dblink';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           OBJECT_NAME => 'snap_bonus',
           OBJECT_TYPE => 'SNAPSHOT',
           DDL_TEXT => templob,
           MASTER_ROLLBACK_SEG => 'RBS');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/

/*************************************************************************
STEP 3:
DEFINE PARAMETER DEFAULTS
*************************************************************************/

--Rather than using the "CREATE" functions and procedures as in the
--other steps, you use the ALTER_TEMPLATE_PARM procedure to define
--a template parameter value and prompt string. You use the
--"ALTER" procedure because the actual parameter was created in
--step 2. Recall that you defined the :dblink and :dept parameters
--in the DDL_TEXT parameter. Oracle detects these parameters in
--the DDL and automatically creates the template parameter. Use
--the ALTER_TEMPLATE_PARM procedure to define the remainder of the
--template parameter information (that is, default parameter value
--and prompt string).
```

```
--STEP 3a: DEPT Parameter

DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := '20';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM (
      REFRESH_TEMPLATE_NAME => 'dt_personnel',
      PARAMETER_NAME => 'dept',
      NEW_DEFAULT_PARM_VALUE => templob,
      NEW_PROMPT_STRING => 'Enter your department number:',
      NEW_USER_OVERRIDE => 'Y');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/

--STEP 3b: DBLINK Parameter

DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'ORC2.WORLD';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM (
      REFRESH_TEMPLATE_NAME => 'dt_personnel',
      PARAMETER_NAME => 'dblink',
      NEW_DEFAULT_PARM_VALUE => templob,
      NEW_PROMPT_STRING => 'Enter target database link:',
      NEW_USER_OVERRIDE => 'N');
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

```
/*************************************************************************
STEP 4:
DEFINE USER PARAMETER VALUES
*************************************************************************/

--To automate the instantiation of custom data sets at
--individual remote snapshot sites, you can define USER
--PARAMETER values that will be used automatically when
--the specified user instantiates the target template.
--The CREATE_USER_PARM_VALUE procedure enables you to assign
--a value to a parameter for a user.

--STEP 4a: Define User Parameter Value for user SCOTT

DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := '30';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           PARAMETER_NAME => 'dept',
           USER_NAME => 'scott',
           PARM_VALUE => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/
```

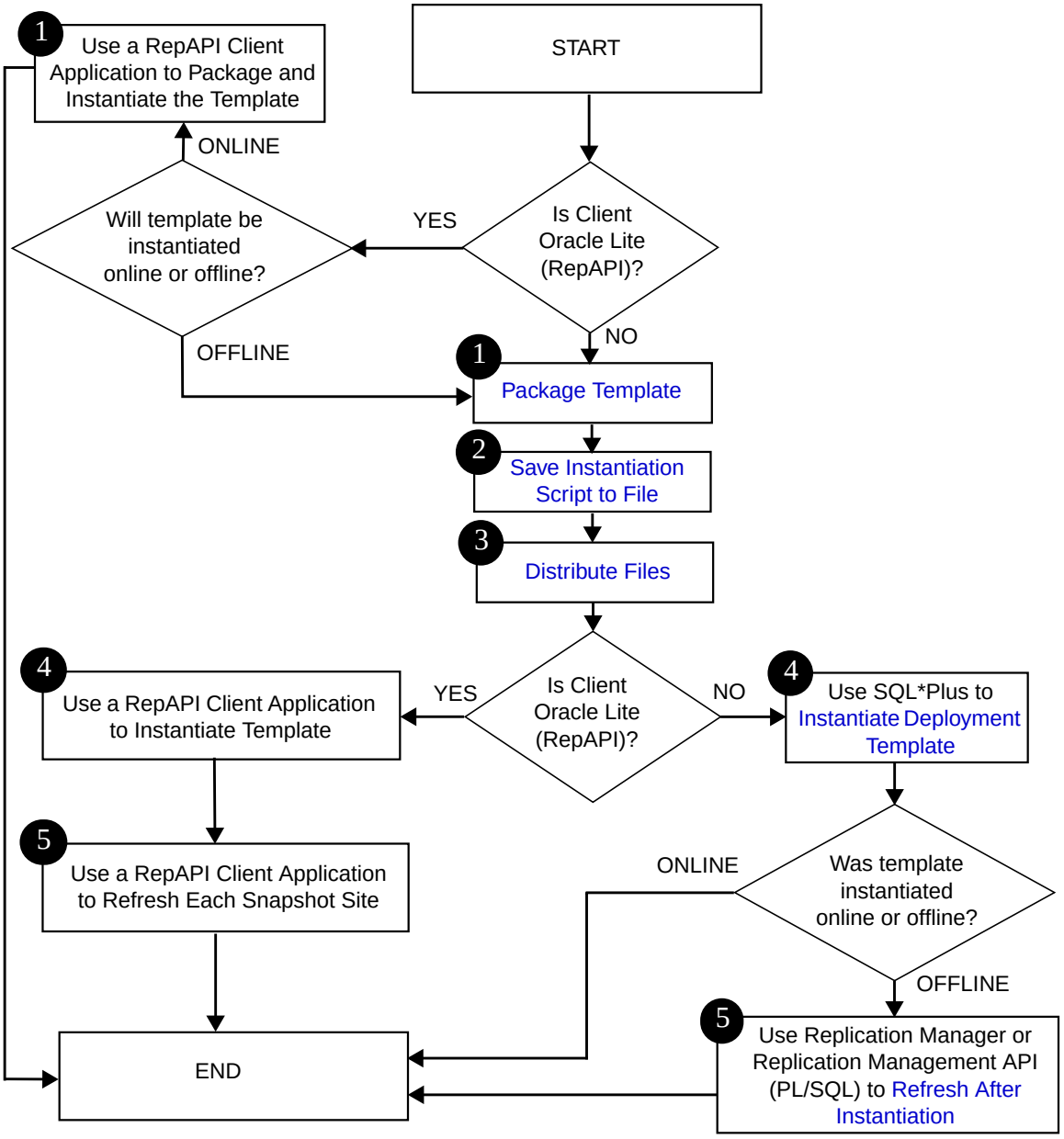
```
--STEP 4b: Define User Parameter Value for user SCOTT

DECLARE
   tempstring VARCHAR2(100);
   templob CLOB;
   a NUMBER;
BEGIN
   DBMS_LOB.CREATETEMPORARY(templob, TRUE, dbms_lob.session);
   tempstring := 'ORC2.WORLD';
   DBMS_LOB.WRITE(templob, length(tempstring), 1, tempstring);
   a := DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE (
           REFRESH_TEMPLATE_NAME => 'dt_personnel',
           PARAMETER_NAME => 'dblink',
           USER_NAME => 'scott',
           PARM_VALUE => templob);
   DBMS_LOB.FREETEMPORARY(templob);
END;
/

/*****************************************************************************
STEP 5:
AUTHORIZE USERS FOR PRIVATE TEMPLATE
*****************************************************************************/

--Because this is a private template (PUBLIC_TEMPLATE => 'N'
--in the DBMS_REPCAT_RGT.CREATE_REFRESH_TEMPLATE function
--defined in STEP 1), you need to authorize users to
--instantiate the DT_PERSONNEL deployment template. Use
--the DBMS_REPCAT_RGT.CREATE_USER_AUTHORIZATION function
--to create authorized users.

DECLARE
   a NUMBER;
BEGIN
   a := DBMS_REPCAT_RGT.CREATE_USER_AUTHORIZATION (
           USER_NAME => 'scott',
           REFRESH_TEMPLATE_NAME => 'dt_personnel');
END;
/

COMMIT;
```

Package for Instantiation

After you have completed building your deployment template, you need to package the template for instantiation. This example illustrates how to use both the online and offline instantiation procedures. Notice that the instantiation procedures are very similar: you simply use either the INSTANTIATE_ONLINE function or INSTANTIATE_OFFLINE function according to your needs. This section accomplishes two tasks: create the instantiation script and save the instantiation script to a file.

See Also: Some of the tasks in Figure 4–2 instruct you to use a RepAPI client application to package and / or instantiate a deployment template for an Oracle8*i* Lite snapshot site. To complete these tasks, use a client application such as the Oracle Client Replication Tool, which is supplied with Oracle8*i* Lite. See your Oracle8*i* Lite documentation for more information. Also, see Appendix C, "Configuring the Oracle8*i* Server for RepAPI" in *Oracle8i Replication* for instructions on configuring your server to support Java RepAPI clients.

Figure 4–2 Package and Instantiate Deployment Template



Package Template

When you execute either the INSTANTIATE_OFFLINE or the INSTANTIATE_ONLINE function, Oracle populates the USER_REPCAT_TEMP_OUTPUT view with the script to create the remote snapshot environment. Both online and offline scripts contain the SQL statements to create the objects specified in the deployment template. The difference is that an offline instantiation script also contains the data to populate the objects. The online instantiation script does not contain the data. Rather, during online instantiation, the snapshot site connects to the master site to download the data.

Complete the steps in either the "Offline Instantiation Package" or "Online Instantiation Package" according to your needs. These sections only apply to packaging templates for snapshot sites with Oracle8*i* Enterprise Edition, Oracle8*i* Standard Edition, or Oracle8*i* Personal Edition installed. These instructions do not apply to snapshot sites with Oracle8*i* Lite installed.

> **See Also:** Appendix C, "Configuring the Oracle8*i* Server for RepAPI" in *Oracle8i Replication* for instructions on packaging a deployment template for offline instantiation at a snapshot site with Oracle8*i* Lite installed. See the Oracle8*i* Lite documentation for information about using the a client application, such as the Oracle Client Replication Tool, to package and instantiate a deployment template online.

Offline Instantiation Package

The INSTANTIATE_OFFLINE function creates a script that creates the snapshot environment according to the contents of a specified deployment template. In addition to containing the DDL (CREATE statements) to create the snapshot environment, this script also contains the DML (INSERT statements) to populate the snapshot environment with the appropriate data set.

> **Note:** If you are packaging your template at the same master site that contains the target master objects for your deployment template, you must create a loopback database link.

```
--Use the INSTANTIATE_OFFLINE function to package the
--template for offline instantiation by a remote snapshot
--site. Executing this procedure both creates a script that
--creates that snapshot environment and populates the
--environment with the proper data set. This script is stored
--in the temporary USER_REPCAT_TEMP_OUTPUT view.
```

```
SET SERVEROUTPUT ON
DECLARE
   dt_num NUMBER;
BEGIN
   dt_num := DBMS_REPCAT_RGT.INSTANTIATE_OFFLINE(
               REFRESH_TEMPLATE_NAME => 'dt_personnel',
               USER_NAME => 'scott',
               SITE_NAME => 'la_regional',
               NEXT_DATE => SYSDATE,
               INTERVAL => 'SYSDATE + (1/144)');
   DBMS_OUTPUT.PUT_LINE('Template ID = ' || dt_num);
END;
/
COMMIT;
/
```

Make a note of the number that is returned for the DT_NUM variable. You must use this number when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated script. Be sure that you complete the steps in "Save Instantiation Script to File" after you complete this section. This script is unique to an individual snapshot site and cannot be used for other snapshot sites.

Online Instantiation Package

The INSTANTIATE_ONLINE function creates a script that creates the snapshot environment according to the contents of a specified deployment template. When this script is executed at the remote snapshot site, Oracle creates the snapshot site according to the DDL (CREATE statements) in the script and populates the environment with the appropriate data set from the master site. This requires that the remote snapshot site has a "live" connection to the master site.

> **See Also:** Chapter 7, "Planning Your Replication Environment" in *Oracle8i Replication* for additional snapshot site requirements.

```
--Use the INSTANTIATE_ONLINE function to "package" the
--template for online instantiation by a remote snapshot
--site. Executing this procedure creates a script which can
--then be used to create a snapshot environment. This script
--is stored in the temporary USER_REPCAT_TEMP_OUTPUT view.
```

```
SET SERVEROUTPUT ON
DECLARE
   dt_num NUMBER;
BEGIN
   dt_num := DBMS_REPCAT_RGT.INSTANTIATE_ONLINE(
               REFRESH_TEMPLATE_NAME => 'dt_personnel',
               USER_NAME => 'scott',
               SITE_NAME => 'snap1.world',
               NEXT_DATE => SYSDATE,
               INTERVAL => 'SYSDATE + (1/144)');
   DBMS_OUTPUT.PUT_LINE('Template ID = ' || dt_num);
END;
/
COMMIT;
/
```

Make a note of the number that is returned for the DT_NUM variable. You must use this number when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated script. Be sure that you complete the steps in "Save Instantiation Script to File" after you complete this section.

Save Instantiation Script to File

The easiest way to save the contents of the USER_REPCAT_TEMP_OUTPUT view is to use the SQL*Plus spool feature to save the results of a SELECT statement. Complete the following steps to save your deployment template script to a file:

> **Note:** The following steps must be performed immediately after you have called either the INSTANTIATE_OFFLINE or INSTANTIATE_ONLINE functions, because the contents of the USER_REPCAT_TEMP_OUTPUT view are temporary. If you have not completed the steps in "Package Template" on page 4-13, do so now and then complete the following steps.

1. Enter SPOOL *filename*.sql, where *filename* is the name of your script. Because you may have to generate many instantiation files, make sure you name your files with an easily recognizable name. For example, you might enter:

   ```
   SQL> SPOOL d:\snap1_world.sql
   ```

Your instantiation script is saved as `snap1_world.sql` in the Oracle home directory, unless otherwise specified, as in the example above. If necessary, precede the filename with a fully qualified path to save the script to a different directory.

2. Issue the following select statement:

```
SQL> SELECT DBMS_REPCAT_RGT.VC2_FROM_CLOB(text) text
       FROM user_repcat_temp_output
       WHERE output_id = dt_num ORDER BY LINE;
```

Here, *dt_num* is the value that was returned when you executed the INSTANTIATE_ONLINE or INSTANTIATE_OFFLINE functions (illustrated in "Package Template" on page 4-13).

3. Enter the following to stop spooling:

```
SQL> SPOOL OFF
```

The file that you specified in Step 1 is saved in the directory specified. This file contains the script required to build the snapshot environment.

Distribute Files

After you have created the instantiation script and saved it to a file, you must distribute this file to the remote snapshot sites that need to instantiate the template. You can distribute this file by posting the file on an FTP site or saving the file to a CD-ROM, floppy disk, or other distribution medium.

Instantiate Deployment Template

After the instantiation script has been distributed to the remote snapshot sites, you are ready to instantiate the deployment template at the remote snapshot site.

> **See Also:** Chapter 7, "Planning Your Replication Environment" in *Oracle8i Replication* for snapshot site requirements that must be met before instantiating your deployment template.

The following script demonstrates how to complete the instantiation process at a remote snapshot site with Oracle8*i* Enterprise Edition, Oracle8*i* Standard Edition, or Oracle8*i* Personal Edition installed. These instructions do not apply to snapshot sites with Oracle8*i* Lite installed. Instead, Oracle8*i* Lite snapshot sites use a client application, such as the Oracle Client Replication Tool, to instantiate deployment templates.

> **See Also:** See the Oracle8*i* Lite documentation for information about instantiating deployment templates at Oracle8*i* Lite snapshot sites.

```
/*************************************************************************
STEP 1:
CREATE SCHEMA AND DATABASE LINKS
*************************************************************************/

--Before you execute the instantiation script at the remote snapshot site,
--you must create the schema that contains the replicated objects. Use
--SQL*Plus to complete Step 1 and Step 2.

CONNECT system/manager@snap1.world

CREATE USER scott IDENTIFIED BY tiger;

GRANT CONNECT, RESOURCE TO scott;

--Before you can create the private database link, you must create a public
--database link.

CREATE PUBLIC DATABASE LINK orc3.world USING 'orc3.world';
```

```
--Connect as the target user (scott) and create a private database link
--to the target master site (the target user must also exist at the master
--site).

CONNECT scott/tiger@snap1.world

CREATE DATABASE LINK orc3.world
   CONNECT TO scott IDENTIFIED BY tiger;

/*************************************************************************
STEP 2:
EXECUTE THE INSTANTIATION SCRIPT
*************************************************************************/

@d:\snap1_world.sql
```

Depending on the size of the snapshot environment created and the amount of data loaded, the instantiation procedure may take a substantial amount of time.

Refresh After Instantiation

If you have just instantiated a deployment template using the offline instantiation method, you should perform a refresh as soon as possible. Issue the following execute statement:

```
EXECUTE DBMS_REFRESH.REFRESH('personnel');
```

5

Create Snapshot Group

This chapter illustrates how to create a snapshot group at a remote snapshot replication site. The following topics are discussed:

- Creating a Snapshot Group Overview
- Create Snapshot Group

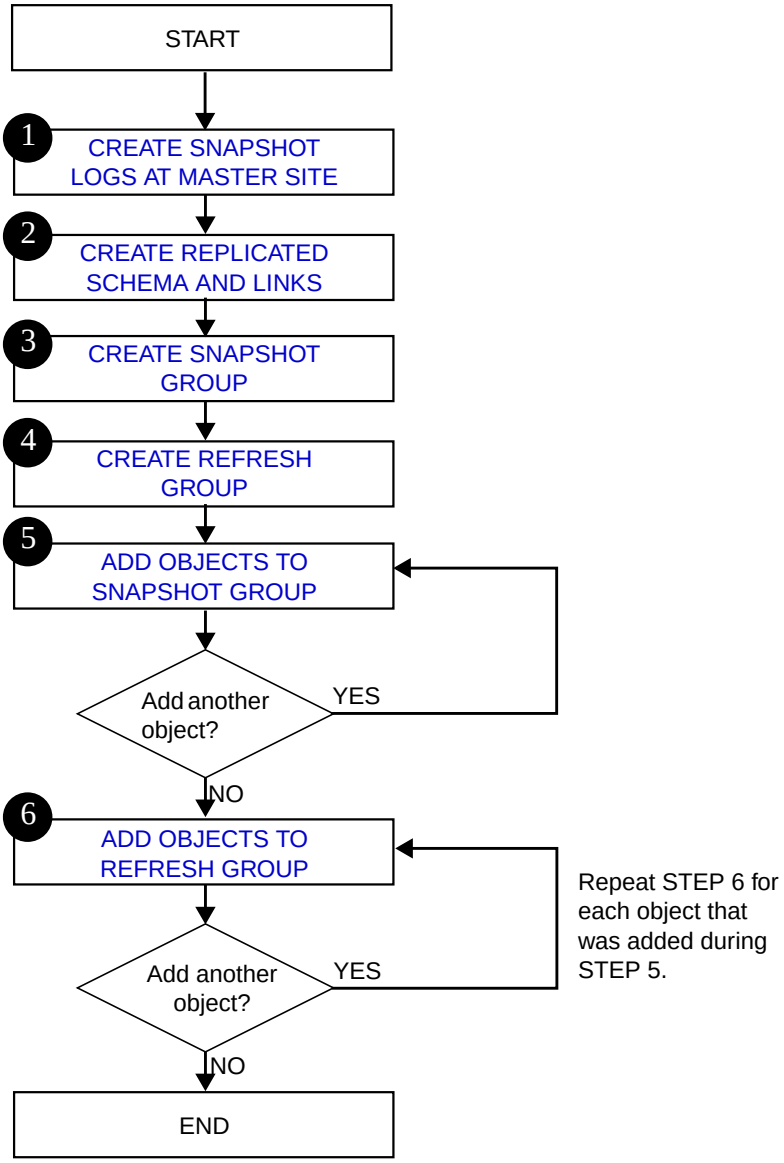
Note: If conflicts are possible at the master site because of activity at the snapshot sites you are creating, configure conflict resolution for the master tables. See Chapter 6, "Conflict Resolution" for information about configuring conflict resolution.

Creating a Snapshot Group Overview

After you have set up your snapshot site and have created at least one master group, you are ready to begin creating a snapshot group at the remote snapshot site. Figure 5–1 illustrates the process of creating a snapshot group.

> **See Also:** Chapter 2, "Create Replication Site" for information about setting up a snapshot site, and see Chapter 3, "Create a Master Group" for information about creating a master group.

Figure 5–1 Set Up Snapshot Group



Create Snapshot Group

```
/*************************************************************************
STEP 1:
CREATE SNAPSHOT LOGS AT MASTER SITE

See the CREATE SNAPSHOT LOG in the Oracle8i SQL Reference for
detailed information about this SQL statement.
*************************************************************************/

--If you want one of your master sites to support a snapshot site, then
--you need to create snapshot logs for each master table that is
--replicated to a snapshot. Recall from Figure 2–1 on page 2-2 that
--ORC1.WORLD serves as the target master site for the SNAP1.WORLD
--snapshot site. The required snapshot logs must be created at ORC1.WORLD.

CONNECT scott/tiger@orc1.world

CREATE SNAPSHOT LOG ON scott.emp;
CREATE SNAPSHOT LOG ON scott.dept;
CREATE SNAPSHOT LOG ON scott.bonus;
CREATE SNAPSHOT LOG ON scott.salgrade;


/*************************************************************************
STEP 2:
CREATE REPLICATED SCHEMA AND LINKS
*************************************************************************/

--Before you begin building your snapshot group, you must make sure that
--the replicated schema exists at the remote snapshot site and that the
--necessary database links have been created.

CONNECT system/manager@snap1.world

CREATE USER scott IDENTIFIED BY tiger;
GRANT connect, resource TO scott;

CONNECT scott/tiger@snap1.world
```

```
--The owner of the snapshots needs a database link pointing to the
--proxy_refresher that was created when the snapshot site was set up; see
--"CREATE MASTER SITE USERS" on page 2-6 for information.

CREATE DATABASE LINK orc1.world
   CONNECT TO proxy_refresher IDENTIFIED BY proxy_refresher;

/*************************************************************************
STEP 3:
CREATE SNAPSHOT GROUP
*************************************************************************/

--The following procedures must be executed by the snapshot administrator
--at the remote snapshot site.

CONNECT snapadmin/snapadmin@snap1.world

--The master group that you specify in the GNAME parameter must match the
--name of the master group that you are replicating at the target master site.

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP (
      GNAME => 'scott_mg',
      MASTER => 'orc1.world',
      PROPAGATION_MODE => 'ASYNCHRONOUS');
END;
/
```

```
/*************************************************************************
STEP 4:
CREATE REFRESH GROUP
*************************************************************************/

--All snapshots that are added to a particular refresh group are
--refreshed at the same time. This ensures transactional consistency
--between the related snapshots in the refresh group.

BEGIN
   DBMS_REFRESH.MAKE (
      NAME => 'snapadmin.scott_rg',
      LIST => '',
      NEXT_DATE => SYSDATE,
      INTERVAL => 'SYSDATE + 1/24',
      IMPLICIT_DESTROY => FALSE,
      ROLLBACK_SEG => '',
      PUSH_DEFERRED_RPC => TRUE,
      REFRESH_AFTER_ERRORS => FALSE);
END;
/

/*************************************************************************
STEP 5:
ADD OBJECTS TO SNAPSHOT GROUP
*************************************************************************/

--Whenever you create a snapshot, always specify the schema name of the table
--owner in the query for the snapshot. In the examples below, SCOTT is specified
--as the owner of the table in each query.

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'bonus',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.bonus REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.bonus@orc1.world',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'dept',
      TYPE => 'SNAPSHOT',
      ddl_text => 'CREATE SNAPSHOT scott.dept REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.dept@orc1.world',
      MIN_COMMUNICATION => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.emp REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.emp@orc1.world',
      MIN_COMMUNICATION => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'salgrade',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.salgrade REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.salgrade@orc1.world',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
/*************************************************************************
STEP 6:
ADD OBJECTS TO REFRESH GROUP
*************************************************************************/

--All of the snapshot group objects that you add to the refresh group
--are refreshed at the same time to preserve referential integrity
--between related snapshots.

BEGIN
   DBMS_REFRESH.ADD (
      NAME => 'snapadmin.scott_rg',
      LIST => 'scott.bonus',
      LAX => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      NAME => 'snapadmin.scott_rg',
      LIST => 'scott.dept',
      LAX => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      NAME => 'snapadmin.scott_rg',
      LIST => 'scott.emp',
      LAX => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      NAME => 'snapadmin.scott_rg',
      LIST => 'scott.salgrade',
      LAX => TRUE);
END;
/
```

6

Conflict Resolution

This chapter illustrates how to define conflict resolution methods for your replicated environment. The following topics are discussed:

- Prepare for Conflict Resolution
- Create Conflict Resolution Methods for Update Conflicts
- Create Conflict Resolution Methods for Uniqueness Conflicts
- Create Conflict Avoidance Methods for Delete Conflicts
- Audit Successful Conflict Resolution

Prepare for Conflict Resolution

Though you may take great care in designing your database and front-end application to avoid conflicts that may arise between multiple sites in a replicated environment, you may not be able to completely eliminate the possibility of conflicts. One of the most important aspects of replication is to ensure data convergence at all sites participating in the replicated environment.

When data conflicts occur, you need a mechanism to ensure that the conflict is resolved in accordance with your business rules and that the data converges correctly at all sites.

Oracle replication lets you define a conflict resolution system for your database that resolves conflicts in accordance with your business rules. If you have a unique situation that Oracle's pre-built conflict resolution methods cannot resolve, you have the option of building and using your own conflict resolution methods.

> **See Also:** *Oracle8i Replication* for conceptual information about conflict resolution methods and detailed information about data convergence for each method.

Plan

Before you begin implementing conflict resolution methods for your replicated tables, analyze the data in your system to determine where the most conflicts may occur. For example, static data such as an employee number may change very infrequently and is not subject to a high occurrence of conflicts. An employee's customer assignments, however, may change often and would therefore be prone to data conflicts.

Once you have determined where the conflicts are most likely to occur, you need to determine how to resolve the conflict. For example, do you want the latest change to have precedence, or should one site over another have precedence?

As you read each of the sections describing the different conflict resolution methods, you will learn what each method is best suited for. So, read each section and then think about how your business would want to resolve any potential conflicts.

After you have identified the potential problem areas and have determined what business rules would resolve the problem, use Oracle's conflict resolution methods (or one of your own) to implement a conflict resolution system.

Create Conflict Resolution Methods for Update Conflicts

The most common data conflict occurs when the same row at two or more different sites were updated at the same time, or before the deferred transaction from one site was successfully propagated to the other sites.

One method to avoid update conflicts is to implement a synchronous replicated environment, though this solution requires large network resource.

The other solution is to use the Oracle conflict resolution methods to deal with update conflicts that may occur when the same row has received two or more updates.

Overwrite and Discard

The overwrite and discard methods ignore the values from either the originating or destination site and therefore can never guarantee convergence with more than one master site. These methods are designed to be used by a single master site and multiple snapshot sites, or with some form of a user-defined notification facility.

The overwrite method replaces the current value at the destination site with the new value from the originating site. Conversely, the discard method ignores the new value from the originating site.

> **See Also:** "ADD_conflicttype_RESOLUTION procedure" on page 8-78 and "Overwrite and Discard" in *Oracle8i Replication* for more information.

Note: This section uses objects not found in the other scripts within this book, because the configuration ORC1.WORLD, ORC2.WORLD, ORC3.WORLD, and SNAP1.WORLD contains three master sites and one snapshot site and is not appropriate for OVERWRITE and DISCARD.

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@saturn.universe

--Before you can define any conflict resolution methods, quiesce the
--master group that contains the table to which you want to apply the
--conflict resolution method.

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'titan_mg');
END;
/

--All Oracle conflict resolution methods are based on logical column groupings
--called "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'titan',
      ONAME => 'planet',
      COLUMN_GROUP => 'planet_cg1',
      LIST_OF_COLUMN_NAMES => 'order,circumference,moons');
END;
/

--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution method for a specified table. This example creates an
--"Overwrite" conflict resolution method.

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'titan',
      ONAME => 'planet',
      COLUMN_GROUP => 'planet_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'OVERWRITE',
      PARAMETER_COLUMN_NAME => 'order,circumference,moons');
END;
/
```

```
--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'titan',
      ONAME => 'planet',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'titan_mg');
END;
/
```

Minimum and Maximum

When the advanced replication facility detects a conflict with a column group and calls either the *minimum* or *maximum* value conflict resolution methods, it compares the new value from the originating site with the current value from the destination site for a designated column in the column group. You must designate this column when you define your conflict resolution method.

If the new value of the designated column is *less than* or *greater than* (depending on the method used) the current value, the column group values from the originating site are applied at the destination site, assuming that all other errors were successfully resolved for the row. Otherwise the rows remain unchanged.

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world

--Before you can define any conflict resolution methods, quiesce the
--master group that contains the table to which you want to apply the
--conflict resolution method.
```

```
BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--All Oracle conflict resolution methods are based on logical column groupings
--called "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'salgrade',
      COLUMN_GROUP => 'salgrade_cg1',
      LIST_OF_COLUMN_NAMES => 'losal');
END;
/

--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution method for a specified table. This example creates a
--"MINIMUM" conflict resolution method.

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'salgrade',
      COLUMN_GROUP => 'salgrade_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'MINIMUM',
      PARAMETER_COLUMN_NAME => 'losal');
END;
/

--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.
```

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'salgrade',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Timestamp

The *earliest timestamp* and *latest timestamp* methods are variations on the minimum and maximum value methods. To use the timestamp method, you must designate a column in the replicated table of type DATE. When an application updates any column in a column group, the application must also update the value of the designated timestamp column with the local SYSDATE. For a change applied from another site, the timestamp value should be set to the timestamp value from the originating site.

Several elements are needed to make timestamp conflict resolution work well:

- Synchronized Time Settings Between Computers
- Timestamp field and trigger to automatically record timestamp

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world

--Before you can define any conflict resolution methods, quiesce the
--master group that contains the table to which you want to apply the
--conflict resolution method.
```

```
BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--If the target table does not already contain a timestamp field,
--then add an additional column to your table to record the
--timestamp value when a row is inserted or updated. Additionally,
--you must use the ALTER_MASTER_REPOBJECT API to apply the DDL to
--the target table. Simply issuing the DDL may cause the replicated
--object to become invalid.

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD (timestamp DATE)');
END;
/

--After you have inserted a new column into your replicated object,
--make sure that you re-generate replication support for the
--affected object. This step should be performed immmediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--Once the timestamp field has been created, create a trigger
--that records the timestamp of when a row is either inserted
--or updated. This recorded value is used in the resolution of
--conflicts based on the Timestamp method. Instead of directly executing the
--DDL, you should use the DBMS_REPCAT.CREATE_MASTER_REPOBJECT procedure to
--create the trigger and add it to your master group.
```

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'TRIGGER',
      ONAME => 'insert_time',
      SNAME => 'scott',
      DDL_TEXT => 'CREATE TRIGGER scott.insert_time
                      BEFORE
                         INSERT OR UPDATE ON scott.emp FOR EACH ROW
                      BEGIN
                         IF DBMS_REPUTIL.FROM_REMOTE = FALSE THEN
                            :NEW.TIMESTAMP := SYSDATE;
                         END IF;
                      END;');
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'insert_time',
      TYPE => 'TRIGGER',
      MIN_COMMUNICATION => TRUE);
END;
/

--All Oracle conflict resolution methods are based on logical column groupings
--called "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      LIST_OF_COLUMN_NAMES => 'mgr, hiredate, sal, timestamp');
END;
/
```

```
--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution method for a specified table. This example specifies the
--"LATEST TIMESTAMP" conflict resolution method using the TIMESTAMP column
--that you created earlier.

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'LATEST TIMESTAMP',
      PARAMETER_COLUMN_NAME => 'timestamp');
END;
/

--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Additive and Average

The *additive* and *average* methods work with column groups consisting of a single numeric column only. Instead of "accepting" one value over another, this conflict resolution method either adds the two compared values together or takes an average of the two compared values.

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world

--Before you can define any conflict resolution methods, quiesce the
--master group that contains the table to which you want to apply the
--conflict resolution method.

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--All Oracle conflict resolution methods are based on logical column groupings
--called "column groups."  Create a column group for your target table by using
--the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'bonus',
      COLUMN_GROUP => 'bonus_cg1',
      LIST_OF_COLUMN_NAMES => 'sal');
END;
/

--Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION API to define the conflict
--resolution method for a specified table. This example specifies the
--"ADDITIVE" conflict resolution method using the SAL column.
```

```
BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'bonus',
      COLUMN_GROUP => 'bonus_cg1',
      SEQUENCE_NO => 1,
      METHOD => 'ADDITIVE',
      PARAMETER_COLUMN_NAME => 'sal');
END;
/

--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'bonus',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Priority Groups

Priority groups allow you to assign a priority level to each possible value of a particular column. If Oracle detects a conflict, Oracle updates the table whose "priority" column has a lower value using the data from the table with the higher priority value.

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--Make sure that the JOB field is part of the column group that your
--site priority conflict resolution mechanism is used for. Use the
--ADD_GROUPED_COLUMN procedure to add this field to an existing column group.
--If you do not already have a column group, you can create a new column group
--using the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      LIST_OF_COLUMN_NAMES => 'mgr, hiredate, sal, job');
END;
/

--Before you begin assigning a priority value to the values in your table, you
--must create a priority group that "holds" the values that you defined.

BEGIN
   DBMS_REPCAT.DEFINE_PRIORITY_GROUP (
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      DATATYPE => 'VARCHAR2');
END;
/
```

```
--The DBMS_REPCAT.ADD_PRIORITY_datatype procedure is avaiable in several
--different versions. There is a version for each available datatype
--(NUMBER, VARCHAR2, and so on).  See "ADD_PRIORITY_datatype procedure"
-- on page 8-75 for more information.  Execute this API as often as
--necessary until you have defined a priority value for all possible
--table values.

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'president',
      PRIORITY => 100);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'manager',
      PRIORITY => 80);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'salesman',
      PRIORITY => 60);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'analyst',
      PRIORITY => 40);
END;
/
```

```
BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      GNAME => 'scott_mg',
      PGROUP => 'job_pg',
      VALUE => 'clerk',
      PRIORITY => 20);
END;
/

--After you have completed assigning your priority values, add the
--PRIORITY GROUP resolution method to your replicated table. The following API
--example shows that it is the second conflict resolution method for the
--specified column group (SEQUENCE_NO).

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      SEQUENCE_NO => 2,
      METHOD => 'PRIORITY GROUP',
      PARAMETER_COLUMN_NAME => 'job',
      PRIORITY_GROUP => 'job_pg');
END;
/

--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Site Priority

Site priority is a specialized form of priority groups. Therefore, many of the procedures associated with site priority behave similarly to the procedures associated with priority groups. Instead of resolving a conflict based on the priority of a field's value, the conflict is resolved based on the priority of the sites involved.

For example, if you assign ORC2.WORLD a higher priority value than ORC1.WORLD and a conflict arises between these two sites, the value from ORC2.WORLD is used.

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--You must add a SITE column to your table to store the site value in
--your replicated table. Use the DBMS_REPCAT.ALTER_MASTER_REPOBJECT procedure
--to apply the DDL to the target table. Simply issuing the DDL may cause
--the replicated object to become invalid.

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD (site VARCHAR2(20))');
END;
/
```

```
--After you have inserted a new column into your replicated object,
--make sure that you re-generate replication support for the
--affected object. This step should be performed immmediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After you have added the SITE column to your table, make sure
--that this field is part of the column group that your site
--priority conflict resolution mechanism is used for. Use the
--ADD_GROUPED_COLUMN procedure to add this field to an existing
--column group. If you do not already have a column group, you can create a
--new column group using the DBMS_REPCAT.MAKE_COLUMN_GROUP procedure.

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      LIST_OF_COLUMN_NAMES => 'mgr, hiredate, sal, site');
END;
/

--Before you begin assigning a site priority value to the sites in your
--replicated environment, you must create a site priority group that "holds"
--the values that you defined.

BEGIN
   DBMS_REPCAT.DEFINE_SITE_PRIORITY (
      GNAME => 'scott_mg',
      NAME => 'site_pg');
END;
/

--Define the priority value for each of the sites in your replication
--environment using the DBMS_REPCAT.ADD_SITE_PRIORITY_SITE procedure.
--Execute this API as often as necessary until you have defined a site
--priority value for each of the sites in our replication environment.
```

```
BEGIN
   DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
      GNAME => 'scott_mg',
      NAME => 'site_pg',
      SITE => 'orc1.world',
      PRIORITY => 100);
END;
/

BEGIN
   DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
      GNAME => 'scott_mg',
      NAME => 'site_pg',
      SITE => 'orc2.world',
      PRIORITY => 50);
END;
/

BEGIN
   DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
      GNAME => 'scott_mg',
      NAME => 'site_pg',
      SITE => 'orc3.world',
      PRIORITY => 25);
END;
/

--After you have completed assigning your site priority values, add the
--SITE PRIORITY resolution method to your replicated table. The following
--API examples shows that it is the third conflict resolution method
--for the specified column group (SEQUENCE_NO).

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      SNAME => 'scott',
      ONAME => 'emp',
      COLUMN_GROUP => 'emp_cg1',
      SEQUENCE_NO => 3,
      METHOD => 'site priority',
      PARAMETER_COLUMN_NAME => 'site',
      PRIORITY_GROUP => 'site_pg');
END;
/
```

```
--After you have defined your conflict resolution method, regenerate
--replication support for the table that received the conflict
--resolution method.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Create Conflict Resolution Methods for Uniqueness Conflicts

In a replicated environment, you may encounter situations where you receive a conflict on a unique constraint, often resulting from an insert. If your business rules allow you to delete the duplicate row, you can define such resolution with Oracle's pre-built conflict resolution methods.

More often, however, you probably want to modify the conflicting value so that it no longer violates the unique constraint. Modifying the conflicting value ensures that you do not lose important data. Oracle's pre-built uniqueness conflict resolution method can make the conflicting value unique by appending a site name or a sequence number to the value.

An additional component that accompanies uniqueness conflict resolution methods is a notification facility. The conflicting information is modified by Oracle so that it can be inserted into the table, but you should be notified so that you can analyze the conflict to determine whether the record should be deleted, or the data merged into another record, or a completely new value be defined for the conflicting data.

```
--The following procedures need to be executed by the replication administrator.

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

--As you might expect, a uniqueness conflict resolution method detects and
--resolves conflicts encountered on columns with a UNIQUE constraint. Use
--the ALTER_MASTER_REPOBJECT procedure (described on page 8-83) to add
--a UNIQUE constraint to the EMP table.

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD
                  (constraint emp_ename_unique UNIQUE(ename))');
END;
/

--After you have added the UNIQUE constraint to your replicated table,
--make sure that you regenerate replication support for
--the affected table. This step should be performed immmediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
--The following table (conf_report) stores the messages received from
--your notification facility.

BEGIN
   DBMS_REPCAT.EXECUTE_DDL(
      GNAME => 'scott_mg',
      DDL_TEXT => 'CREATE TABLE scott.conf_report (
                   line NUMBER(2),
                   txt VARCHAR2(80),
                   timestamp DATE,
                   table_name VARCHAR2(30),
                   table_owner VARCHAR2(30),
                   conflict_type VARCHAR2(7))');
END;
/

CONNECT scott/tiger@orc1.world

--The following package (notify) sends a notification to the CONF_REPORT
--table when a conflict is detected.

--The conflict resolution notification package that is created in this script is
--described in detail in Appendix B, "User-Defined Conflict Resolution Methods".

CREATE OR REPLACE PACKAGE notify AS
   FUNCTION emp_unique_violation (ename IN OUT VARCHAR2,
      discard_new_values IN OUT BOOLEAN)
   RETURN BOOLEAN;
END notify;
/
```

```
CREATE OR REPLACE PACKAGE BODY notify AS
   TYPE message_table IS TABLE OF VARCHAR2(80) INDEX BY BINARY_INTEGER;
   PROCEDURE report_conflict(conflict_report IN MESSAGE_TABLE,
      report_length IN NUMBER,
      conflict_time IN DATE,
      conflict_table IN VARCHAR2,
      table_owner IN VARCHAR2,
      conflict_type IN VARCHAR2) IS
      BEGIN
         FOR idx IN 1..report_length LOOP
            BEGIN
               INSERT INTO scott.conf_report
               (line, txt, timestamp, table_name, table_owner, conflict_type)
               VALUES (idx, SUBSTR(conflict_report(idx),1,80), conflict_time,
               conflict_table, table_owner, conflict_type);
            EXCEPTION WHEN others THEN NULL;
            END;
         END LOOP;
      END report_conflict;
   FUNCTION emp_unique_violation(ename IN OUT VARCHAR2,
      discard_new_values IN OUT BOOLEAN)
   RETURN BOOLEAN IS
      local_node VARCHAR2(128);
      conf_report MESSAGE_TABLE;
      conf_time DATE := SYSDATE;
   BEGIN
      BEGIN
         SELECT global_name INTO local_node FROM global_name;
      EXCEPTION WHEN others THEN local_node := '?';
      END;
      conf_report(1) := 'UNIQUENESS CONFLICT DETECTED IN TABLE EMP ON ' ||
         TO_CHAR(conf_time, 'MM-DD-YYYY HH24:MI:SS');
      conf_report(2) := ' AT NODE ' || local_node;
      conf_report(3) := 'ATTEMPTING TO RESOLVE CONFLICT USING' ||
         ' APPEND SITE NAME METHOD';
      conf_report(4) := 'ENAME: ' || ename;
      conf_report(5) := NULL;
      report_conflict(conf_report, 5, conf_time, 'EMP', 'SCOTT', 'UNIQUE');
      discard_new_values := FALSE;
      RETURN FALSE;
   END emp_unique_violation;
END notify;
/
```

```
CONNECT repadmin/repadmin@orc1.world

--The following package is replicated to all of the master sites in your
--replication environment, which ensures that the notification facility is
--available at all master sites.

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE =>  'PACKAGE',
      ONAME => 'notify',
      SNAME => 'scott');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE =>  'PACKAGE BODY',
      ONAME => 'notify',
      SNAME => 'scott');
END;
/

--After you have completed building your notification facility, add the
--notification facility as one of your conflict resolution methods,
--even though it only notifies of a conflict. The following API example
--demonstrates adding the notification facility as a USER FUNCTION.

BEGIN
   DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
      SNAME => 'scott',
      ONAME => 'emp',
      CONSTRAINT_NAME => 'emp_ename_unique',
      SEQUENCE_NO => 1,
      METHOD => 'USER FUNCTION',
      COMMENT => 'Notify DBA',
      PARAMETER_COLUMN_NAME => 'ename',
      FUNCTION_NAME => 'scott.notify.emp_unique_violation');
END;
/
```

```
--After you have added the notification facility, you are ready to add the
--actual conflict resolution method to your table. The following API example
--demonstrates adding the APPEND SITE NAME uniqueness conflict resolution
--method to your replicated table.

BEGIN
   DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
      SNAME => 'scott',
      ONAME => 'emp',
      CONSTRAINT_NAME => 'emp_ename_unique',
      SEQUENCE_NO => 2,
      METHOD => 'APPEND SITE NAME',
      PARAMETER_COLUMN_NAME => 'ename');
END;
/

--After you have defined your conflict resolution methods, regenerate
--replication support for the table that received the conflict
--resolution methods.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Create Conflict Avoidance Methods for Delete Conflicts

Unlike update conflicts, where there are two values to compare, simply deleting a row makes the update conflict resolution methods described in the previous section ineffective because only one value would exist.

The best way to deal with deleting rows in a replication environment is to "avoid" the conflict by marking a row for deletion and periodically purging the table of all "marked" records. Because you are not physically removing this row, your data can converge at all master sites if a conflict arises because you still have two values to compare, assuming that no other errors have occurred. After you are sure that your data has converged, you can purge "marked" rows using a replicated purge procedure.

When you are developing your front-end application for your database, you probably want to "filter out" the rows that have been marked for deletion, because doing so makes it appear to your users as though the row was physically deleted. Simply exclude the rows that have been marked for deletion in the SELECT statement for your data set. For example, a select statement for a current employee listing might be similar to the following:

```
SELECT * FROM emp WHERE remove_date IS NULL;
```

This section describes how to prepare your replicated table to avoid delete conflicts. You also see how to use procedural replication to purge those records that have been "marked" for deletion.

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

```
--You must add a column to your replicated table that stores the
--mark for deleted records. It is advisable to use a timestamp to mark your
--records for deletion (timestamp reflects when the record was marked for
--deletion). Because you are using a timestamp, your new column must be
--a DATE datatype.  Use the DBMS_REPCAT.ALTER_MASTER_REPOBJECT procedure to add
--the REMOVE_DATE column to your existing replicated table.

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD (remove_date DATE)');
END;
/

--After you have inserted a new column into your replicated object,
--make sure that you regenerate replication support for
--the affected object. This step should be performed immmediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/
```

```
--The following package is replicated to all of the master sites in your
--replication environment. This package purges all "marked" records from
--the specified table.

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'PACKAGE',
      ONAME => 'purge',
      SNAME => 'scott',
      DDL_TEXT => 'CREATE OR REPLACE PACKAGE scott.purge AS
                      PROCEDURE remove_emp(purge_date DATE);
                   END;');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      GNAME => 'scott_mg',
      TYPE => 'PACKAGE BODY',
      ONAME => 'purge',
      SNAME => 'scott',
      DDL_TEXT => 'CREATE OR REPLACE PACKAGE BODY scott.purge AS
                      PROCEDURE remove_emp(purge_date IN DATE) IS
                      BEGIN
                         DBMS_REPUTIL.REPLICATION_OFF;
                         LOCK TABLE scott.emp IN EXCLUSIVE MODE;
                         DELETE scott.emp WHERE remove_date IS NOT NULL AND
                            remove_date < purge_date;
                         DBMS_REPUTIL.REPLICATION_ON;
                      EXCEPTION WHEN others THEN
                         DBMS_REPUTIL.REPLICATION_ON;
                      END;
                   END;');
END;
/
```

```
--After you have created your package (package and package body), generate
--replication support for each component. After you generate
--replication support, a synonym is created for you and added to your
--master group as a replicated object. This synonym is labeled as
--DEFER_PURGE.REMOVE_EMP.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'purge',
      TYPE => 'PACKAGE',
      MIN_COMMUNICATION => TRUE);
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'purge',
      TYPE => 'PACKAGE BODY',
      MIN_COMMUNICATION => TRUE);
END;
/

--After replication support has been regenerated, resume replication
--activity by using the RESUME_MASTER_ACTIVITY procedure API.

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Audit Successful Conflict Resolution

Whenever Oracle detects and successfully resolves an update, delete, or uniqueness conflict, you can view information about what method was used to resolve the conflict by querying the DBA_REPRESOLUTION_STATISTICS data dictionary view. This view is updated only if you have chosen to turn on conflict resolution statistics gathering for the table involved in the conflict.

> **See Also:** The ALL_REPRESOLUTION_STATISTICS on page 9-32 for more information.

Gathering Conflict Resolution Statistics

Use the REGISTER_STATISTICS procedure in the DBMS_REPCAT package to collect information about the successful resolution of update, delete, and uniqueness conflicts for a table. The following example gathers statistics for the EMP table in the ACCT_REC schema:

```
DBMS_REPCAT.REGISTER_STATISTICS(sname    =>   'acct_rec',
                                oname    =>   'emp');
```

> **See Also:** The REGISTER_STATISTICS procedure on page 8-140 for more information.

Viewing Conflict Resolution Statistics

After you call REGISTER_STATISTICS for a table, each conflict that is successfully resolved for that table is logged in the DBA_REPRESOLUTION_ STATISTICS view. Information about unresolved conflicts is always logged to the DEFERROR view, whether the object is registered or not.

> **See Also:** ALL_REPRESOLUTION_STATISTICS on page 9-32 and DEFERROR on page 9-49 for more information.

Canceling Conflict Resolution Statistics

Use the CANCEL_STATISTICS procedure in the DBMS_REPCAT package if you no longer want to collect information about the successful resolution of update, delete, and uniqueness conflicts for a table. The following example cancels statistics gathering on the EMP table in the ACCT_REC schema:

```
DBMS_REPCAT.CANCEL_STATISTICS(sname    => '  acct_rec',
                              oname    =>   'emp');
```

> **See Also:** The CANCEL_STATISTICS procedure on page 8-91 for more information.

Deleting Statistics Information

If you registered a table to log information about the successful resolution of update, delete, and uniqueness conflicts, you can remove this information from the DBA_REPRESOLUTION_STATISTICS view by calling the PURGE_STATISTICS procedure in the DBMS_REPCAT package.

The following example purges the statistics gathered about conflicts resolved due to inserts, updates, and deletes on the EMP table between January 1 and March 31:

```
DBMS_REPCAT.PURGE_STATISTICS(sname      =>   'acct_rec',
                             oname      =>   'emp',
                             start_date =>   '01-JAN-99',
                             end_date   =>   '31-MAR-99);
```

> **See Also:** The PURGE_STATISTICS procedure on page 8-136 for more information.

7

Manage Replicated Environment with APIs

This chapter illustrates how to manage your replication environment using the replication management API. The following topics are discussed:

- Managing Master Sites
- Managing Snapshot Sites
- Managing Deferred Transactions
- Managing the Error Queue
- Altering a Replicated Object
- Performing an Offline Instantiation Using Export/Import
- Determining Differences Between Replicated Tables
- Updating the Comments Fields in Data Dictionary Views

Managing Master Sites

As your data delivery needs change due to growth, shrinkage, or emergencies, you are undoubtedly going to need to change the configuration of your replication environment. This section is devoted to managing the master sites of your replication environment, which will help you alter and reconfigure your master sites.

Change Master Definition Site

Many replication administrative tasks can be performed only from the master definition site. Use the DBMS_REPCAT.RELOCATE_MASTERDEF procedure to move the master definition site to another master site. This API is especially useful when the master definition site becomes unavailable and you need to specify a new master definition site (see "Option 2" on page 7-3).

Option 1

If all master sites are available, complete the following:

Executed As: Replication Administrator

Executed At: Any Master Site

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.RELOCATE_MASTERDEF (
      GNAME => 'scott_mg',
      OLD_MASTERDEF => 'orc1.world',
      NEW_MASTERDEF => 'orc2.world',
      NOTIFY_MASTERS => TRUE,
      INCLUDE_OLD_MASTERDEF => TRUE);
END;
/
```

Option 2

If the old master definition site is NOT available, complete the following:

Executed As: Replication Administrator

Executed At: Any Master Site

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc3.world

BEGIN
   DBMS_REPCAT.RELOCATE_MASTERDEF (
      GNAME => 'scott_mg',
      OLD_MASTERDEF => 'orc1.world',
      NEW_MASTERDEF => 'orc2.world',
      NOTIFY_MASTERS => TRUE,
      INCLUDE_OLD_MASTERDEF => FALSE);
END;
/
```

See Also: The RELOCATE_MASTERDEF procedure on page 8-141 for more information on using this procedure.

Add a Master Site

As your replicated environment expands, you can use the ADD_MASTER_DATABASE procedure to add additional master sites to an existing master group. Executing this procedure replicates existing master objects to the new site.

Before you add a new master site, be sure that you properly set up your new master site for replication. Make sure that you follow the steps described in the "Set Up Master Sites" section on page 2-4.

Executed As: Replication Administrator

Executed At: Master Definition Site

Replication Status: Quiesced

```
CONNECT repadmin/repadmin@orc1.world

-- If the replication status is normal, change the status to quiesced.
BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

```
BEGIN
   DBMS_REPCAT.ADD_MASTER_DATABASE (
      GNAME => 'scott_mg',
      MASTER => 'orc4.world',
      USE_EXISTING_OBJECTS => TRUE,
      COPY_ROWS => TRUE,
      PROPAGATION_MODE => 'ASYNCHRONOUS');
END;
/

--NOTE: You should wait until the DBA_REPCATLOG view is empty. This view has
--temporary information that is cleared after successful execution. Execute
--the following SELECT statement in another SQL*Plus session to monitor
--the DBA_REPCATLOG view:
--
--SELECT * FROM dba_repcatlog WHERE gname = 'scott_mg';

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Drop a Master Site

When it becomes necessary to remove a master site from a master group, use the REMOVE_MASTER_DATABASES procedure to drop one or more master sites.

Executed As: Replication Administrator

Executed At: Master Definition Site

Replication Status: Quiesced

```
CONNECT repadmin/repadmin@orc1.world

-- If the replication status is normal, change the status to quiesced.
BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

```
BEGIN
   DBMS_REPCAT.REMOVE_MASTER_DATABASES (
      GNAME => 'scott_mg',
      MASTER_LIST => 'orc4.world');
END;
/

--NOTE: You should wait until the DBA_REPCATLOG view is empty. Execute
--the following SELECT statement in another SQL*Plus session to monitor
--the DBA_REPCATLOG view:
--
--SELECT * FROM dba_repcatlog WHERE gname = 'scott_mg';

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Managing Snapshot Sites

Snapshot replication provides you with the flexibility to build data sets to meet your users' needs, your security configuration needs, and your front-end applications' needs. The following two sections describe how to manage snapshot sites with the replication management API.

Using a Group Owner

Specifying a group owner when you define a new snapshot group and its related objects allows you to create multiple snapshot groups based on the same master group at a single snapshot site. At a snapshot site, specifying group owners enables you to create multiple snapshot groups that are based on the same master group. You accomplish this by creating the snapshot groups under different schemas at the snapshot site.

> **See Also:** The "Organizational Mechanisms" sections in Chapter 3 of *Oracle8i Replication* for a complete discussion on using group owners and the advantages of using multiple data sets.

```
--The following procedures must be executed by the snapshot administrator
--at the remote snapshot site.

CONNECT snapadmin/snapadmin@snap1.world

--The master group that you specify in the GNAME parameter must match the
--name of the master group that you are replicating at the target master site.
--The GOWNER parameter allows you to specify an additional identifier that lets
--you create multiple snapshot groups based on the same master group at the same
--snapshot site.

--In this example, snapshot groups are created for the group owners BOB and
--JANE, and these two snapshot groups are based on the same master group.

--Create snapshot group with group owner (GOWNER) BOB.
BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP (
      GNAME => 'scott_mg',
      MASTER => 'orc1.world',
      PROPAGATION_MODE => 'ASYNCHRONOUS',
      GOWNER => 'bob');
END;
/

--Create snapshot group with group owner (GOWNER) JANE.
BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP (
      GNAME => 'scott_mg',
      MASTER => 'orc1.world',
      PROPAGATION_MODE => 'ASYNCHRONOUS',
      GOWNER => 'jane');
END;
/

--The GOWNER value used when creating your snapshot objects must match the
--GOWNER value specified when you created the snapshot group (previous
--procedures). After you have created the snapshot groups, you can create
--snapshots of the same master table in the SCOTT_MG snapshot group owned by BOB
--and JANE.
```

```
--WARNING: You need to make sure that each object created has a unique name.
--When using a GOWNER to create multiple snapshot groups, duplicate object names
--could become a problem. To avoid any object-naming conflicts, you may want to
--append the GOWNER value to the end of the object name that you create, as
--illustrated in the following procedures (that is, CREATE SNAPSHOT
--scott.bonus_bob). Such a naming method ensures that you do not create any
--objects with conflicting names.

--Create snapshot based on the scott.bonus table in the SCOTT_MG snapshot
--group owned by BOB.
BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'bonus_bob',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.bonus_bob REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.bonus@orc1.world',
      MIN_COMMUNICATION => TRUE,
      GOWNER => 'bob');
END;
/

--Create snapshot based on the same scott.bonus table in the SCOTT_MG snapshot
--group owned by JANE.
BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'bonus_jane',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.bonus_jane REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.bonus@orc1.world',
      MIN_COMMUNICATION => TRUE,
      GOWNER => 'jane');
END;
/
```

```
--Create snapshot based on the scott.dept table in the SCOTT_MG snapshot
--group owned by BOB.

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'dept_bob',
      TYPE => 'SNAPSHOT',
      ddl_text => 'CREATE SNAPSHOT scott.dept_bob REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.dept@orc1.world',
      MIN_COMMUNICATION => TRUE,
      GOWNER => 'bob');
END;
/

--Create snapshot based on the scott.emp table in the SCOTT_MG snapshot
--group owned by JANE.
BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'emp_jane',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.emp_jane REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.emp@orc1.world',
      MIN_COMMUNICATION => TRUE,
      GOWNER => 'jane');
END;
/
```

```
--Create snapshot based on the scott.salgrade table in the SCOTT_MG snapshot
--group owned by BOB.
BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      ONAME => 'salgrade_bob',
      TYPE => 'SNAPSHOT',
      DDL_TEXT => 'CREATE SNAPSHOT scott.salgrade_bob REFRESH FAST WITH
                   PRIMARY KEY FOR UPDATE AS SELECT * FROM
                   scott.salgrade@orc1.world',
      MIN_COMMUNICATION => TRUE,
      GOWNER => 'bob');
END;
/

--After you have finished building your snapshot groups, you should add your
--snapshots to a refresh group. See Chapter 5, "Create Snapshot Group"
--(Step 6) for more information about adding snapshots to a refresh group.
```

Changing a Snapshot Group's Master Site

To change the master site of a snapshot group to another master site, call the SWITCH_SNAPSHOT_MASTER procedure in the DBMS_REPCAT package, as shown in the following example:

```
BEGIN
   DBMS_REPCAT.SWITCH_SNAPSHOT_MASTER(
      gname          =>   'sales',
      master         =>   'dbs2.acme.com'
      execute_as_user =>  'FALSE');
END;
/
```

In this example, the master site for the SALES object group is changed to the DBS2 master site. You must call this procedure at the snapshot site whose master site you want to change. The new database must be a master site in the replicated environment. When you call this procedure, Oracle uses the new master to perform a full refresh of each snapshot in the local snapshot group.

The entries in the SYS.SLOG$ table at the old master site for the switched snapshot are not removed. As a result, the MLOG$ table of the switched updatable snapshot at the old master site has the potential to grow indefinitely, unless you purge it by calling DBMS_SNAPSHOT.PURGE_LOG.

> **See Also:** The SWITCH_SNAPSHOT_MASTER procedure on page 8-149 for more information.

Dropping Snapshot Sites

You may need to drop replication activity at a snapshot site for a number of reasons. Perhaps the data requirements have changed or an employee has left the company. In any case, as a DBA you will need to drop the replication support for the target snapshot site.

Drop Snapshot Group Created with Deployment Templates

The process for dropping a snapshot group that was created by instantiating a deployment template at a snapshot site is slightly different than the methods described in the following sections. Before you drop the snapshot group at the remote snapshot site, you need to execute the DROP_SITE_INSTANTIATION procedure at the target master site for snapshot group. In addition to removing the metadata relating to the snapshot group, this procedure also removes the related deployment template data regarding this site.

The DROP_SITE_INSTANTIATION procedure has a public and a private version. The public version allows the owner of the snapshot group to drop the snapshot site, while the private version allows the replication administrator to drop a snapshot site on behalf of the snapshot group owner.

Public

The following steps are to be performed by owner of the snapshot group.

Executed As: Snapshot Group Owner

Executed At: Master Site for Target Snapshot Site

Replication Status: Normal

```
CONNECT scott/tiger@orc1.world

--If you need to drop a snapshot site that was instantiated on an Oracle8i Lite
--database, see the Oracle8i Lite documentation for information.

BEGIN
   DBMS_REPCAT_INSTANTIATE.DROP_SITE_INSTANTIATION(
      REFRESH_TEMPLATE_NAME =>  'personnel',
      SITE_NAME => 'snap1.world');
END;
/

--After you have executed the DROP_SITE_INSTANTIATION procedure, you should
--connect to the remote snapshot site and drop the snapshot group. If you are
--not able to connect to the remote snapshot site due to loss or theft, the
--target snapshot group cannot refresh, but the existing data
--still remains at the snapshot site.
CONNECT snapadmin/snapadmin@snap1.world

--If you want to physically remove the contents of the snapshot group, be sure
--that you specify TRUE for the DROP_CONTENTS parameter.

BEGIN
   DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP (
      GNAME => 'scott_mg',
      DROP_CONTENTS => TRUE);
END;
/

--After you remove the snapshot group, you should remove the refresh group.

BEGIN
   DBMS_REFRESH.DESTROY (
       NAME => 'personnel01');
END;
/
```

Private

The following steps are to be performed by the replication administrator on behalf of the snapshot group owner.

Executed As: Replication Administrator

Executed At: Master Site for Target Snapshot Site

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc1.world

--If you need to drop a snapshot site that was instantiated on an Oracle8i Lite
--database, see the Oracle8i Lite documentation for information.

BEGIN
   DBMS_REPCAT_RGT.DROP_SITE_INSTANTIATION (
      REFRESH_TEMPLATE_NAME =>  'personnel',
      USER_NAME => 'scott',
      SITE_NAME => 'snap1.world');
END;
/

--After you have executed the DROP_SITE_INSTANTIATION procedure, you should
--connect to the remote snapshot site and drop the snapshot group. If you are
--not able to connect to the remote snapshot site due to loss or theft, the
--target snapshot group cannot refresh, but the existing data
--still remains at the snapshot site.

CONNECT snapadmin/snapadmin@snap1.world

--If you want to physically remove the contents of the snapshot group, be sure
--that you specify TRUE for the DROP_CONTENTS parameter.

BEGIN
   DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP (
      GNAME => 'scott_mg',
      DROP_CONTENTS => TRUE);
END;
/
```

Drop Snapshot Objects at Snapshot Site

The most secure method of removing replication support for a snapshot site is to physically drop the replicated objects or groups at the snapshot site. The following two sections describe how to drop these objects and groups while connected to the snapshot group.

Ideally, these procedures should be executed while the snapshot is connected to its target master site. A connection ensures that any related metadata at the master site is removed. If a connection to the master site is not possible, be sure to complete the procedure described in the "Clean Up Master Site" on page 7-14 to manually remove the related metadata.

Drop Snapshot Group at Snapshot Site When it becomes necessary to remove a snapshot group from a snapshot site, use the DROP_SNAPSHOT_REPGROUP procedure to drop a snapshot group. When you execute this procedure and are connected to the target master site, the metadata for the target snapshot group at the master site is removed. If you cannot connect, see "Clean Up Master Site" on page 7-14 for more information.

Executed As: Snapshot Administrator

Executed At: Remote Snapshot Site

Replication Status: N / A

```
CONNECT snapadmin/snapadmin@snap1.world

--If you want to physically remove the contents of the snapshot group, be sure
--that you specify TRUE for the DROP_CONTENTS parameter.

BEGIN
   DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP (
      GNAME => 'scott_mg',
      DROP_CONTENTS => TRUE);
END;
/
```

Drop Individual Snapshot at Snapshot Site When it becomes necessary to remove an individual snapshot from a snapshot site, use the DROP_SNAPSHOT_REPOBJECT procedure API to drop a snapshot. When you execute this procedure, the metadata for the target snapshot at the master site is removed. When you execute this procedure and are connected to the target master site, the metadata for the target snapshot group at the master site is removed. If you cannot connect, see "Clean Up Master Site" on page 7-14 for more information.

Executed As: Snapshot Administrator

Executed At: Remote Snapshot Site

Replication Status: N / A

```
CONNECT snapadmin/snapadmin@snap1.world

--If you want to physically remove the contents of the snapshot, be sure
--that you specify TRUE for the DROP_CONTENTS parameter.

BEGIN
   DBMS_REPCAT.DROP_SNAPSHOT_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'bonus',
      TYPE => 'SNAPSHOT',
      DROP_OBJECTS => TRUE);
END;
/
```

Clean Up Master Site

If you are unable to drop a snapshot group or snapshot object while connected to the target master site, you must remove the related metadata at the master site manually. Cleaning up the metadata also ensures that you are not needlessly maintaining master table changes to a snapshot log. The following sections help you clean up your master site after dropping a snapshot group or object.

Clean Up After Dropping Snapshot Group If you have executed the steps described in the "Drop Snapshot Group at Snapshot Site" section on page 7-13 and were not connected to the master site, you are encouraged to complete the following steps to clean up the target master site.

Executed As: Replication Administrator

Executed At: Master Site for Target Snapshot Site

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_REPCAT.UNREGISTER_SNAPSHOT_REPGROUP (
      GNAME => 'scott_mg',
      SNAPSITE => 'snap1.world');
END;
/

--After you unregister the snapshot group, you should purge the snapshot logs
--of the entries that were marked for the target snapshots. The
--PURGE_SNAPSHOT_FROM_LOG procedure must be executed for each snapshot
--that was in the snapshot replication group.

--NOTE: If for some reason unregistering the snapshot group fails, you are still
--encouraged to complete the following steps.

BEGIN
   DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
      SNAPOWNER => 'scott',
      SNAPNAME => 'emp',
      SNAPSITE => 'snap1.world');
END;
/

BEGIN
   DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
      SNAPOWNER => 'scott',
      SNAPNAME => 'dept',
      SNAPSITE => 'snap1.world');
END;
/

BEGIN
   DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
      SNAPOWNER => 'scott',
      SNAPNAME => 'bonus',
      SNAPSITE => 'snap1.world');
END;
/
```

```
BEGIN
   DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
      SNAPOWNER => 'scott',
      SNAPNAME => 'salgrade',
      SNAPSITE => 'snap1.world');
END;
/
```

Clean Up Individual Snapshot Support at Master Site If you have executed the steps described in the "Drop Individual Snapshot at Snapshot Site" section on page 7-14 and were not connected to the master site, you are encouraged to complete the following steps to clean up the target master site.

Executed As: Replication Administrator

Executed At: Master Site for Target Snapshot Site

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_SNAPSHOT.UNREGISTER_SNAPSHOT (
      SNAPOWNER => 'scott',
      SNAPNAME => 'bonus',
      SNAPSITE => 'snap1.world');
END;
/

--After you unregister the snapshot, you should purge the associated snapshot
--log of the entries that were marked for the target snapshots.

--NOTE: If for some reason unregistering the snapshot fails, you are still
--encouraged to complete the following step.

BEGIN
   DBMS_SNAPSHOT.PURGE_SNAPSHOT_FROM_LOG (
      SNAPOWNER => 'scott',
      SNAPNAME => 'bonus',
      SNAPSITE => 'snap1.world');
END;
/
```

Managing Snapshot Logs

The following sections explain how to manage snapshot logs. Topics include:

- Altering Snapshot Logs
- Managing Snapshot Log Space
- Reorganizing Master Tables that Have Snapshot Logs
- Deleting a Snapshot Log

Altering Snapshot Logs

After you create a snapshot log, you can alter its storage parameters and support for corresponding snapshots. The following sections explain more about altering snapshot logs. Only the following users can alter a snapshot log:

- the owner of the master table
- a user with the SELECT privilege for the master table and ALTER privilege on the MLOG$_*master_table_name*, where *master_table_name* is the name of the master table for the snapshot log. For example, if the master table is EMP, the snapshot log table name is MLOG$_EMP.

Altering Snapshot Log Storage Parameters To alter a snapshot log's storage parameters, use the ALTER SNAPSHOT LOG statement. For example, the following statement alters a snapshot log on the CUSTOMERS table in the SALES schema:

```
ALTER SNAPSHOT LOG ON sales.customers
 PCTFREE 25
 PCTUSED 40;
```

Altering a Snapshot Log to Add Filter Columns To add new filter columns to a snapshot log, use the SQL statement ALTER SNAPSHOT LOG. For example, the following statement alters a snapshot log on the CUSTOMERS table in the SALES schema:

```
ALTER SNAPSHOT LOG ON sales.customers
 ADD (zip);
```

> **See Also:** *Oracle8i Replication* for more information about filter columns.

Managing Snapshot Log Space

Oracle automatically tracks which rows in a snapshot log have been used during the refreshes of snapshots, and purges these rows from the log so that the log does not grow endlessly. Because multiple simple snapshots can use the same snapshot log, rows already used to refresh one snapshot may still be needed to refresh another snapshot. Oracle does not delete rows from the log until *all* snapshots have used them.

For example, Oracle refreshes the CUSTOMERS snapshot at the SPDB1 database. However, the server that manages the master table and associated snapshot log does not purge the snapshot log rows used during the refresh of this snapshot until the CUSTOMERS snapshot at the SPDB2 database also refreshes using these rows.

As a result of how Oracle purges rows from a snapshot log, unwanted situations can occur that cause a snapshot log to grow indefinitely when multiple snapshots are based on the same master table. For example, such situations can occur when more than one snapshot is based on a master table and one of the following conditions is true:

- One snapshot is not configured for automatic refreshes and has not been manually refreshed for a long time.
- One snapshot has an infrequent refresh interval, such as every year (365 days).
- A network failure has prevented an automatic refresh of one or more of the snapshots based on the master table.
- A network or site failure has prevented a master from becoming aware that a snapshot has been dropped.

Purging Rows from a Snapshot Log Always try to keep a snapshot log as small as possible to minimize the database space that it uses. To remove rows from a snapshot log and make space for newer log records, you can perform one of the following actions:

- Refresh the snapshots associated with the log so that Oracle can purge rows from the snapshot log.
- Manually purge records in the log by deleting rows required only by the *n*th least recently refreshed snapshots.

To manually purge rows from a snapshot log, execute the PURGE_LOG stored procedure of the DBMS_SNAPSHOT package at the database that contains the log. For example, to purge entries from the snapshot log of the CUSTOMERS table that are necessary only for the least recently refreshed snapshot, execute the following procedure:

```
BEGIN
   DBMS_SNAPSHOT.PURGE_LOG (
      master => 'sales.customers',
      num    => 1,
      flag   => 'DELETE');
END;
/
```

> **See Also:** The PURGE_LOG procedure on page 8-236 for more information.

Only the owner of a snapshot log or a user with the EXECUTE privilege for the DBMS_SNAPSHOT package can purge rows from the snapshot log by executing the PURGE_LOG procedure.

Truncating a Snapshot Log If a snapshot log grows and allocates many extents, purging the log of rows does not reduce the amount of space allocated for the log. To reduce the space allocated for a snapshot log:

1. Acquire an exclusive lock on the master table to prevent updates from occurring during the following process. For example, issue a statement similar to the following:

   ```
   LOCK TABLE sales.customers IN EXCLUSIVE MODE;
   ```

2. Using a second database session, copy the rows in the snapshot log (in other words, the MLOG$ base table) to a temporary table. For example, issue a statement similar to the following:

   ```
   CREATE TABLE sales.templog AS SELECT * FROM sales.mlog$_customers;
   ```

3. Using the second session, truncate the log using the SQL statement TRUNCATE. For example, issue a statement similar to the following:

   ```
   TRUNCATE sales.mlog$_customers;
   ```

4. Using the second session, reinsert the old rows so that you do not have to perform a complete refresh of the dependent snapshots. For example, issue a statement similar to the following:

```
INSERT INTO sales.mlog$_customers SELECT * FROM sales.templog;
DROP TABLE sales.templog;
```

5. Using the first session, release the exclusive lock on the master table by performing a rollback:

```
ROLLBACK;
```

> **Note:** Any changes made to the master table between the time you copy the rows to a new location and when you truncate the log do not appear until after you perform a *complete* refresh.

Only the owner of a snapshot log or a user with the DELETE ANY TABLE system privilege can truncate a snapshot log.

Reorganizing Master Tables that Have Snapshot Logs

To improve performance and optimize disk use, you can periodically reorganize tables. This section discusses how to reorganize a master table and preserve the fast refresh capability of associated snapshots.

Reorganization Notification When you reorganize a table, any ROWID information of the snapshot log must be invalidated. Oracle detects a table reorganization automatically only if the table is *truncated* as part of the reorganization. See "Method 2 for Reorganizing Table T" on page 7-22.

If the table is not truncated, Oracle must be notified of the table reorganization. To support table reorganizations, two procedures, DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION and DBMS_SNAPSHOT.END_TABLE_REORGANIZATION, notify Oracle that the specified table has been reorganized. The procedures perform clean-up operations, verify the integrity of the logs and triggers that the fast refresh mechanism needs, and invalidate the ROWID information in the table's snapshot log. The inputs are the owner and name of the master table to be reorganized. There is no output.

Truncating Master Tables When a table is truncated, its snapshot log is also truncated. However, for primary key snapshots, you can preserve the snapshot log, allowing fast refreshes to continue. Although the information stored in a snapshot log is preserved, the snapshot log becomes invalid with respect to ROWIDs when the master table is truncated. The ROWID information in the snapshot log will seem to be newly created and cannot be used by ROWID snapshots for fast refresh.

If you specify the PRESERVE SNAPSHOT LOG option or no option, the information in the master table's snapshot log is preserved, but current ROWID snapshots can use the log for a fast refresh only *after* a complete refresh has been performed. This is the default behavior.

Note: To ensure that any previously fast refreshable snapshot is still refreshable, follow the guidelines in "Methods of Reorganizing a Database Table" on page 7-21.

If the PURGE SNAPSHOT LOG option is specified, the snapshot log is purged along with the master table.

Examples Either of the following two statements preserves snapshot log information when the master table named ORDERS is truncated:

```
TRUNCATE TABLE orders PRESERVE SNAPSHOT LOG;
TRUNCATE TABLE orders;
```

The following statement truncates the snapshot log along with the master table:

```
TRUNCATE TABLE orders PURGE SNAPSHOT LOG;
```

Methods of Reorganizing a Database Table Oracle provides four table reorganization methods that preserve the capability for fast refresh. These appear in the following sections. Other reorganization methods require an initial complete refresh to enable subsequent fast refreshes.

Note: Do *not* use Direct Loader during a reorganization of a master table. Direct Loader can cause reordering of the columns, which could invalidate the log information used in subquery and LOB snapshots.

Method 1 for Reorganizing Table T

1. Call DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION for table T.
2. Rename table T to T_OLD.
3. Create table T as SELECT * FROM T_OLD.
4. Call DBMS_SNAPSHOT.END_TABLE_REORGANIZATION for new table T.

> **Caution:** When a table is renamed, its associated PL/SQL triggers are also adjusted to the new name of the table.

Ensure that no transaction is issued against the reorganized table between calling DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION and DBMS_SNAPSHOT.END_TABLE_REORGANIZATION.

Method 2 for Reorganizing Table T

1. Call DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION for table T.
2. Export table T.
3. Truncate table T with PRESERVE SNAPSHOT LOG option.
4. Import table T using conventional path.
5. Call DBMS_SNAPSHOT.END_TABLE_REORGANIZATION for new table T.

> **Caution:** When you truncate master tables as part of a reorganization, you must use the PRESERVE SNAPSHOT LOG clause of the truncate table DDL.

Ensure that no transaction is issued against the reorganized table between calling DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION and DBMS_SNAPSHOT.END_TABLE_REORGANIZATION.

Method 3 for Reorganizing Table T

1. Call DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION for table T.
2. Export table T.
3. Rename table T to T_OLD.
4. Import table T using conventional path.
5. Call DBMS_SNAPSHOT.END_TABLE_REORGANIZATION for new table T.

> **Caution:** When a table is renamed, its associated PL/SQL triggers are also adjusted to the new name of the table.

Ensure that no transaction is issued against the reorganized table between calling DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION and DBMS_SNAPSHOT.END_TABLE_REORGANIZATION.

Method 4 for Reorganizing Table T

1. Call DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION for table T.
2. Select contents of table T to a flat file.
3. Rename table T to T_OLD.
4. Create table T with the same shape as T_OLD.
5. Run SQL*Loader using conventional path.
6. Call DBMS_SNAPSHOT.END_TABLE_REORGANIZATION for new table T.

> **Caution:** When a table is renamed, its associated PL/SQL triggers are also adjusted to the new name of the table.

Ensure that no transaction is issued against the reorganized table between calling DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION and DBMS_SNAPSHOT.END_TABLE_REORGANIZATION.

Deleting a Snapshot Log

You can delete a snapshot log regardless of its master table or any existing snapshots. For example, you might decide to drop a snapshot log if one of the following conditions is true:

- All snapshots of a master table have been dropped.
- All snapshots of a master table are to be completely refreshed, not fast refreshed.
- A master table no longer supports snapshots that require fast refreshes.

To delete a snapshot log, execute the DROP SNAPSHOT LOG SQL statement in SQL*Plus. For example, the following statement deletes the snapshot log for a table named CUSTOMERS in the SALES schema:

```
DROP SNAPSHOT LOG ON sales.customers;
```

Only the owner of the master table or a user with the DROP ANY TABLE system privilege can drop a snapshot log.

Managing Deferred Transactions

Typically, Oracle replication is configured to push and purge the deferred transaction queue automatically. At times, however, you may need to push or purge the deferred transaction queue manually. The following sections contain examples for pushing and purging the deferred transaction queue at a snapshot site, but the process is the same at master sites.

Pushing the Deferred Transaction Queue

If you do not automatically propagate the transactions in your deferred transaction queue during the refresh of your snapshot, you must complete the following steps to propagate changes made to the updateable snapshot to its master table.

```
--The following procedures must be executed by the snapshot administrator
--at the remote snapshot site.

CONNECT snapadmin/snapadmin@snap1.world

--Propagation of the deferred transaction queue is based on the destination of
--the transaction. Execute the following SELECT statement to view the deferred
--transactions and their destinations. Each distinct destination and the number
--of transactions pending for the destination wil be displayed:
```

```
SELECT DISTINCT(dblink), COUNT(deferred_tran_id)
   FROM deftrandest GROUP BY dblink;

--You need to execute the DBMS_DEFER_SYS.PUSH function for each master
--site that is listed as a destination for a deferred transaction.

DECLARE
   temp INTEGER;
BEGIN
   temp := DBMS_DEFER_SYS.PUSH (
      DESTINATION => 'orc1.world',
      STOP_ON_ERROR => FALSE,
      DELAY_SECONDS => 0,
      PARALLELISM => 0);
END;
/

--Repeat the above procedure for each destination that was returned in the above
--SELECT statement.
```

Purging the Deferred Transaction Queue

If you do not automatically purge the successfully propagated transactions in your deferred transaction queue periodically, you must complete the following steps to purge them manually.

```
--The following procedures must be executed by the snapshot administrator
--at the remote snapshot site.

CONNECT snapadmin/snapadmin@snap1.world

DECLARE
   temp INTEGER;
BEGIN
   temp := DBMS_DEFER_SYS.PURGE (
      PURGE_METHOD => purge_method_quick);
END;
/
```

Managing the Error Queue

As an administrator of a replication environment, you should regularly monitor the error queue to determine if any deferred transactions were not successfully applied at the target master site.

To check the error queue, issue the following SELECT statement as the replication administrator when connected to the target master site:

```
SELECT * FROM deferror;
```

If the error queue contains errors, you should resolve the error condition and re-execute the deferred transaction. You have two options when re-executing a deferred transaction: you can re-execute in the security context of the user who received the deferred transaction or you can re-execute the deferred transaction with an alternate security context.

Re-execute Error Transaction as the Receiver

The procedure below re-executes a specified deferred transaction in the security context of the user who received the deferred transaction. This procedure should not be executed until the error situation has been resolved.

Executed As: Replication Administrator

Executed At: Site Containing Errors

Replication Status: Normal

```
CONNECT repadmin/repadmin@orc2.world

BEGIN
   DBMS_DEFER_SYS.EXECUTE_ERROR (
      DEFERRED_TRAN_ID => '128323',
      DESTINATION => 'orc2.world');
END;
/
```

Re-execute Error Transaction as Alternate User

The procedure below re-executes a specified deferred transaction in the security context of the currently connected user. This procedure should not be executed until the error situation has been resolved.

Executed As: Connected User

Executed At: Site Containing Errors

Replication Status: Normal

```
CONNECT scott/tiger@orc2.world

BEGIN
   DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER (
      DEFERRED_TRAN_ID => '128323',
      DESTINATION => 'orc2.world');
END;
/
```

Altering a Replicated Object

As your database needs change, you may need to modify the characteristics of your replicated objects. It is important that you do not directly execute DDL to alter your replicated objects. Doing so may cause your replicated environment to fail.

Use the DBMS_REPCAT.ALTER_MASTER_REPOBJECT procedure to alter the characteristics of your replicated objects. From the example below, notice that you simply include the necessary DDL within the procedure call (see the DDL_TEXT parameter).

Executed As: Replication Administrator

Executed At: Master Definition Site

Replication Status: Quiesced

```
CONNECT repadmin/repadmin@orc1.world

-- If the replication status is normal, change the status to quiesced.
BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      DDL_TEXT => 'ALTER TABLE scott.emp ADD (site VARCHAR2(20))');
END;
/

--After you have inserted a new column into your replicated object,
--you need to make sure that you regenerate replication support for
--the affected object. This step should be performed immmediately
--after you alter the replicated object.

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      SNAME => 'scott',
      ONAME => 'emp',
      TYPE => 'TABLE',
      MIN_COMMUNICATION => TRUE);
END;
/

--NOTE: You should wait until the DBA_REPCATLOG view is empty. Execute
--the following SELECT statement in another SQL*Plus session to monitor
--the DBA_REPCATLOG view:
--
--SELECT * FROM dba_repcatlog WHERE gname = 'scott_mg';

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/
```

Performing an Offline Instantiation Using Export/Import

Expanding established replicated environments can cause network traffic when you add a new master or snapshot site to your replicated environment. This is caused by propagating the entire contents of the table or snapshot through the network to the new replicated site.

To minimize such network traffic, you can expand your replicated environment by using the offline instantiation procedure. Offline instantiation takes advantage of Oracle's Export and Import utilities, which allow you to create an export file and transfer the data to the new site through another storage medium, such as CD-ROM, tape, and so on.

Master Site

The following script is an example of how to perform an offline instantiation of a master site. This script can potentially save large amounts of network traffic caused by the normal method of adding a new master site to an existing master group.

Executed As: Replication Administrator

Executed At: Master Definition Site and New Master Site

Replication Status: Quiesced and Partial

```
/*****************************************************************
SET UP NEW MASTER SITE

You need to complete the steps illustrated in the "Set Up Master Sites" section
on page 2-4. Make sure the appropriate schema and database links have been
created before you perform the offline instantiation of your new master
site. Be sure to create the database links from the new master site to
each of the existing masters sites. Also, create a database link from each of
the existing master sites to the new master site.

After the database links have been created, make sure that you also
define the SCHEDULED LINKS for each of the new database links (STEP 9:
CREATE SCHEDULED LINKS).
*****************************************************************/
```

```
/*************************************************************************
SUSPEND MASTER ACTIVITY

You need to suspend master activity for the existing master sites
before you export your master data and begin the offline instantiation
process.
*************************************************************************/

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      GNAME => 'scott_mg');
END;
/

/*************************************************************************
VERIFY THAT THERE ARE NO PENDING TRANSACTIONS

This includes pushing any outstanding deferred transactions,
resolving any error transactions, and/or pushing any administrative
transactions. This step must be performed at each of the existing
master sites.
*************************************************************************/

--Connect to master definition site.

CONNECT repadmin/repadmin@orc1.world

--Check for error transaction queue.

SELECT * FROM deferror;

--If any deferred transactions have been entered into the error queue, then
--you need to resolve the error situation and then manually re-execute the
--deferred transaction.

BEGIN
   DBMS_DEFER_SYS.EXECUTE_ERROR (
      DEFERRED_TRAN_ID => '128323',
      DESTINATION => 'orc1.world');
END;
/
```

```
--Check for outstanding administrative requests.

SELECT * FROM dba_repcatlog;

--If any administrative requests remain, then you can manually push these
--transactions or wait for them to be executed automatically. You may need
--to execute the DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN API several times,
--because some administrative operations have multiple steps.

BEGIN
   DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN (
      GNAME => 'scott_mg',
      ALL_SITES => TRUE);
END;
/

/*************************************************************************
BEGIN OFFLINE INSTANTIATION PROCEDURE
*************************************************************************/
--Connect as replication administrator to Master Definition Site

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_OFFLINE_OG.BEGIN_INSTANTIATION (
      GNAME => 'scott_mg',
      NEW_SITE => 'orc4.world');
END;
/

--NOTE: You should wait until the DBA_REPCATLOG view is empty. This view has
--temporary information that is cleared after successful execution. Execute
--the following SELECT statement in another SQL*Plus session to monitor
--the DBA_REPCATLOG view:
--
--SELECT * FROM dba_repcatlog WHERE gname = 'scott_mg';
```

```
/*****************************************************************
CONNECT AS SCOTT/TIGER TO EXPORT

Use the Oracle export utility to generate the export file that you will transfer
to the new master site. The export file contains the replicated objects to be
added at the new master site. See Oracle8i Utilities for additional information.
*****************************************************************/

exp scott/tiger@orc1.world

/*****************************************************************
RESUME PARTIAL REPLICATION ACTIVITY

Because it may take you some time to complete the offline instantiation
process, you can resume replication activity for the remaining
master sites (excluding the new master site) by executing the
DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS procedure after the export is
complete. In the DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS procedure below,
replication activity is resumed at all master sites except the
new master site -- orc4.world.
*****************************************************************/

--Connect as replication administrator to master definition site.

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS (
      GNAME => 'scott_mg',
      NEW_SITE => 'orc4.world');
END;
/

/*****************************************************************
PREPARE NEW MASTER SITE

After you have transferred the export file from the master definition
site to the new master site, you must prepare the new site to import
the data in your export file. The following procedure is to be executed
at the new master site.
*****************************************************************/
```

```
CONNECT repadmin/repadmin@orc4.world

BEGIN
   DBMS_OFFLINE_OG.BEGIN_LOAD (
      GNAME => 'scott_mg',
      NEW_SITE => 'orc4.world');
END;
/

/*********************************************************************
IMPORT DATA FROM EXPORT FILE

Once you have imported the export file that you generated earlier,
you have transferred the data from your master definition site
to your new master site.
*********************************************************************/

imp scott/tiger@orc4.world FULL=y IGNORE=y

/*********************************************************************
COMPLETE LOAD PROCESS AT NEW MASTER SITE

After you have imported the export file, you are ready to complete
the offline instantiation process at the new master site. Executing the
DBMS_OFFLINE_OG.END_LOAD procedure prepares the new site for
normal replication activity.
*********************************************************************/

CONNECT repadmin/repadmin@orc4.world

BEGIN
   DBMS_OFFLINE_OG.END_LOAD (
      GNAME => 'scott_mg',
      NEW_SITE => 'orc4.world');
END;
/
```

```
/*************************************************************************
COMPLETE INSTANTIATION PROCESS

Once you have completed the steps at the new master site, you are ready
to complete the offline instantiation process. Executing the
DBMS_OFFLINE_OG.END_INSTANTIATION procedure completes the process
and resumes normal replication activity at all master sites. The
following procedure is to be executed at the master definition site.
*************************************************************************/
CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_OFFLINE_OG.END_INSTANTIATION (
      GNAME => 'scott_mg',
      NEW_SITE => 'orc4.world');
END;
/
```

Snapshot Site

For the same reasons that you might want to perform an offline instantiation of a master site, you may also want to create a new snapshot group at a snapshot site using the offline instantiation process. In some cases, it is even more useful for snapshots considering that the target computer could very well be a laptop using a modem connection.

The following script describes the process of performing an offline instantiation for a new snapshot group.

Executed As: Replication Administrator and Snapshot Administrator

Executed At: Master Site and New Snapshot Site

Replication Status: Normal

```
/*************************************************************************
SET UP SNAPSHOT SITE

You need to complete the steps illustrated in the "Set Up Snapshot Sites"
section on page 2-15. Make sure that the appropriate schema and database
links have been created before you perform the offline instantiation of your
snapshot.
*************************************************************************/
```

```
/*****************************************************************************
CREATE SNAPSHOT LOGS

If snapshot logs do not already exist for the target master tables, create them
at the target master site.
*****************************************************************************/

CONNECT repadmin/repadmin@orc1.world

CREATE SNAPSHOT LOG ON scott.emp;
CREATE SNAPSHOT LOG ON scott.dept;
CREATE SNAPSHOT LOG ON scott.bonus;
CREATE SNAPSHOT LOG ON scott.salgrade;

/*****************************************************************************
CREATE TEMPORARY SNAPSHOTS

Create temporary snapshots at the master site that will contain the
data that you transfer to your new snapshot site using the export file.

NOTE: If you added any of the conflict resolution routines described in
Chapter 6, "Conflict Resolution", you may have additional columns in your
tables. Be certain to include these additional columns in the SELECT
statements below. Updatable snapshots require that you explicity select all
columns in the master table (no SELECT *).
*****************************************************************************/

CREATE SNAPSHOT scott.snap_emp REFRESH FAST WITH PRIMARY KEY FOR UPDATE
   AS SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
   FROM scott.emp@orc1.world;

CREATE SNAPSHOT scott.snap_dept REFRESH FAST WITH PRIMARY KEY FOR UPDATE
   AS SELECT deptno, dname, loc
   FROM scott.dept@orc1.world;

CREATE SNAPSHOT scott.snap_bonus REFRESH FAST WITH PRIMARY KEY FOR UPDATE
   AS SELECT ename, job, sal, comm
   FROM scott.bonus@orc1.world;

CREATE SNAPSHOT scott.snap_salgrade REFRESH FAST WITH PRIMARY KEY FOR UPDATE
   AS SELECT grade, losal, hisal
   FROM scott.salgrade@orc1.world;
```

```
/*****************************************************************
CONNECT AS SCOTT/TIGER TO EXPORT

Use the Oracle export utility to generate the export file that you will transfer
to the new snapshot site. The export file will contain the base tables of your
temporary snapshots. See Oracle8i Utilities for additional information.

NOTE: The following example is to be used for Oracle8i databases
only. Base tables in database versions earlier than Oracle8i are
preceded by the SNAP$ prefix (that is, SNAP$_SNAP_EMP).
*****************************************************************/

exp scott/tiger@orc1.world TABLES='snap_emp','snap_dept',
'snap_bonus','snap_salgrade'

/*****************************************************************
DELETE THE TEMPORARY SNAPSHOTS

After you have completed your export, you should delete the temporary
snapshots that you created during the beginning of this procedure.
*****************************************************************/

CONNECT scott/tiger@orc1.world

DROP SNAPSHOT snap_emp;
DROP SNAPSHOT snap_dept;
DROP SNAPSHOT snap_bonus;
DROP SNAPSHOT snap_salgrade;

/*****************************************************************
CREATE NECESSARY SCHEMA AND DATABASE LINK

Before you perform the offline instantiation of your snapshots, create the
schema that will contain the snapshots at the new snapshot site (which need to
be in the same schema that contains the master objects at the master site) and
the database link from the snapshot site to the master site.
*****************************************************************/
CONNECT system/manager@snap2.world

CREATE USER scott IDENTIFIED by tiger;

GRANT connect, resource, create snapshot TO scott;

CONNECT scott/tiger@snap2.world
```

```
CREATE DATABASE LINK orc1.world CONNECT TO scott IDENTIFIED by tiger;

/*************************************************************************
CREATE EMPTY SNAPSHOT GROUP

Execute the DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP API at
the new snapshot site to contain an empty snapshot group that you
will add your snapshots to.
*************************************************************************/

CONNECT snapadmin/snapadmin@snap2.world

BEGIN
   DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP (
      GNAME => 'scott_mg',
      MASTER => 'orc1.world',
      PROPAGATION_MODE => 'ASYNCHRONOUS');
END;
/

/*************************************************************************
PREPARE SNAPSHOT SITE FOR OFFLINE INSTANTIATION

The DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD API creates the necessary support
mechanisms for the new snapshots. This step also adds the new
snapshots to the snapshot group that you created in the previous step.

Be sure to execute the DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD API for each
snapshot that you will be importing.
*************************************************************************/

CONNECT system/manager@snap2.world

BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      MASTER_SITE => 'orc1.world',
      SNAPSHOT_ONAME => 'snap_emp');
END;
/
```

```
BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      MASTER_SITE => 'orc1.world',
      SNAPSHOT_ONAME => 'snap_dept');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      MASTER_SITE => 'orc1.world',
      SNAPSHOT_ONAME => 'snap_bonus');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      MASTER_SITE => 'orc1.world',
      SNAPSHOT_ONAME => 'snap_salgrade');
END;
/

/*********************************************************************
CONNECT AS SCOTT/TIGER TO IMPORT AT NEW SNAPSHOT SITE

Use the Oracle import utility to import the file that you exported earlier.
Make sure that you import your data as the same user who exported the data (that
is, scott/tiger).
*********************************************************************/

imp scott/tiger@snap2.world FULL=y IGNORE=y
```

```
/*************************************************************************
COMPLETE THE OFFLINE INSTANTIATION

Execute the DBMS_OFFLINE_SNAPSHOT.END_LOAD API to finish the offline
instantiation of the imported snapshots.
*************************************************************************/

CONNECT system/manager@snap2.world

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      SNAPSHOT_ONAME => 'snap_emp');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      SNAPSHOT_ONAME => 'snap_dept');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      SNAPSHOT_ONAME => 'snap_bonus');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      GNAME => 'scott_mg',
      SNAME => 'scott',
      SNAPSHOT_ONAME => 'snap_salgrade');
END;
/
```

```
/*************************************************************************
REFRESH SNAPSHOTS TO REGISTER AT MASTER SITE

In addition to retrieving the latest changes from the master tables,
refreshing the snapshots at the new snapshot site registers the offline
instantiated snapshots at the target master site.
*************************************************************************/

CONNECT scott/tiger@snap2.world

BEGIN
   DBMS_SNAPSHOT.REFRESH ('snap_emp');
END;
/

BEGIN
   DBMS_SNAPSHOT.REFRESH ('snap_dept');
END;
/

BEGIN
   DBMS_SNAPSHOT.REFRESH ('snap_bonus');
END;
/

BEGIN
   DBMS_SNAPSHOT.REFRESH ('snap_salgrade');
END;
/
```

Determining Differences Between Replicated Tables

When administering a replicated environment, you may want to check, periodically, whether the contents of two replicated tables are identical. The following procedures in the DBMS_RECTIFIER_DIFF package let you identify, and optionally rectify, the differences between two tables when both sites are Oracle release 7.3 or higher.

DIFFERENCES

The DIFFERENCES procedure compares two replicas of a table, and determines all rows in the first replica that are not in the second and all rows in the second that are not in the first. The output of this procedure is stored in two user-created tables. The first table stores the values of the missing rows, and the second table is used to indicate which site contains each row.

RECTIFY

The RECTIFY procedure uses the information generated by the DIFFERENCES procedure to rectify the two tables. Any rows found in the first table and not in the second are inserted into the second table. Any rows found in the second table and not in the first are deleted from the second table.

To restore equivalency between all copies of a replicated table, complete the following steps:

1. Select one copy of the table to be the "reference" table. This copy will be used to update all other replicas of the table as needed.
2. Determine if it is necessary to check all rows and columns in the table for differences, or only a subset.

   For example, it may not be necessary to check rows that have not been updated since the last time that you checked for differences. Although it is not necessary to check all columns, your column list must include all columns that make up the primary key (or that you designated as a substitute identity key) for the table.
3. After determining which columns you will be checking in the table, create two tables to hold the results of the comparison.

   You must create one table that can hold the data for the columns being compared. For example, if you decide to compare the EMPNO, SAL, and

BONUS columns of the EMPLOYEE table, your CREATE statement would need to be similar to the following:

```
CREATE TABLE missing_rows_data (
  empno    NUMBER,
  sal      NUMBER,
  bonus    NUMBER);
```

You must also create a table that indicates where the row is found. This table must contain three columns with the datatypes shown in the following example:

```
CREATE TABLE missing_rows_location (
  present     VARCHAR2(128),
  absent      VARCHAR2(128),
  r_id        ROWID);
```

4. Suspend replication activity for the object group containing the tables that you want to compare. Although suspending replication activity for the group is not a requirement, rectifying tables that were not quiesced first can result in inconsistencies in your data.

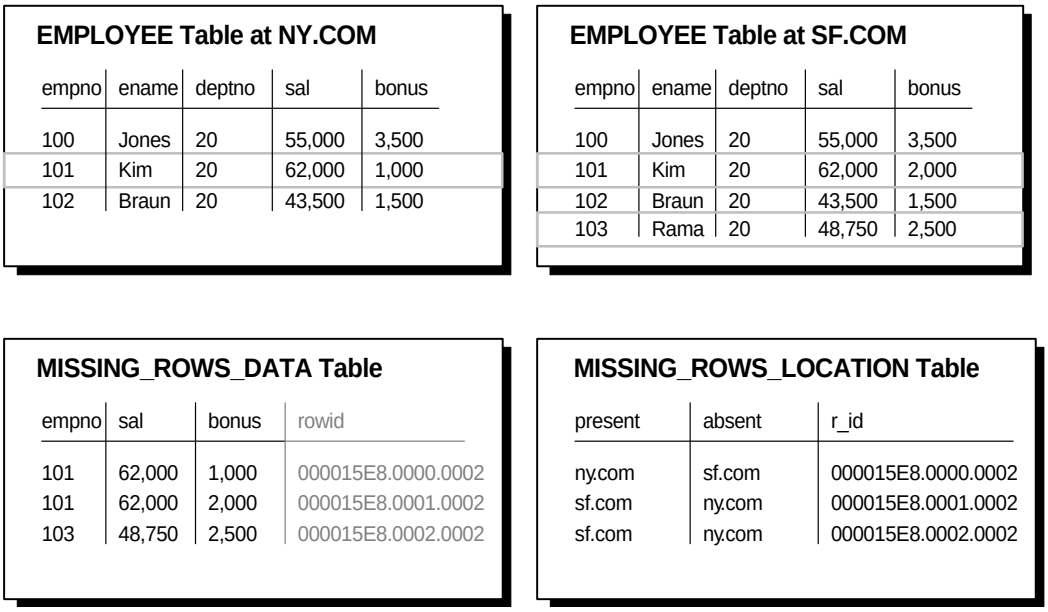
5. At the site containing the "reference" table, call the DBMS_RECTIFIER_DIFF.DIFFERENCES procedure.

   For example, if you wanted to compare the EMPLOYEE tables at the New York and San Francisco sites, your procedure call would look similar to the following:

```
BEGIN
   DBMS_RECTIFIER_DIFF.DIFFERENCES(
      sname1              =>   'hr',
      oname1              =>   'employee',
      reference_site      =>   'ny.com',
      sname2              =>   'hr',
      oname2              =>   'employee',
      comparison_site     =>   'sf.com',
      where_clause        =>   '',
      column_list         =>   'empno,sal,bonus',
      missing_rows_sname  =>   'scott',
      missing_rows_oname1 =>   'missing_rows_data',
      missing_rows_oname2 =>   'missing_rows_location',
      missing_rows_site   =>   'ny.com',
      commit_rows         =>    50);
END;
/
```

Figure 7–1 shows an example of two replicas of the EMPLOYEE table and what the resulting missing rows tables would look like if you executed the DIFFERENCES procedure on these replicas.

Figure 7–1 Determining Differences Between Replicas



EMPLOYEE Table at NY.COM

empno	ename	deptno	sal	bonus
100	Jones	20	55,000	3,500
101	Kim	20	62,000	1,000
102	Braun	20	43,500	1,500

EMPLOYEE Table at SF.COM

empno	ename	deptno	sal	bonus
100	Jones	20	55,000	3,500
101	Kim	20	62,000	2,000
102	Braun	20	43,500	1,500
103	Rama	20	48,750	2,500

MISSING_ROWS_DATA Table

empno	sal	bonus	rowid
101	62,000	1,000	000015E8.0000.0002
101	62,000	2,000	000015E8.0001.0002
103	48,750	2,500	000015E8.0002.0002

MISSING_ROWS_LOCATION Table

present	absent	r_id
ny.com	sf.com	000015E8.0000.0002
sf.com	ny.com	000015E8.0001.0002
sf.com	ny.com	000015E8.0002.0002

Notice that the two missing rows tables are related by the ROWID and R_ID columns.

6. Rectify the table at the "comparison" site to be equivalent to the table at the "reference" site by calling the DBMS_RECTIFIER_DIFF.RECTIFY procedure as shown in the following example:

```
BEGIN
   DBMS_RECTIFIER_DIFF.RECTIFY(
      sname1              =>   'hr',
      oname1              =>   'employee',
      reference_site      =>   'ny.com',
      sname2              =>   'hr',
      oname2              =>   'employee',
      comparison_site     =>   'sf.com',
      column_list         =>   'empno,sal,bonus',
      missing_rows_sname  =>   'scott',
      missing_rows_oname1 =>   'missing_rows_data',
      missing_rows_oname2 =>   'missing_rows_location',
      missing_rows_site   =>   'ny.com',
      commit_rows          =>   50);
END;
/
```

   The RECTIFY procedure temporarily disables replication at the "comparison" site while it performs the necessary insertions and deletions, as you would not want to propagate these changes. RECTIFY first performs all of the necessary DELETEs and then performs all of the INSERTs. This ensures that there are no violations of a PRIMARY KEY constraint.

   After you have successfully executed the RECTIFY procedure, your missing rows tables should be empty.

   Caution: If you have any additional constraints on the "comparison" table, you must ensure that they are not violated when you call RECTIFY. You may need to update the table directly using the information from the missing rows table. If so, be certain to DELETE the appropriate rows from the missing rows tables.

7. Repeat Steps 5 and 6 for the remaining copies of the replicated table. Remember to use the same "reference" table each time to ensure that all copies are identical when you complete this procedure.

8. Resume replication activity for the master group.

Updating the Comments Fields in Data Dictionary Views

Several procedures in the DBMS_REPCAT package let you update the comment information in the various data dictionary views associated with replication. Table 7–1 lists the appropriate procedure to call for each view.

Table 7–1 Updating Comments in Advanced Replication Facility Views

View	DBMS_REPCAT Procedure	See for Parameter Information

DBA_REPGROUP	`COMMENT_ON_REPGROUP(`
	`gname IN VARCHAR2,`
	`Comment IN VARCHAR2)`
		COMMENT_ON_REPGROUP procedure on page 8-94.

DBA_REPOBJECT	`COMMENT_ON_REPOBJECT(`
	`sname IN VARCHAR2,`
	`oname IN VARCHAR2,`
	`type IN VARCHAR2,`
	`comment IN VARCHAR2)`
		COMMENT_ON_REPOBJECT procedure on page 8-95.

DBA_REPSITES	`COMMENT_ON_REPSITES(`
	`gname IN VARCHAR2,`
	`master IN VARCHAR,`
	`comment IN VARCHAR2)`
		COMMENT_ON_REPSITES procedure on page 8-97.

DBA_ REPCOLUMN_ GROUP	`COMMENT_ON_COLUMN_GROUP(`
	`sname IN VARCHAR2,`
	`oname IN VARCHAR2,`
	`column_group IN VARCHAR2,`
	`comment IN VARCHAR2)`
		COMMENT_ON_COLUMN_GROUP procedure on page 8-92.

DBA_ REPPRIORITY_ GROUP	`COMMENT_ON_PRIORITY_GROUP(`
	`gname IN VARCHAR2,`
	`pgroup IN VARCHAR2)`
	`comment IN VARCHAR2)`
		COMMENT_ON_PRIORITY_ GROUP/COMMENT_ON_SITE_ PRIORITY procedures on page 8-93.

DBA_ REPPRIORITY_ GROUP (site priority group)	`COMMENT_ON_SITE_PRIORITY(`
	`gname IN VARCHAR2,`
	`name IN VARCHAR2,`
	`comment IN VARCHAR2)`
		COMMENT_ON_PRIORITY_ GROUP/COMMENT_ON_SITE_ PRIORITY procedures on page 8-93.

Table 7–1 Updating Comments in Advanced Replication Facility Views

View	DBMS_REPCAT Procedure	See for Parameter Information
DBA_REPRESOLUTION (uniqueness conflicts)	`COMMENT_ON_UNIQUE_RESOLUTION( sname IN VARCHAR2, oname IN VARCHAR2, constraint_name IN VARCHAR2, sequence_no IN NUMBER, Comment IN VARCHAR2)`	The parameters for the COMMENT_ON_UNIQUE_RESOLUTION procedures are described in COMMENT_ON_conflicttype_RESOLUTION procedure on page 8-99.
DBA_REPRESOLUTION (update conflicts)	`COMMENT_ON_UPDATE_RESOLUTION( sname IN VARCHAR2, oname IN VARCHAR2, column_group IN VARCHAR2, sequence_no IN NUMBER, Comment IN VARCHAR2)`	The parameters for the COMMENT_ON_UNIQUE_RESOLUTION procedures are described in COMMENT_ON_conflicttype_RESOLUTION procedure on page 8-99.
DBA_REPRESOLUTION (delete conflicts)	`COMMENT_ON_DELETE_RESOLUTION( sname IN VARCHAR2, oname IN VARCHAR2, sequence_no IN NUMBER, comment IN VARCHAR2)`	The parameters for the COMMENT_ON_UNIQUE_RESOLUTION procedures are described in COMMENT_ON_conflicttype_RESOLUTION procedure on page 8-99.

8

Replication Management API Reference

All installations of Oracle replication include the replication management application programming interface (API). This *replication management API* is a collection of PL/SQL packages that administrators use to configure and manage replication features at each site. Oracle Replication Manager also uses the procedures and functions of each site's replication management API to perform work.

This chapter describes the packages that constitute the Oracle replication management API, including:

- The procedures and functions in each package
- The parameters for each packaged procedure or function
- Exceptions that each procedure or function can raise

> **Note:** Some of the PL/SQL procedures and functions described in this chapter are overloaded. That is, two or more procedures or functions have the same name in a single package, but their formal parameters differ in number, order, or datatype family. When a procedure or function is overloaded, it is noted in the description. See the *PL/SQL User's Guide and Reference* for more information about overloading and for more information about PL/SQL in general.

Packages

Oracle's replication management API includes the following packages:

- DBMS_DEFER Package
- DBMS_DEFER_QUERY Package
- DBMS_DEFER_SYS Package
- DBMS_OFFLINE_OG Package
- DBMS_OFFLINE_SNAPSHOT Package
- DBMS_RECTIFIER_DIFF Package
- DBMS_REFRESH Package
- DBMS_REPCAT Package
- DBMS_REPCAT_ADMIN Package
- DBMS_REPCAT_INSTANTIATE Package
- DBMS_REPCAT_RGT Package
- DBMS_REPUTIL Package
- DBMS_SNAPSHOT Package

Examples of Using Oracle's Replication Management API

To use Oracle's replication management API, you issue procedure or function calls using a query tool such as an Enterprise Manager SQL Worksheet or SQL*Plus. For example, the following call to the DBMS_REPCAT.CREATE_MASTER_REPOBJECT procedure creates a new replicated table SALES.EMP in the ACCT replication group.

```
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
     sname                => 'sales',
     oname                => 'emp',
     type                 => 'table',
     use_existing_object  => TRUE,
     ddl_text             => 'CREATE TABLE acct_rec.emp AS . . .',
     comment              => 'created by . . .',
     retry                => FALSE,
     copy_rows            => TRUE,
     gname                => 'acct');
```

To call a replication management API function, you must provide an environment to receive the return value of the function. For example, the following anonymous PL/SQL block calls the DBMS_DEFER_SYS.DISABLED function in an IF statement.

```
BEGIN
 IF DBMS_DEFER_SYS.DISABLED('inst2') THEN
  DBMS_OUTPUT.PUT_LINE('Propagation to INST2 is disabled.');
 ELSE
  DBMS_OUTPUT.PUT_LINE('Propagation to INST2 is enabled.');
 END IF;
END;
```

Issues to Consider

For many procedures and functions in the replication management API, there are important issues to consider. For example:

- Some procedures or functions are appropriate to call only from the master definition site in a multimaster configuration.
- To perform some administrative operations for master groups, you must first suspend replication activity for the group before calling replication management API procedures and functions.
- The order in which you call different procedures and functions in Oracle's replication management API is extremely important. See the next section for more information about learning how to correctly issue replication management calls.

Replication Manager and Oracle Replication Management API

Oracle Replication Manager uses the replication management API to perform most of its functions. Using Replication Manager is much more convenient than issuing replication management API calls individually because the utility:

- Provides a GUI interface to type in and adjust API call parameters.
- Automatically orders numerous, related API calls in the proper sequence.
- Displays output returned from API calls in message boxes and error files.

An easy way to learn how to use Oracle's replication management API is to use the Replication Manager scripting feature. When you start an administrative session with Replication Manager, turn scripting on. When you are finished, turn scripting off and then review the script file. The script file contains all replication management API calls that were made during the session. See the Replication Manager help documentation for more information about its scripting feature.

DBMS_DEFER Package

Summary of Subprograms

Table 8–1 DBMS_DEFER Package Subprograms

Subprogram	Description
CALL procedure on page 8-4	Builds a deferred call to a remote procedure.
COMMIT_WORK procedure on page 8-6	Performs a transaction commit after checking for well-formed deferred remote procedure calls.
datatype_ARG procedure on page 8-7	Provides the data that is to be passed to a deferred remote procedure call.
TRANSACTION procedure on page 8-9	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 :=''});
```

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

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. **Note:** You must include all of the arguments for the procedure, even if some of the parameters have defaults.
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.

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.

You must specify each parameter in your procedure using the *datatype*_ARG procedure after you execute DBMS_DEFER.CALL. That is, you cannot use the default parameters for the deferred remote procedure call. For example, suppose you have the following procedure:

```
CREATE OR REPLACE PACKAGE my_pack AS
   PROCEDURE my_proc(a VARCHAR2, b VARCHAR2 DEFAULT 'SALES');
END;
/
```

When you run the DBMS_DEFER.CALL procedure, you must include a separate line for each parameter in the MY_PROC procedure:

```
CREATE OR REPLACE PROCEDURE load_def_tx IS
   node DBMS_DEFER.NODE_LIST_T;
BEGIN
   node(1) := 'MYCOMPUTER.WORLD';
   node(2) := NULL;
   DBMS_DEFER.TRANSACTION(node);
   DBMS_DEFER.CALL('SCOTT', 'MY_PACK', 'MY_PROC', 2);
   DBMS_DEFER.VARCHAR2_ARG('TEST');
   DBMS_DEFER.VARCHAR2_ARG('SALES'); -- required, cannot omit to use default
END;
/
```

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);
```

> **Note:** This 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.

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.

DBMS_DEFER_QUERY Package

Summary of Subprograms

Table 8–10 DBMS_DEFER_QUERY Package Subprograms

Subprogram	Description
GET_ARG_FORM function on page 8-11	Determines the form of an argument in a deferred call.
GET_ARG_TYPE function on page 8-12	Determines the type of an argument in a deferred call.
GET_CALL_ARGS procedure on page 8-14	Returns the text version of the various arguments for the specified call.
GET_datatype_ARG function on page 8-15	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:** The Replication Manager online help for information about displaying deferred transactions and error transactions in Replication Manager.

Syntax

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

Parameters

Table 8–11 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.
deferred_tran_id	Deferred transaction ID.

Exceptions

Table 8–12 GET_ARG_FORM Function Exceptions

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

Returns

Table 8–13 GET_ARG_Form Function Returns

Return Value	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 remote procedure call (RPC) parameter is returned.

> **See Also:** The Replication Manager online help for information about displaying deferred transactions and error transactions in Replication Manager.

Syntax

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

Parameters

Table 8–14 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 8–15 GET_ARG_TYPE Function Exceptions

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

Returns

Table 8–16 GET_ARG_TYPE Function Returns

Return Value	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 specified 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 8–17 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 8–18 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:** The Replication Manager online help for information about displaying deferred transactions and error transactions in Replication Manager.

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 8–19 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 1.
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 8–20 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.

DBMS_DEFER_SYS Package

Summary of Subprograms

Table 8–21 DBMS_DEFER_SYS Package Subprograms **(Page 1 of 2)**

Subprogram	Description
ADD_DEFAULT_DEST procedure on page 8-19	Adds a destination database to the DEFDEFAULTDEST view.
DELETE_DEFAULT_DEST procedure on page 8-19	Removes a destination database from the DEFDEFAULTDEST view.
DELETE_DEF_DESTINATION procedure on page 8-20	Removes a destination database from the DEFSCHEDULE view.
DELETE_ERROR procedure on page 8-20	Deletes a transaction from the DEFERROR view.
DELETE_TRAN procedure on page 8-21	Deletes a transaction from the DEFTRANDEST view.
DISABLED function on page 8-22	Determines whether propagation of the deferred transaction queue from the current site to a specified site is enabled.
EXCLUDE_PUSH procedure on page 8-23	Acquires an exclusive lock that prevents deferred transaction PUSH.
EXECUTE_ERROR procedure on page 8-24	Re-executes a deferred transaction that did not initially complete successfully in the security context of the original receiver of the transaction.
EXECUTE_ERROR_AS_USER procedure on page 8-25	Re-executes a deferred transaction that did not initially complete successfully in the security context of the user who executes this procedure.
PURGE function on page 8-26	Purges pushed transactions from the deferred transaction queue at your current master or snapshot site.
PUSH function on page 8-28	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 8-31	Registers the specified user as the propagator for the local database.

Table 8–21 DBMS_DEFER_SYS Package Subprograms **(Page 2 of 2)**

Subprogram	Description
SCHEDULE_PURGE procedure on page 8-32	Schedules a job to purge pushed transactions from the deferred transaction queue at your current master or snapshot site.
SCHEDULE_PUSH procedure on page 8-34	Schedules a job to push the deferred transaction queue to a remote master destination.
SET_DISABLED procedure on page 8-36	Disables or enables propagation of the deferred transaction queue from the current site to a specified destination site.
UNREGISTER_PROPAGATOR procedure on page 8-37	Unregisters a user as the propagator from the local database.
UNSCHEDULE_PURGE procedure on page 8-38	Stops automatic purges of pushed transactions from the deferred transaction queue at a snapshot or master site.
UNSCHEDULE_PUSH procedure on page 8-38	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 8–22 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 8–23 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 8–24 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 8–25 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

This procedure deletes a transaction from the DEFERROR view.

Syntax

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

Parameters

Table 8–26 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

This procedure deletes 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 8–27 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

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

Syntax

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

Parameters

Table 8–28 DISABLED Function Parameters

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

Returns

Table 8–29 DISABLED Function Return Values

Value	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 8–30 DISABLED Function Exceptions

Exception	Description
NO_DATA_FOUND	Specified destination does not appear in the DEFSCHEDULE view.

EXCLUDE_PUSH procedure

This function acquires an exclusive lock that prevents deferred transaction PUSH (either serial or parallel). This function performs 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 8–31 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

E_PUSH

Table 8–32 EXCLUDE_PUSH Function Return Values

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

EXECUTE_ERROR procedure

This procedure re-executes a deferred transaction that did not initially complete successfully in the security context of the original receiver of the transaction.

Syntax

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

Parameters

Table 8–33 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. If the provided database name is not fully qualified or is invalid, no error will be raised.

Exceptions

Table 8–34 EXECUTE_ERROR Procedure Exceptions

Exception	Description
ORA-24275 error	Illegal combinations of NULL and non-NULL parameters were used.
badparam	Parameter value missing or invalid (for example, if destination is NULL).
missinguser	Invalid user.

EXECUTE_ERROR_AS_USER procedure

This procedure re-executes a deferred transaction that did not initially complete successfully. Each transaction is executed in the security context of the connected user.

Syntax

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

Parameters

Table 8–35 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.
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 8–36 EXECUTE_ERROR_AS_USER Procedure Exceptions

Exception	Description
ORA-24275 error	Illegal combinations of NULL and non-NULL parameters were used.
badparam	Parameter value missing or invalid (for example, if destination is NULL).
missinguser	Invalid user.

PURGE function

This function purges 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 8–37 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. If you use purge_method_quick, deferred transactions and deferred procedure calls that have been successfully pushed may remain in the DEFTRAN and DEFCALL data dictionary views for longer than expected before they are purged. See "Usage Notes" on page 8-27 for more information.
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 shut down 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 8–38 Purge Function Return Values

Value	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 8–39 PURGE Function Exceptions

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

Usage Notes

When you use the purge_method_quick for the purge_method parameter in the DBMS_DEFER_SYS.PURGE function, deferred transactions and deferred procedure calls may remain in the DEFCALL and DEFTRAN data dictionary views after they have been successfully pushed. This behavior occurs in replication environments that have more than one database link and the push is executed to only one database link.

To purge the deferred transactions and deferred procedure calls, perform one of the following actions:

- Use `purge_method_precise` for the `purge_method` parameter instead of the `purge_method_quick`. Using `purge_method_precise` is more expensive, but it ensures that the deferred transactions and procedure calls are purged after they have been successfully pushed.
- Using `purge_method_quick` for the `purge_method` parameter, push the deferred transactions to all database links. The deferred transactions and deferred procedure calls are purged efficiently when the push to the last database link is successful.

PUSH function

This function forces a deferred remote procedure (RPC) call queue at your current master or snapshot site to be pushed (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 8–40 PUSH Function Parameters **(Page 1 of 2)**

Parameter	Description
destination	The fully qualified database name of the master to which you are forwarding changes.
parallelism	0 specifies serial propagation; *n* > 1 specifies parallel propagation with *n* parallel server processes; 1 specifies 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 shut down (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 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. The execution_seconds parameter only controls the duration of time that operations can be started. It does not include the amount of time that the transactions require at remote sites. Therefore, the execution_seconds parameter is not intended to be used as a precise control to stop the propagation of transactions to a remote site. If a precise control is required, use the transaction_count or delivery_order parameters.
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.

Table 8–40 PUSH Function Parameters **(Page 2 of 2)**

Parameter	Description
delivery_order_limit	Stop execution cleanly before pushing a transaction where delivery_order >= delivery_order_limit

Returns

PUSH

Table 8–41 PUSH Function Returns

Value	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 8–42 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 specified user as the propagator for the local database. It also grants to the specified 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 8–43 REGISTER_PROPAGATOR Procedure Parameters

Parameter	Description
username	Name of the user.

Exceptions

Table 8–44 REGISTER_PROPAGATOR Procedure Exceptions

Exception	Description
missinguser	Specified user does not exist.
alreadypropagator	Specified user is already the propagator.
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 8–45 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, it is created with a value of NULL. If you do not supply a value for this field, you must supply a value for next_date.
next_date	Allows you to specify a 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, 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 shut down 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 8–46 SCHEDULE_PUSH Procedure Parameters **(Page 1 of 2)**

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, 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 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.

Table 8–46 SCHEDULE_PUSH Procedure Parameters **(Page 2 of 2)**

Parameter	Description
reset	Set to TRUE to reset LAST_TXN_COUNT, LST_ERROR, and LAST_MSG to NULL.
parallelism	0 specifies serial propagation; *n* > 1 specifies parallel propagation with *n* parallel server processes; 1 specifies 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 shut down (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 specified destination site. If the disabled parameter is TRUE, then the procedure disables propagation to the specified 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 specified 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 specified destination and, although this does not push the queue, it permits future invocations of PUSH to push the queue to the specified 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 8–47 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 specified destination. Set this to FALSE to enable propagation.

Exceptions

Table 8–48 SET_DISABLED Procedure Exceptions

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

UNREGISTER_PROPAGATOR procedure

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

- Deletes the specified propagator from DEFPROPAGATOR.
- Revokes privileges granted by REGISTER_PROPAGATOR from the specified user (including identical privileges granted independently).
- Drops any generated wrappers in the schema of the specified 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 8–49 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 8–50 UNREGISTER_PROPAGATOR Procedure Exceptions

Parameter	Description
missingpropagator	Specified 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 8–51 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 8–52 UNSCHEDULE_PUSH Procedure Exceptions

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

DBMS_OFFLINE_OG Package

Summary of Subprograms

Table 8–53 DBMS_OFFLINE_OG Package Subprograms

Subprogram	Description
BEGIN_INSTANTIATION procedure on page 8-40	Starts offline instantiation of a replicated master group.
BEGIN_LOAD procedure on page 8-41	Disables triggers while data is imported to new master site as part of offline instantiation.
END_INSTANTIATION procedure on page 8-42	Completes offline instantiation of a replicated master group.
END_LOAD procedure on page 8-44	Re-enables triggers after importing data to new master site as part of offline instantiation.
RESUME_SUBSET_OF_MASTERS procedure on page 8-45	Resumes replication activity at all existing sites except the new site during offline instantiation of a replicated master group.

BEGIN_INSTANTIATION procedure

This procedure starts offline instantiation of a replicated master group. You must call this procedure from the master definition site.

Note: This procedure is used in performing an offline instantiation of a master table in a multimaster replication environment.

This 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 use.

Syntax

```
DBMS_OFFLINE_OG.BEGIN_INSTANTIATION (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2
   fname     IN   VARCHAR2);
```

Parameters

Table 8–54 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 parameter is for internal use only. Do not set this parameter unless directed to do so by Oracle Worldwide Support.

Exceptions

Table 8–55 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	Specified 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 the new master site as part of offline instantiation. You must call this procedure from the new master site.

> **Note:** This procedure is used in performing an offline instantiation of a master table in a multimaster replication environment.
>
> This 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 use.

Syntax

```
DBMS_OFFLINE_OG.BEGIN_LOAD (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2);
```

Parameters

Table 8–56 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 8–57 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	Specified 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.

Note: This procedure is used in performing an offline instantiation of a master table in a multimaster replication environment.

This 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 use.

Syntax

```
DBMS_OFFLINE_OG.END_INSTANTIATION (
  gname     IN  VARCHAR2,
  new_site  IN  VARCHAR2);
```

Parameters

Table 8–58 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 8–59 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	Specified 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.

Note: This procedure is used in performing an offline instantiation of a master table in a multimaster replication environment.

This 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 use.

Syntax

```
DBMS_OFFLINE_OG.END_LOAD (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2
   fname     IN   VARCHAR2);
```

Parameters

Table 8–60 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 parameter is for internal use only. Do not set this parameter unless directed to do so by Oracle Worldwide Support.

Exceptions

Table 8–61 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	Specified 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.

Note: This procedure is used in performing an offline instantiation of a master table in a multimaster replication environment.

This 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 use.

Syntax

```
DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS (
   gname     IN  VARCHAR2,
   new_site  IN  VARCHAR2
   override  IN  BOOLEAN := FALSE);
```

Parameters

Table 8–62 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 any pending RepCat administration requests are ignored and normal replication activity is restored at each master as quickly as possible. The override parameter should be set to TRUE only in emergency situations. If this is FALSE, then normal replication activity is restored at each master only when there is no pending RepCat administration request for gname at that master.

Exceptions

Table 8–63 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	Specified 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.

DBMS_OFFLINE_SNAPSHOT Package

Summary of Subprograms

Table 8–64 DBMS_OFFLINE_SNAPSHOT Package Subprograms

Subprogram	Description
BEGIN_LOAD procedure on page 8-48	Prepares a snapshot site for import of a new snapshot as part of offline instantiation.
END_LOAD procedure on page 8-50	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.

Note: This procedure is used in performing an offline instantiation of a snapshot.

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_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their use.

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 8–65 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.
min_communication	If TRUE, then the update trigger sends the new value of a column only if the update statement modifies the column. Also, if TRUE, 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 8–66 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 specified snapshot at specified master site.
dbms_repcat. missingschema	Specified 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.

Note: This procedure is used in performing an offline instantiation of a snapshot.

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_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their use.

Syntax

```
DBMS_OFFLINE_SNAPSHOT.END_LOAD (
   gname           IN  VARCHAR2,
   sname           IN  VARCHAR2,
   snapshot_oname  IN  VARCHAR2);
```

Parameters

Table 8–67 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 8–68 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.

DBMS_RECTIFIER_DIFF Package

Summary of Subprograms

Table 8–69 DBMS_RECTIFIER_DIFF Package Subprograms

Subprogram	Description
DIFFERENCES procedure on page 8-53	Determines the differences between two tables.
RECTIFY procedure on page 8-56	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 8–70 DIFFERENCES Procedure Parameters **(Page 1 of 2)**

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.
comparison_site	Name of the comparison database site. The default, NULL, indicates the current site.

Table 8–70 DIFFERENCES Procedure Parameters **(Page 2 of 2)**

Parameter	Description
where_clause	Only rows satisfying this restriction are selected for comparison. The default, NULL, indicates all rows are compared.
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 a comma. The default, NULL, indicates that all columns will 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 that are missing from the table at COMPARISON site, and information about the rows at COMPARISON site that are missing from the table at REFERENCE site.
missing_rows_oname2	Name of the table at MISSING_ROWS_SITE that stores information 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.
max_missing	Integer that specifies the maximum number of rows that should be inserted into the missing_rows_oname table. If more than max_missing rows are 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 if the fragments are so different that the missing rows table has too many entries and there is 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 be issued only after all rows for a single table have been inserted or deleted.

Exceptions

Table 8–71 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 8–72 RECTIFY Procedure Parameters **(Page 1 of 2)**

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.
comparison_site	Name of the comparison database site. The default, NULL, indicates the current site.

Table 8–72 RECTIFY Procedure Parameters **(Page 2 of 2)**

Parameter	Description
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 a comma. The default, NULL, indicates that all columns will 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 that are missing from the table at COMPARISON site, and information about the rows at COMPARISON site that are missing from the table at REFERENCE site.
missing_rows_oname2	Name of the table at MISSING_ROWS_SITE that stores information 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 be issued only after all rows for a single table have been inserted or deleted.

Exceptions

Table 8–73 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.

DBMS_REFRESH Package

Summary of Subprograms

Table 8–74 DBMS_REFRESH Package Subprograms

Subprogram	Description
ADD procedure on page 8-60	Adds snapshots to a refresh group.
CHANGE procedure on page 8-61	Changes the refresh interval for a refresh group.
DESTROY procedure on page 8-63	Removes all of the snapshots from a refresh group and deletes the refresh group.
MAKE procedure on page 8-64	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 8-66	Manually refreshes a refresh group.
SUBTRACT procedure on page 8-67	Removes snapshots from a refresh group.

ADD procedure

This procedure adds snapshots to a refresh group.

> **See Also:** "ADD OBJECTS TO REFRESH GROUP" on page 5-8, and see Chapter 3, "Snapshot Concepts & Architecture" in *Oracle8i Replication* for more information.

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 8–75 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 refresh group.

> **See Also:** Chapter 3, "Snapshot Concepts & Architecture" in the *Oracle8i Replication* for more information.

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 8–76 CHANGE Procedures Parameters **(Page 1 of 2)**

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 refresh 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.

Table 8–76 CHANGE Procedures Parameters **(Page 2 of 2)**

Parameter	Description
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 (that is, parallelism is set to 1 or greater), then: ■ 0 = do not 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 *do not purge* and occasionally execute PUSH with purge set to *aggressive* to reduce the queue.
parallelism	0 specifies serial propagation; $n > 1$ specifies parallel propagation with n parallel server processes; 1 specifies 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:** Chapter 3, "Snapshot Concepts & Architecture" in the *Oracle8i Replication* for more information.

Syntax

```
DBMS_REFRESH.DESTROY (
   name    IN    VARCHAR2);
```

Parameters

Table 8–77 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:** "CREATE REFRESH GROUP" on page 5-6, and see Chapter 3, "Snapshot Concepts & Architecture" in *Oracle8i Replication*.

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 8–78 MAKE Procedure Parameters **(Page 1 of 2)**

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.
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.

Table 8–78 MAKE Procedure Parameters **(Page 2 of 2)**

Parameter	Description
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 = do not 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 *do not purge* and occasionally execute PUSH with purge set to *aggressive* to reduce the queue.
parallelism	0 specifies serial propagation; *n* > 1 specifies parallel propagation with *n* parallel server processes; 1 specifies 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:** Chapter 3, "Snapshot Concepts & Architecture" in *Oracle8i Replication*.

Syntax

```
DBMS_REFRESH.REFRESH (
   name    IN     VARCHAR2);
```

Table 8–79 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:** Chapter 3, "Snapshot Concepts & Architecture" in *Oracle8i Replication*.

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 8–80 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.

DBMS_REPCAT Package

Summary of Subprograms

Table 8–81 DBMS_REPCAT Package Subprograms **(Page 1 of 5)**

Subprogram	Description
ADD_GROUPED_COLUMN procedure on page 8-72	Adds members to an existing column group.
ADD_MASTER_DATABASE procedure on page 8-73	Adds another master site to your replicated environment.
ADD_PRIORITY_datatype procedure on page 8-75	Adds a member to a priority group.
ADD_SITE_PRIORITY_SITE procedure on page 8-76	Adds a new site to a site priority group.
ADD_conflicttype_RESOLUTION procedure on page 8-78	Designates a method for resolving an update, delete, or uniqueness conflict.
ALTER_MASTER_PROPAGATION procedure on page 8-82	Alters the propagation method for a specified object group at a specified master site.
ALTER_MASTER_REPOBJECT procedure on page 8-83	Alters an object in your replicated environment.
ALTER_PRIORITY procedure on page 8-85	Alters the priority level associated with a specified priority group member.
ALTER_PRIORITY_datatype procedure on page 8-86	Alters the value of a member in a priority group.
ALTER_SITE_PRIORITY procedure on page 8-88	Alters the priority level associated with a specified site.
ALTER_SITE_PRIORITY_SITE procedure on page 8-89	Alters the site associated with a specified priority level.
ALTER_SNAPSHOT_PROPAGATION procedure on page 8-90	Alters the propagation method for a specified object group at the current snapshot site.
CANCEL_STATISTICS procedure on page 8-91	Stops collecting statistics about the successful resolution of update, uniqueness, and delete conflicts for a table.
COMMENT_ON_COLUMN_GROUP procedure on page 8-92	Updates the comment field in the ALL_REPCOLUMN_GROUP view for a column group.

Table 8–81 DBMS_REPCAT Package Subprograms **(Page 2 of 5)**

Subprogram	Description
COMMENT_ON_PRIORITY_GROUP/COMMENT_ON_SITE_PRIORITY procedures on page 8-93	Updates the comment field in the ALL_REPPRIORITY_GROUP view for a (site) priority group.
COMMENT_ON_REPGROUP procedure on page 8-94	Updates the comment field in the ALL_REPGROUP view for a replicated master group.
COMMENT_ON_REPOBJECT procedure on page 8-95	Updates the comment field in the ALL_REPOBJECT view for a replicated object.
COMMENT_ON_REPSITES procedure on page 8-97	Updates the comment field in the ALL_REPSITE view for a replicated site.
COMMENT_ON_SNAPSHOT_REPSITES procedure on page 8-98	Updates the SCHEMA_COMMENT field in the ALL_REPGROUP view for a snapshot site.
COMMENT_ON_conflicttype_RESOLUTION procedure on page 8-99	Updates the comment field in the ALL_REPRESOLUTION view for a conflict resolution routine.
COMPARE_OLD_VALUES procedure on page 8-101	Compares old column values at each master site for each non-key column of a replicated table for updates and deletes.
CREATE_MASTER_REPGROUP procedure on page 8-103	Creates a new, empty, quiesced master replication object group.
CREATE_MASTER_REPOBJECT procedure on page 8-104	Indicates that an object is a replicated object.
CREATE_SNAPSHOT_REPGROUP procedure on page 8-107	Creates a new, empty snapshot replication object group in your local database.
CREATE_SNAPSHOT_REPOBJECT procedure on page 8-108	Adds a replicated object to your snapshot site.
DEFINE_COLUMN_GROUP procedure on page 8-110	Creates an empty column group
DEFINE_PRIORITY_GROUP procedure on page 8-111	Creates a new priority group for a replicated master group.
DEFINE_SITE_PRIORITY procedure on page 8-113	Creates a new site priority group for a replicated master group.

Table 8–81 DBMS_REPCAT Package Subprograms **(Page 3 of 5)**

Subprogram	Description
DO_DEFERRED_REPCAT_ADMIN procedure on page 8-114	Executes the local outstanding deferred administrative procedures for the specified replicated master group at the current master site, or for all master sites.
DROP_COLUMN_GROUP procedure on page 8-115	Drops a column group.
DROP_GROUPED_COLUMN procedure on page 8-116	Removes members from a column group.
DROP_MASTER_REPGROUP procedure on page 8-117	Drops a replicated master group from your current site.
DROP_MASTER_REPOBJECT procedure on page 8-118	Drops a replicated object from a replicated master group.
DROP_PRIORITY procedure on page 8-119	Drops a member of a priority group by priority level.
DROP_PRIORITY_GROUP procedure on page 8-120	Drops a priority group for a specified replicated master group.
DROP_PRIORITY_datatype procedure on page 8-121	Drops a member of a priority group by value.
DROP_SITE_PRIORITY procedure on page 8-123	Drops a site priority group for a specified replicated master group.
DROP_SITE_PRIORITY_SITE procedure on page 8-124	Drops a specified site, by name, from a site priority group.
DROP_SNAPSHOT_REPGROUP procedure on page 8-125	Drops a snapshot site from your replicated environment.
DROP_SNAPSHOT_REPOBJECT procedure on page 8-126	Drops a replicated object from a snapshot site.
DROP_conflicttype_ RESOLUTION procedure on page 8-127	Drops an update, delete, or uniqueness conflict resolution routine.
EXECUTE_DDL procedure on page 8-129	Supplies DDL that you want to have executed at each master site.
GENERATE_REPLICATION_ SUPPORT procedure on page 8-130	Generates the triggers, packages, and procedures needed to support replication.

Table 8–81 DBMS_REPCAT Package Subprograms **(Page 4 of 5)**

Subprogram	Description
GENERATE_SNAPSHOT_SUPPORT procedure on page 8-132	Activates triggers and generate packages needed to support the replication of updatable snapshots or procedural replication.
MAKE_COLUMN_GROUP procedure on page 8-134	Creates a new column group with one or more members.
PURGE_MASTER_LOG procedure on page 8-135	Removes local messages in the RepCatLog associated with a specified identification number, source, or replicated master group.
PURGE_STATISTICS procedure on page 136	Removes information from the ALL_REPRESOLUTION_STATISTICS view.
REFRESH_SNAPSHOT_REPGROUP procedure on page 8-137	Refreshes a snapshot site object group with the most recent data from its associated master site.
REGISTER_SNAPSHOT_REPGROUP procedure on page 8-138	Facilitates the administration of snapshots at their respective master sites by inserting, modifying, or deleting from DBA_REGISTERED_SNAPSHOT_GROUPS.
REGISTER_STATISTICS procedure on page 8-140	Collects information about the successful resolution of update, delete, and uniqueness conflicts for a table.
RELOCATE_MASTERDEF procedure on page 141	Changes your master definition site to another master site in your replicated environment.
REMOVE_MASTER_DATABASES procedure on page 8-142	Removes one or more master databases from a replicated environment.
REPCAT_IMPORT_CHECK procedure on page 8-143	Ensures that the objects in the replicated master 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 8-144	Resumes normal replication activity after quiescing a replicated environment.
SEND_OLD_VALUES procedure on page 8-145	Specifies whether to send old column values for each non-key column of a replicated table for updates and deletes.
SET_COLUMNS procedure on page 8-147	Specifies 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.
SUSPEND_MASTER_ACTIVITY procedure on page 8-149	Suspends replication activity for an object group.

Table 8–81 DBMS_REPCAT Package Subprograms **(Page 5 of 5)**

Subprogram	Description
SWITCH_SNAPSHOT_MASTER procedure on page 8-149	Changes the master database of a snapshot replicated master group to another master site.
UNREGISTER_SNAPSHOT_REPGROUP procedure on page 8-150	Facilitates the administration of snapshots at their respective master sites by inserting, modifying, or deleting from repcat$_repsite.
VALIDATE function on page 8-151	Validates the correctness of key conditions of a multiple master replication environment.
WAIT_MASTER_LOG procedure on page 8-154	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 8–82 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.

Table 8–83 ADD_GROUPED_COLUMN Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified table does not exist.
missinggroup	Specified column group does not exist.
missingcolumn	Specified column does not exist in the specified table.
duplicatecolumn	Specified column is already a member of another column group.
missingschema	Specified schema does not exist.
notquiesced	Object group to which the specified table belongs 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 8–84 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 Chapter 2, "Master Concepts & Architecture" in *Oracle8i Replication* 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 DBA_REPSITES view.
propagation_mode	Method of forwarding changes to and receiving changes from new master database. Accepted values are SYNCHRONOUS and ASYNCHRONOUS.
fname	This parameter is for internal use only. Do not set this parameter unless directed to do so by Oracle Worldwide Support.

Exceptions

Table 8–85 ADD_MASTER_DATABASE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Replicated master group has not been suspended.
missingrepgroup	Object group does not exist at the specified 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:** *Oracle8i Replication* for more information about conflict resolution methods.

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 8–86 ADD_PRIORITY_datatype Procedure Parameters

Parameter	Description
`gname`	Replicated master group for which you are creating a priority group.
`pgroup`	Name of the priority group.
`value`	Value of the priority group member. This is 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 8–87 ADD_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
duplicatevalue	Specified value already exists in the priority group.
duplicatepriority	Specified priority already exists in the priority group.
missingrepgroup	Specified replicated master group does not exist.
missingprioritygroup	Specified priority group does not exist.
typefailure	Specified value has the incorrect datatype for the priority group.
notquiesced	Specified replicated master 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:** *Oracle8i Replication* for more information about conflict resolution methods.

Syntax

```
DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
   gname          IN   VARCHAR2,
   name           IN   VARCHAR2
   site           IN   VARCHAR2,
   priority       IN   NUMBER);
```

Parameters

Table 8–88 ADD_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	Replicated master 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 8–89 ADD_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified replicated master group does not exist.
missingpriority	Specified site priority group does not exist.
duplicatepriority	Specified priority level already exists for another site in the group.
duplicatevalue	Specified site already exists in the site priority group.
notquiesced	Replicated master group is not quiesced.

ADD_*conflicttype*_RESOLUTION procedure

These procedures designate 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.

Table 8–90 ADD_conflicttype_RESOLUTION Procedures

Conflict Type	Procedure Name
update	ADD_UPDATE_RESOLUTION
uniqueness	ADD_UNIQUE_RESOLUTION
delete	ADD_DELETE_RESOLUTION

> **See Also:** *Oracle8i Replication* for more information about designating methods to resolve update conflicts, selecting uniqueness conflict resolution methods, and assigning delete conflict resolution methods.

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 8–91 ADD_conflicttype_RESOLUTION Procedure Parameters **(Page 1 of 2)**

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, and DISCARD (for uniqueness conflicts). There are no standard methods for delete conflicts.

Table 8–91 ADD_conflicttype_RESOLUTION Procedure Parameters **(Page 2 of 2)**

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.VARCHAR2. 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 *Oracle8i Replication*. If you are using a different method, 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 DBA_REPRESOLUTION view.

Exceptions

Table 8–92 ADD_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist as a table in the specified schema using row-level replication.
missingschema	Specified schema does not exist.
missingcolumn	Column that you specified as part of the PARAMETER_COLUMN_NAME argument does not exist.
missinggroup	Specified column group does not exist.
missingprioritygroup	The 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 to which the specified table belongs is not quiesced.
duplicateresolution	Specified conflict resolution method is already registered.
duplicatesequence	The specified sequence number already exists for the specified object.
invalidprioritygroup	The specified priority group does not exist.
paramtype	Type is different from the type assigned to the priority group.

ALTER_MASTER_PROPAGATION procedure

This procedure alters the propagation method for a specified object group at a specified 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 8–93 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 specified 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 DBA_REPPROP view.

Exceptions

Table 8–94 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 8–95 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. 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.
comment	If not NULL, then this comment is added to the COMMENT field of the DBA_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 8–96 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	Specified type parameter is not supported.
ddlfailure	DDL at the master definition site did not succeed.
commfailure	At least one master site is not accessible.

ALTER_PRIORITY procedure

This procedure alters the priority level associated with a specified priority group member. You must call this procedure from the master definition site.

> **See Also:** *Oracle8i Replication* for more information about conflict resolution methods.

Syntax

```
DBMS_REPCAT.ALTER_PRIORITY (
   gname            IN   VARCHAR2,
   pgroup           IN   VARCHAR2,
   old_priority     IN   NUMBER,
   new_priority     IN   NUMBER);
```

Parameters

Table 8–97 ALTER_PRIORITY Procedure Parameters

Parameter	Description
gname	Replicated master 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 8–98 ALTER_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
duplicatepriority	New priority level already exists in the priority group.
missingrepgroup	Specified replicated master group does not exist.
missingvalue	Value was not registered by a call to DBMS_REPCAT.ADD_PRIORITY_datatype.
missingprioritygroup	Specified priority group does not exist.
notquiesced	Specified replicated master 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:** *Oracle8i Replication* for more information about conflict resolution methods.

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 8–99 ALTER_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	Replicated master 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 8–100 ALTER_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
duplicatevalue	New value already exists in the priority group.
missingrepgroup	Specified replicated master group does not exist.
missingprioritygroup	Specified priority group does not exist.
missingvalue	Old value does not exist.
paramtype	New value has the incorrect datatype for the priority group.
typefailure	Specified value has the incorrect datatype for the priority group.
notquiesced	Specified replicated master group is not quiesced.

ALTER_SITE_PRIORITY procedure

This procedure alters the priority level associated with a specified site. You must call this procedure from the master definition site.

> **See Also:** *Oracle8i Replication* for more information about conflict resolution methods.

Syntax

```
DBMS_REPCAT.ALTER_SITE_PRIORITY (
   gname         IN   VARCHAR2,
   name          IN   VARCHAR2,
   old_priority  IN   NUMBER,
   new_priority  IN   NUMBER);
```

Parameters

Table 8–101 ALTER_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	Replicated master 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 8–102 ALTER_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified replicated master 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 master group is not quiesced.

ALTER_SITE_PRIORITY_SITE procedure

This procedure alters the site associated with a specified priority level. You must call this procedure from the master definition site.

> **See Also:** *Oracle8i Replication* for more information about conflict resolution methods.

Syntax

```
DBMS_REPCAT.ALTER_SITE_PRIORITY_SITE (
   gname     IN   VARCHAR2,
   name      IN   VARCHAR2,
   old_site  IN   VARCHAR2,
   new_site  IN   VARCHAR2);
```

Parameters

Table 8–103 ALTER_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	Replicated master 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 disassociate from the priority level.
new_site	New global database name that you want to associate with the current priority level.

Exceptions

Table 8–104 ALTER_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified replicated master group does not exist.
missingpriority	Specified site priority group does not exist.
missingvalue	Old site is not a group member.
notquiesced	Replicated master group is not quiesced

ALTER_SNAPSHOT_PROPAGATION procedure

This procedure alters the propagation method for a specified 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   := ''
   gowner               IN  VARCHAR2   := 'PUBLIC');
```

Parameters

Table 8–105 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 DBA_REPPROP view.
gowner	Owner of the snapshot group.

Exceptions

Table 8–106 ALTER_SNAPSHOT_PROPAGATION Procedure Exceptions

Exception	Description
missingrepgroup	Specified replicated master group does not exist.
typefailure	Propagation mode was specified incorrectly.
nonsnapshot	Current site is not a snapshot site for the specified object group.
commfailure	Cannot contact master.
notcompat	Compatibility mode must be 7.3.0.0 or greater.
failaltersnaprop	Snapshot group propagation can be altered only when there are no other snapshot groups with the same master sharing the snapshot site.

CANCEL_STATISTICS procedure

This procedure stops the collection of 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 8–107 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 8–108 CANCEL_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Specified schema does not exist.
missingobject	Specified table does not exist.
statnotreg	Specified table is not currently registered to collect statistics.

COMMENT_ON_COLUMN_GROUP procedure

This procedure updates the comment field in the DBA_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 8–109 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 DBA_REPCOLUMN_GROUP view.

Exceptions

Table 8–110 COMMENT_ON_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missinggroup	Specified column group does not exist.
missingobj	Object is missing.

COMMENT_ON_PRIORITY_GROUP/COMMENT_ON_SITE_PRIORITY procedures

COMMENT_ON_PRIORITY_GROUP updates the comment field in the DBA_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 DBA_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 8–111 COMMENT_ON_PRIORITY_GROUP and COMMENT_ON_SITE_PRIORITY Parameters

Parameter	Description
gname	Name of the replicated master 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 DBA_REPPRIORITY_GROUP view.

Exceptions

Table 8–112 COMMENT_ON_PRIORITY_GROUP and COMMENT_ON_SITE_PRIORITY Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified replicated master group does not exist.
missingprioritygroup	Specified priority group does not exist.

COMMENT_ON_REPGROUP procedure

This procedure updates the comment field in the DBA_REPGROUP view for a replicated master 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 8–113 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 DBA_REPGROUP view.

Exceptions

Table 8–114 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_REPOBJECT procedure

This procedure updates the comment field in the DBA_REPOBJECT view for a replicated object in a master group. 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 8–115 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. The types supported are: TABLE, INDEX, SYNONYM, TRIGGER, VIEW, PROCEDURE, FUNCTION, PACKAGE, and PACKAGE BODY.
comment	Text of the updated comment that you want to include in the OBJECT_COMMENT field of the DBA_REPOBJECT view.

Exceptions

Table 8–116 COMMENT_ON_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist.
typefailure	Specified type parameter is not supported.
commfailure	At least one master site is not accessible.

COMMENT_ON_REPSITES procedure

If the object group is a master group, this procedure updates the MASTER_COMMENT field in the DBA_REPSITES view for a master site. If the object group is a snapshot group, this procedure updates the SCHEMA_COMMENT field in the DBA_REPGROUP view for a snapshot site.

This procedure can be executed at either a master site or a snapshot site. If you execute this procedure on a a snapshot site, the snapshot group owner must be PUBLIC.

> **See Also:** "COMMENT_ON_SNAPSHOT_REPSITES procedure" on page 8-98 for instructions on placing a comment in the SCHEMA_COMMENT field of the DBA_REPGROUP view for a snapshot site if the snapshot group owner is not PUBLIC.

Syntax

```
DBMS_REPCAT.COMMENT_ON_REPSITES (
   gname       IN   VARCHAR2,
   [ master    IN   VARCHAR,]
   comment     IN   VARCHAR2);
```

Parameters

Table 8–117 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 on which you want to comment. If you are executing the procedure on a master site, this parameter is required. To update comments at a snapshot site, omit this parameter.
comment	Text of the updated comment that you want to include in the comment field of the appropriate dictionary view. If the site is a master site, this procedure updates the MASTER_COMMENT field of the DBA_REPSITES view. If the site is a snapshot site, this procedure updates the SCHEMA_COMMENT field of the DBA_REPGROUP view.

Exceptions

Table 8–118 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.
missingrepgroup	Object group does not exist.
commfailure	One or more master sites are not accessible.
corrupt	There is an inconsistency in the replication catalog views.

COMMENT_ON_SNAPSHOT_REPSITES procedure

This procedure updates the SCHEMA_COMMENT field in the DBA_REPGROUP data dictionary view for the specified snapshot group. The group name must be registered locally as a replicated snapshot group. This procedure must be executed at the snapshot site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_SNAPSHOT_REPSITES (
   gowner    IN   VARCHAR2,
   gname     IN   VARCHAR2,
   comment   IN   VARCHAR2);
```

Parameters

Table 8–119 COMMENT_ON_SNAPSHOT_REPSITES Procedure Parameters

Parameter	Description
gowner	Owner of the snapshot object group.
gname	Name of the snapshot object group.
comment	Updated comment to include in the SCHEMA_COMMENT field of the DBA_REPGROUP view.

Table 8–120 COMMENT_ON_SNAPSHOT_REPSITES Procedure Exceptions

Parameter	Description
missingrepgroup	The snapshot object group does not exist.
nonsnapshot	The connected site is not a snapshot site.

COMMENT_ON_*conflicttype*_RESOLUTION procedure

This procedure updates the comment field in the DBA_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.

Table 8–121 COMMENT_ON_conflicttype_RESOLUTION Procedures

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 8–122 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 DBA_REPRESOLUTION view.

Exceptions

Table 8–123 COMMENT_ON_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist.
missingresolution	Specified conflict resolution routine is not registered.

COMPARE_OLD_VALUES procedure

This procedure lets you compare old column values at each master site 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 8–124 COMPARE_OLD_VALUES Procedure Parameters **(Page 1 of 2)**

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 position 1, the second at position 2, and so on.
operation	Possible values are: UPDATE, DELETE, or the asterisk wildcard '*', which means update and delete.

Table 8–124 COMPARE_OLD_VALUES Procedure Parameters **(Page 2 of 2)**

Parameter	Description
compare	If compare is TRUE, the old values of the specified columns are compared when sent. If compare is FALSE, 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 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.

See *Oracle8i Replication* for information about reduced data propagation before changing the default behavior of Oracle.

Exceptions

Table 8–125 COMPARE_OLD_VALUES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist as a table in the specified schema awaiting row-level replication information.
missingcolumn	At least one column is not in the table.
notquiesced	Replicated master group has not been quiesced.
typefailure	An illegal operation is specified.

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 8–126 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 DBA_REPGROUP view.
master_comment	This comment is added to the DBA_REPSITES view.
qualifier	Connection qualifier for object group. Be sure to use the @ sign. See *Oracle8i Replication* for more information about connection qualifiers.

Exceptions

Table 8–127 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 makes an object a replicated object.

Replication of clustered tables is supported, but the use_existing_object parameter cannot be set to FALSE for clustered tables. In other words, the clustered table must be pre-created at all master sites participating in the master group. However, the pre-created tables do not need to contain the table data. So, the copy_rows parameter can be set to TRUE for clustered tables.

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

The following table describes the parameters for this procedure.

Table 8–128 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 specified 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 8–130 for more information. **Note:** This parameter must be set to TRUE for clustered tables.
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. If the DDL is supplied without specifying a schema (sname parameter), 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.
comment	This comment is added to the OBJECT_COMMENT field of the DBA_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 8–130 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 8–129 CREATE_MASTER_REPOBJECT Procedure Exceptions

Exceptions	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Replicated master group has not been suspended.
duplicateobject	Specified object already exists in the replicated master 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 specified 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 at least 7.3 compatibility mode.

Object Creations

Table 8–130 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.

CREATE_SNAPSHOT_REPGROUP procedure

This procedure creates a new, empty snapshot group in your local database. CREATE_SNAPSHOT_REPGROUP automatically calls REGISTER_SNAPSHOT_REPGROUP, but ignores any errors that may have happened during registration.

Syntax

```
DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP (
   gname              IN   VARCHAR2,
   master             IN   VARCHAR2,
   comment            IN   VARCHAR2     := '',
   propagation_mode   IN   VARCHAR2     := 'ASYNCHRONOUS',
   fname              IN   VARCHAR2     := NULL
   gowner               IN   VARCHAR2     := 'PUBLIC');
```

Parameters

Table 8–131 CREATE_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated master group. This object group must exist at the specified master site.
master	Fully qualified database name of the database in the replicated environment to use as the master. You can include a connection qualifier if necessary. See *Oracle8i Replication* and *Oracle8i Distributed Database Systems* for information about using connection qualifiers.
comment	This comment is added to the DBA_REPGROUP view.
propagation_mode	Method of propagation for all updatable snapshots in the object group. Acceptable values are SYNCHRONOUS and ASYNCHRONOUS.
fname	This parameter is for internal use only. Do not set this parameter unless directed to do so by Oracle Worldwide Support.
gowner	Owner of the snapshot group.

Exceptions

Table 8–132 CREATE_SNAPSHOT_REPGROUP Procedure Exceptions

Exception	Description
duplicaterepgroup	Object group already exists at the invocation site.
nonmaster	Specified database is not a master site.
commfailure	Specified database is not accessible.
norepopt	Advanced replication option is not installed.
typefailure	Propagation mode was specified incorrectly.
missingrepgroup	Replicated master group does not exist at master site.
invalidqualifier	Connection qualifier specified for master is not valid for the object group.
alreadymastered	At the local site, there is another snapshot group with the same group name, but different master site.

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
   gowner                 IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 8–133 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 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 master group of the master site designated for this schema, then Oracle raises a missingobject error. 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.
comment	This comment is added to the OBJECT_COMMENT field of the DBA_REPOBJECT view.
gname	Name of the replicated master 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 about conflict resolution methods, see *Oracle8i Replication*.
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.
gowner	Owner of the snapshot group.

Exceptions

Table 8–134 CREATE_SNAPSHOT_REPOBJECT Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
nonmaster	Master is no longer a master site.
missingobject	Specified object does not exist in the master's replicated master group.
duplicateobject	Specified 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 master group does not exist.

DEFINE_COLUMN_GROUP procedure

This procedure creates an empty column group. You must call this procedure from the master definition site.

> **See Also:** *Oracle8i Replication* for more information about conflict resolution methods.

Syntax

```
DBMS_REPCAT.DEFINE_COLUMN_GROUP (
   sname            IN   VARCHAR2,
   oname            IN   VARCHAR2,
   column_group     IN   VARCHAR2,
   comment          IN   VARCHAR2 := NULL);
```

Parameters

Table 8–135 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 DBA_REPCOLUMN_GROUP view.

Exceptions

Table 8–136 DEFINE_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified table does not exist.
duplicategroup	Specified column group already exists for the table.
notquiesced	Object group to which the specified table belongs is not quiesced.

DEFINE_PRIORITY_GROUP procedure

This procedure creates a new priority group for a replicated master group. You must call this procedure from the master definition site.

> **See Also:** *Oracle8i Replication* for more information about conflict resolution methods.

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 8–137 DEFINE_PRIORITY_GROUP Procedure Parameters

Parameter	Description
gname	Replicated master 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 DBA_REPPRIORITY view.

Exceptions

Table 8–138 DEFINE_PRIORITY_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified replicated master group does not exist.
duplicatepriority group	Specified priority group already exists in the replicated master group.
typefailure	Specified datatype is not supported.
notquiesced	Replicated master group is not quiesced.

DEFINE_SITE_PRIORITY procedure

This procedure creates a new site priority group for a replicated master group. You must call this procedure from the master definition site.

> **See Also:** *Oracle8i Replication* for more information about conflict resolution methods.

Syntax

```
DBMS_REPCAT.DEFINE_SITE_PRIORITY (
   gname        IN   VARCHAR2,
   name         IN   VARCHAR2,
   comment      IN   VARCHAR2 := NULL);
```

Parameters

Table 8–139 DEFINE_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	The replicated master 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 DBA_REPPRIORITY view.

Exceptions

Table 8–140 DEFINE_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified replicated master group does not exist.
duplicate prioritygroup	Specified site priority group already exists in the replicated master group.
notquiesced	Replicated master group is not quiesced.

DO_DEFERRED_REPCAT_ADMIN procedure

This procedure executes the local outstanding deferred administrative procedures for the specified replicated master group at the current master site, or (with assistance from job queues) for all master sites.

DO_DEFERRED_REPCAT_ADMIN executes only those administrative requests submitted by the connected user who called DO_DEFERRED_REPCAT_ADMIN. Requests submitted by other users are ignored.

Syntax

```
DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN (
   gname          IN   VARCHAR2,
   all_sites      IN   BOOLEAN := FALSE);
```

Parameters

Table 8–141 DO_DEFERRED_REPCAT_ADMIN Procedure Parameters

Parameter	Description
gname	Name of the replicated master group.
all_sites	If this is TRUE, then use a job to execute the local administrative procedures at each master site.

Exceptions

Table 8–142 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.

DROP_COLUMN_GROUP procedure

This procedure drops a column group. You must call this procedure from the master definition site.

> **See Also:** *Oracle8i Replication* for more information about conflict resolution methods.

Syntax

```
DBMS_REPCAT.DROP_COLUMN_GROUP (
   sname        IN   VARCHAR2,
   oname        IN   VARCHAR2,
   column_group IN   VARCHAR2);
```

Parameters

Table 8–143 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 8–144 DROP_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
referenced	Specified column group is being used in conflict detection and resolution.
missingobject	Specified table does not exist.
missinggroup	Specified column group does not exist.
notquiesced	Replicated master group to which the table belongs 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:** *Oracle8i Replication* for more information about conflict resolution methods.

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 8–145 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 8–146 DROP_GROUPED_COLUMN Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified table does not exist.
notquiesced	Replicated master group that the table belongs to is not quiesced.

DROP_MASTER_REPGROUP procedure

This procedure drops a replicated master group from your current site. To drop the replicated master 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 8–147 DROP_MASTER_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated master 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 master 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 8–148 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 master group.
masternotremoved	Master does not recognize the master definition site and ALL_SITES is TRUE.

DROP_MASTER_REPOBJECT procedure

This procedure drops a replicated object from a replicated master 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 8–149 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 master 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 master 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 parameter to TRUE. If the parameter is set to TRUE, the object is dropped from the database at each master site.

Exceptions

Table 8–150 DROP_MASTER_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist.
typefailure	Specified 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:** *Oracle8i Replication* for more information about conflict resolution methods.

Syntax

```
DBMS_REPCAT.DROP_PRIORITY(
      gname          IN   VARCHAR2,
      pgroup         IN   VARCHAR2,
      priority_num   IN   NUMBER);
```

Parameters

Table 8–151 DROP_PRIORITY Procedure Parameters

Parameter	Description
gname	Replicated master 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 8–152 DROP_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified replicated master group does not exist.
missingprioritygroup	Specified priority group does not exist.
notquiesced	Replicated master group is not quiesced.

DROP_PRIORITY_GROUP procedure

This procedure drops a priority group for a specified replicated master group. You must call this procedure from the master definition site.

> **See Also:** *Oracle8i Replication* for more information about conflict resolution methods.

Syntax

```
DBMS_REPCAT.DROP_PRIORITY_GROUP (
   gname     IN   VARCHAR2,
   pgroup    IN   VARCHAR2);
```

Parameters

Table 8–153 DROP_PRIORITY_GROUP Procedure Parameters

Parameter	Description
gname	Replicated master group with which the priority group is associated.
pgroup	Name of the priority group that you want to drop.

Exceptions

Table 8–154 DROP_PRIORITY_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified replicated master group does not exist.
referenced	Specified priority group is being used in conflict resolution.
notquiesced	Specified replicated master 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:** *Oracle8i Replication* for more information about conflict resolution methods.

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 8–155 DROP_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	Replicated master 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 8–156 DROP_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified replicated master group does not exist.
missingprioritygroup	Specified priority group does not exist.
paramtype, typefailure	Value has the incorrect datatype for the priority group.
notquiesced	Specified replicated master group is not quiesced

DROP_SITE_PRIORITY procedure

This procedure drops a site priority group for a specified replicated master group. You must call this procedure from the master definition site.

> **See Also:** *Oracle8i Replication* for more information about conflict resolution methods.

Syntax

```
DBMS_REPCAT.DROP_SITE_PRIORITY (
   gname       IN   VARCHAR2,
   name        IN   VARCHAR2);
```

Parameters

Table 8–157 DROP_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	Replicated master group with which the site priority group is associated.
name	Name of the site priority group that you want to drop.

Exceptions

Table 8–158 DROP_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified replicated master group does not exist.
referenced	Specified site priority group is being used in conflict resolution.
notquiesced	Specified replicated master group is not quiesced

DROP_SITE_PRIORITY_SITE procedure

This procedure drops a specified site, by name, from a site priority group. You must call this procedure from the master definition site.

> **See Also:** *Oracle8i Replication* for more information about conflict resolution methods.

Syntax

```
DBMS_REPCAT.DROP_SITE_PRIORITY_SITE (
   gname      IN   VARCHAR2,
   name       IN   VARCHAR2,
   site       IN   VARCHAR2);
```

Parameters

Table 8–159 DROP_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	Replicated master 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 8–160 DROP_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified replicated master group does not exist.
missingpriority	Specified site priority group does not exist.
notquiesced	Specified replicated master group is not quiesced.

DROP_SNAPSHOT_REPGROUP procedure

This procedure drops a snapshot site from your replicated environment. DROP_SNAPSHOT_REPGROUP automatically calls UNREGISTER_SNAPSHOT_REPGROUP to unregister the snapshot, but ignores any errors that may have occurred during unregistration.

Syntax

```
DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP (
   gname                IN   VARCHAR2,
   drop_contents        IN   BOOLEAN   := FALSE
   gowner               IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 8–161 DROP_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated master 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 master 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 master group are dropped from their schemas.
gowner	Owner of the snapshot group.

Exceptions

Table 8–162 DROP_SNAPSHOT_REPGROUP Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
missingrepgroup	Specified object group does not exist.

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 8–163 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 master 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. If the parameter is set to TRUE, the object is dropped from the database at the snapshot site.

Exceptions

Table 8–164 DROP_SNAPSHOT_REPOBJECT Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
missingobject	Specified object does not exist.
typefailure	Specified 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.

Parameters

Table 8–165 DROP_conflicttype_RESOLUTION Procedure Parameters

Parameter	Description
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 8–166 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 8–167 DROP_conflicttype_RESOLUTION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist as a table in the specified schema, or a conflict resolution routine with the specified sequence number is not registered.
notquiesced	Replicated master 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);
```

> **Note:** This procedure is overloaded. The MASTER_LIST and MASTER_TABLE parameters are mutually exclusive.

Parameters

Table 8–168 EXECUTE_DDL Procedure Parameters

Parameter	Description
gname	Name of the replicated master 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 position 1, the second at position 2, and so on.
ddl_text	The DDL that you want to have executed at each of the specified master sites. 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.

Exceptions

Table 8–169 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.

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 8–170 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, the objects are 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 *Oracle8i Replication*.
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 8–171 GENERATE_REPLICATION_SUPPORT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist as a table in the specified schema awaiting row-level replication information or as a package (body) awaiting wrapper generation.
typefailure	Specified type parameter is not supported.
notquiesced	Replicated master group has not been quiesced.
commfailure	At least one master site is not accessible.
missingschema	Schema does not exist.
dbnotcompatible	One of the masters is not 7.3.0.0 compatible.
notcompat	One of the masters is not 7.3.0.0 compatible. (Equivalent to dbnotcompatible.)
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.

> **Note:** CREATE_SNAPSHOT_REPOBJECT automatically generates snapshot support for updatable snapshots.

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 8–172 GENERATE_SNAPSHOT_SUPPORT Procedure Parameters

Parameter	Description
sname	Schema in which the object is located.
oname	The name of the object for which you are generating support.
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, the objects are 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 8–173 GENERATE_SNAPSHOT_SUPPORT Procedure Exceptions

Exceptions	Descriptions
nonsnapshot	Invocation site is not a snapshot site.
missingobject	Specified 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	Specified 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.

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:** *Oracle8i Replication* for more information about conflict resolution methods.

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 8–174 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 8–175 MAKE_COLUMN_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the masterdef site.
duplicategroup	Specified column group already exists for the table.
missingobject	Specified table does not exist.
missingcolumn	Specified column does not exist in the designated table.
duplicatecolumn	Specified column is already a member of another column group.
notquiesced	Replicated master group is not quiesced.

PURGE_MASTER_LOG procedure

This procedure removes local messages in the DBA_REPCATLOG view associated with a specified identification number, source, or replicated master group.

Syntax

```
DBMS_REPCAT.PURGE_MASTER_LOG (
   id      IN   BINARY_INTEGER,
   source  IN   VARCHAR2,
   gname   IN   VARCHAR2);
```

Parameters

Table 8–176 PURGE_MASTER_LOG Procedure Parameters

Parameter	Description
id	Identification number of the request, as it appears in the DBA_REPCATLOG view.
source	Master site from which the request originated.
gname	Name of the replicated master group for which the request was made.

Exceptions

Table 8–177 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 DBA_REPRESOLUTION_STATISTICS view.

Syntax

```
DBMS_REPCAT.PURGE_STATISTICS (
   sname      IN   VARCHAR2,
   oname      IN   VARCHAR2,
   start_date IN   DATE,
   end_date   IN   DATE);
```

Parameters

Table 8–178 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 8–179 PURGE_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Specified schema does not exist.
missingobject	Specified 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
   gowner                IN   VARCHAR2   := 'PUBLIC');
```

Parameters

Table 8–180 REFRESH_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replicated master group.
drop_missing_ contents	If an object was dropped from the replicated master 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 master group.
refresh_other_ objects	Set this to TRUE to refresh the contents of the non-snapshot objects in the replicated master group.
gowner	Owner of the snapshot group.

Exceptions

Table 8–181 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 or modifying a snapshot group in DBA_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
   gowner           IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 8–182 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_unknown (the default), reg_v7_group, reg_v8_group, and reg_repapi_group.
fname	This parameter is for internal use only. Do not set this parameter unless directed to do so by Oracle Worldwide Support.
gowner	Owner of the snapshot group.

Exceptions

Table 8–183 REGISTER_SNAPSHOT_REPGROUP Procedure Exceptions

Exception	Description
failregsnaprepgroup	Registration of snapshot group failed.
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 8–184 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 8–185 REGISTER_STATISTICS Procedure Exceptions

Exception	Description
missingschema	Specified schema does not exist.
missingobject	Specified table does not exist.

RELOCATE_MASTERDEF procedure

This procedure changes your master definition site to another master site in your replicated environment.

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 set to TRUE and INCLUDE_OLD_MASTERDEF set to TRUE.

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 8–186 RELOCATE_MASTERDEF Procedure Parameters **(Page 1 of 2)**

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. If just the master definition site fails, then you should invoke RELOCATE_MASTERDEF with NOTIFY_MASTERS set to TRUE and INCLUDE_OLD_MASTERDEF set to 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 set to FALSE.
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.

Table 8–186 RELOCATE_MASTERDEF Procedure Parameters **(Page 2 of 2)**

Parameter	Description
require_flavor_change	This parameter is for internal use only. Do not set this parameter unless directed to do so by Oracle Worldwide Support.

Exceptions

Table 8–187 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.

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);
```

Note: This procedure is overloaded. The master_list and master_table parameters are mutually exclusive.

Parameters

Table 8–188 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 can specify the database names in a PL/SQL table of type DBMS_UTILITY.DBLINK_ARRAY.

Exceptions

Table 8–189 REMOVE_MASTER_DATABASES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
nonmaster	At least one of the specified databases is not a master site.
reconfigerror	One of the specified 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 master group have the appropriate object identifiers and status values after you perform an export/import of a replicated object or an object used by Oracle replication.

Syntax

```
DBMS_REPCAT.REPCAT_IMPORT_CHECK (
   gname      IN   VARCHAR2,
   master     IN   BOOLEAN
   gowner     IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 8–190 REPCAT_IMPORT_CHECK Procedure Parameters

Parameter	Description
gname	Name of the replicated master group. If you omit both parameters, then the procedure checks all replicated master groups at your current site.
master	Set this to TRUE if you are checking a master site and FALSE if you are checking a snapshot site.
gowner	Owner of the master group.

Exceptions

Table 8–191 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 specified replicated object group does not exist.
missingschema	The specified replicated object group 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 8–192 RESUME_MASTER_ACTIVITY Procedure Parameters

Parameter	Description
gname	Name of the replicated master 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 8–193 RESUME_MASTER_ACTIVITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Replicated master group is not quiescing or quiesced.
commfailure	At least one master site is not accessible.
notallgenerated	Generate replication support before resuming replication activity.

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 8–194 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 position 1, the second at position 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.

See *Oracle8i Replication* for information about reduced data propagation before changing the default behavior of Oracle.

Exceptions

Table 8–195 SEND_OLD_VALUES Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist as a table in the specified schema awaiting row-level replication information.
missingcolumn	At least one column is not in the table.
notquiesced	Replicated master group has not been quiesced.
typefailure	An illegal operation is specified.

SET_COLUMNS procedure

This procedure lets you 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: *Oracle8i Replication* for more information.

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 8–196 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 position 1, the second at position 2, and so on.

Exceptions

Table 8–197 SET_COLUMNS Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified object does not exist as a table in the specified schema awaiting row-level replication information.
missingcolumn	At least one column is not in the table.
notquiesced	Replicated master group is not quiescing or quiesced.

SUSPEND_MASTER_ACTIVITY procedure

This procedure suspends replication activity for a master group. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
   gname    IN    VARCHAR2);
```

Parameters

Table 8–198 SUSPEND_MASTER_ACTIVITY Procedure Parameters

Parameter	Description
gname	Name of the master group for which you want to suspend activity.

Exceptions

Table 8–199 SUSPEND_MASTER_ACTIVITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notnormal	Replicated master group is not in normal operation.
commfailure	At least one master site is not accessible.

SWITCH_SNAPSHOT_MASTER procedure

This procedure changes the master database of a snapshot replicated master 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
   gowner         IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 8–200 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.
gowner	Owner of the snapshot group.

Exceptions

Table 8–201 SWITCH_SNAPSHOT_MASTER Procedure Exceptions

Exception	Description
nonsnapshot	Invocation site is not a snapshot site.
nonmaster	Specified database is not a master site.
commfailure	Specified database is not accessible.
missingrepgroup	Snapshot group does not exist.
qrytoolong	Snapshot definition query is greater 32 KB.
alreadymastered	At the local site, there is another snapshot group with the same group name mastered at the old master site.

UNREGISTER_SNAPSHOT_REPGROUP procedure

This procedure facilitates the administration of snapshots at their respective master sites by deleting a snapshot group from DBA_REGISTERED_SNAPSHOT_GROUPS.

Syntax

```
DBMS_REPCAT.UNREGISTER_SNAPSHOT_REPGROUP (
   gname     IN   VARCHAR2,
   snapsite  IN   VARCHAR2
   gowner    IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 8–202 UNREGISTER_SNAPSHOT_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the snapshot object group to be unregistered.
snapsite	Global name of the snapshot site.
gowner	Owner of the snapshot group.

VALIDATE function

This function validates the correctness of key conditions of a multiple master replication environment.

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;
```

Note: This function is overloaded. The return value of VALIDATE is the number of errors found. The function's OUT parameter returns any errors that are found. In the first interface function shown under "Syntax" above, 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 function shown under "Syntax" above is similar except that there are two OUT arrays: a VARCHAR2 array with the error messages and a NUMBER array with the error numbers.

Parameters

Table 8–203 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 master definition site only.
check_valid_objs	Check that the underlying objects for objects in the group valid. This must be done at the master definition site only. The master definition 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.
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 messages 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 8–204 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 the GLOBAL_NAMES initialization parameter 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 master definition site.
opnotsupported	Operation is not supported if the object group is replicated at a pre-version 8 node. Ensure that all nodes of the replicated master group are running version 8 of Oracle.

Usage Notes

The return value of VALIDATE is the number of errors found. The function's OUT parameter 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 8–205 WAIT_MASTER_LOG Procedure Parameters

Parameter	Description
gname	Name of the replicated master 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 8–206 WAIT_MASTER_LOG Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.

DBMS_REPCAT_ADMIN Package

Summary of Subprograms

Table 8–207 DBMS_REPCAT_ADMIN Package Subprograms

Subprogram	Description
GRANT_ADMIN_ANY_SCHEMA procedure on page 8-156	Grants the necessary privileges to the replication administrator to administer any replicated master group at the current site.
GRANT_ADMIN_SCHEMA procedure on page 8-156	Grants the necessary privileges to the replication administrator to administer a schema at the current site.
REGISTER_USER_REPGROUP procedure on page 157	Assigns proxy snapshot administrator or receiver privileges at the master site for use with remote sites.
REVOKE_ADMIN_ANY_SCHEMA procedure on page 8-159	Revokes the privileges and roles from the replication administrator that were granted by GRANT_ADMIN_ANY_SCHEMA.
REVOKE_ADMIN_SCHEMA procedure on page 8-160	Revokes the privileges and roles from the replication administrator that were granted by GRANT_ADMIN_SCHEMA.
UNREGISTER_USER_REPGROUP procedure on page 8-161	Revokes the privileges and roles from the proxy snapshot administrator or receiver that were 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 master group at the current site.

Syntax

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
   username  IN   VARCHAR2);
```

Parameters

Table 8–208 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 master groups at the current site.

Exceptions

Table 8–209 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 8–210 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 master group at the current site.

Exceptions

Table 8–211 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. It does not grant the powerful privileges granted by the GRANT_ADMIN_SCHEMA or GRANT_ADMIN_ANY_SCHEMA procedures.

> **See Also:** Appendix A, "Security Options" for more information about 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)};
```

> **Note:** This procedure is overloaded. The list_of_gnames and table_of_gnames parameters are mutually exclusive.

Parameters

Table 8–212 REGISTER_USER_REPGROUP Procedure Parameters

Parameter	Description
username	Name of the user to whom you are giving either proxy snapshot administrator or receiver privileges.
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 snapshot administration 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.
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 (the positions start at 1 and increment by 1). 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.

Exceptions

Table 8–213 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 were 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 8–214 REVOKE_ADMIN_ANY_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator whose privileges you want to revoke.

Exceptions

Table 8–215 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 were 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 8–216 REVOKE_ADMIN_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator whose privileges you want to revoke.

Exceptions

Table 8–217 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 were 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)};
```

Note: This procedure is overloaded. The list_of_gnames and table_of_gnames parameters are mutually exclusive.

Parameters

Table 8–218 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 snapshot administration 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.
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 (the positions start at 1 and increment by 1). 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.

Exceptions

Table 8–219 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.

DBMS_REPCAT_INSTANTIATE Package

Summary of Subprograms

Table 8–220 DBMS_REPCAT_INSTANTIATE Package Subprograms

Subprogram	Description
DROP_SITE_INSTANTIATION procedure on page 8-164	Public procedure that removes the target site from the DBA_REPCAT_TEMPLATE_SITES view.
INSTANTIATE_OFFLINE function on page 8-164	Public function that generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while offline.
INSTANTIATE_OFFLINE_REPAPI function on page 8-167	Public function that generates a binary file at the master site that is used to create the snapshot environment at a RepAPI remote snapshot site while offline.
INSTANTIATE_ONLINE function on page 8-170	Public function that generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while online.

For more information about the DBMS_REPCAT_INSTANTIATE package, see the following file:

ORACLE_HOME/rdbms/admin/dbmsrint.sql

DROP_SITE_INSTANTIATION procedure

This procedure drops a template instantiation at a target site. This procedure removes all related metadata at the master site and disables the specified site from refreshing its snapshots. You must execute this procedure as the user who originally instantiated the template. To see who instantiated the template, query the ALL_REPCAT_TEMPLATE_SITES view.

Syntax

```
DBMS_REPCAT_INSTANTIATE.DROP_SITE_INSTANTIATION(
     refresh_template_name  IN   VARCHAR2,
     site_name              IN   VARCHAR2);
```

Table 8–221 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, do not specify a REPAPI_SITE_ID).

INSTANTIATE_OFFLINE function

This function generates a file at the master site that is used to create the snapshot environment at the remote snapshot site while offline. This generated file is an offline instantiation file and 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 file and data into a single file that can be posted on an FTP site or loaded to a CD-ROM, floppy disk, and so on.

> **See Also:** *Oracle8i Replication* and Replication Manager online help 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 who executes this public function becomes the "registered" user of the instantiated template at the specified site.

Note: This function is used in performing an offline instantiation of a deployment template.

This function 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',
     use_default_gowner      IN   BOOLEAN   := TRUE)
     return NUMBER;
```

Table 8–222 INSTANTIATE_OFFLINE Function Parameters

Parameter	Description
refresh_template_name	The name of the deployment template to be instantiated.
site_name	The 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, specify the ID used when creating the runtime parameters (the ID was retrieved by using the GET_RUNTIME_PARM_ID function).
next_date	The next refresh date value to be used when creating the refresh group.
interval	The refresh interval to be used when creating the refresh group.
use_default_gowner	If TRUE, then any snapshot object groups created are owned by the default user PUBLIC. If FALSE, then any snapshot object groups created are owned by the user performing the instantiation.

Table 8–223 INSTANTIATE_OFFLINE Function Exceptions

Exception	Description
miss_refresh_ template	The deployment template name specified is invalid or does not exist.
dupl_template_site	The deployment template has already been instantiated at the snapshot site. A deployment template can be instantiated only once at a particular snapshot site.
not_authorized	The user attempting to instantiate the deployment template is not authorized to do so.

Returns

Table 8–224 INSTANTIATE_OFFLINE Function Returns

Return Value	Description
<system-generated number>	Specifies 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_OFFLINE_REPAPI function

This function generates a file at the master site that is used to create the snapshot environment at a remote RepAPI snapshot site while offline. This offline instantiation file should be used at remote RepAPI 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 running Oracle8*i* Lite (which includes RepAPI). The generated file can be posted on an FTP site or loaded to a CD-ROM, floppy disk, and so on.

The file generated by this function is stored at the master site in the directory specified by the parameter `OFFLINE_DIRPATH`. The file is named based on the `USER_NAME`, `REFRESH_TEMPLATE_NAME`, and `SITE_ID` and is identified with the file type extension `.oli`. For example, an offline instantiation for the user SCOTT of the template named MYTEMPLATE at site 1234 is named the following:

```
scott_mytemplate_1234.oli.
```

This is a public function to generate an offline instantiation file for the connected user.

Note: This function is used in performing an offline instantiation of a deployment template.

This function 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_REPAPI(
  refresh_template_name   IN   VARCHAR2,
  site_id                 IN   VARCHAR2   := NULL,
  master                  IN   VARCHAR2   := NULL,
  url                     IN   VARCHAR2   := NULL,
  ssl                     IN   NUMBER     := 0,
  trace_vector            IN   NUMBER     := DBMS_REPCAT_RGT.NO_TRACE_DUMP,
  resultset_threshold     IN   NUMBER     := DBMS_REPCAT_INSTANTIATE.
                                             RESULTSET_THRESHOLD,
  lob_threshold           IN   NUMBER     := DBMS_REPCAT_INSTANTIATE.
                                             LOB_THRESHOLD);
```

Table 8–225 INSTANTIATE_OFFLINE_REPAPI Function Parameters **(Page 1 of 2)**

Parameter	Description
refresh_template_name	The name of the deployment template to be instantiated.
site_id	A temporary name assigned to this offline instantiation. This temporary name is part of the offline instantiation file name. If the default NULL value is used, Oracle generates a number for this temporary name. It may be useful to specify a name if you want the offline instantiation file name to indicate its intended snapshot site. The site_id must be unique. That is, no two offline instantiation files can have the same site_id. This temporary name is always overwritten by the RepAPI client when the deployment template is instantiated. Oracle Corporation recommends that you use the default NULL value for this parameter.
master	An optional alias used for the server by the RepAPI client. If specified, then the RepAPI client must always refer to the server by this alias.
url	The published URL at the master site for access to the database. If specified, then the RepAPI client must always refer to the server by this URL.
ssl	1 indicates that the snapshots use secure sockets layer (SSL) to communicate with the master site. 0 indicates that SSL is not used.
trace_vector	The trace level for debugging.
resultset_threshold	The maximum size of non-LOB row data sent during the snapshot refresh process.

Table 8–225 INSTANTIATE_OFFLINE_REPAPI Function Parameters **(Page 2 of 2)**

Parameter	Description
lob_threshold	The maximum size of LOB row data sent during the snapshot refresh process.

Table 8–226 INSTANTIATE_OFFLINE_REPAPI Function Exceptions

Exception	Description
miss_refresh_template	The template does not exist.
miss_user	The username does not exist in the database.
miss_template_site	The template has not been instantiated for the user and site.

Returns

Table 8–227 INSTANTIATE_OFFLINE_REPAPI Function Returns

Return Value	Description
0	An error was encountered.
1	No errors were encountered.

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).

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 who executes this public function becomes 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',
     use_default_gowner      IN   BOOLEAN   := TRUE)
     return NUMBER;
```

Table 8–228 INSTANTIATE_ONLINE Function Parameters

Parameter	Description
refresh_template_name	The name of the deployment template to be instantiated.
site_name	The 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, 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.
use_default_gowner	If TRUE, then any snapshot object groups created are owned by the default user PUBLIC. If FALSE, then any snapshot object groups created are owned by the user performing the instantiation.

Table 8–229 INSTANTIATE_ONLINE Function Exceptions

Exception	Description
miss_refresh_ template	The deployment template name specified is invalid or does not exist.
dupl_template_site	The deployment template has already been instantiated at the snapshot site. A deployment template can be instantiated only once at a particular snapshot site.
not_authorized	The user attempting to instantiate the deployment template is not authorized to do so.

Returns

Table 8–230 INSTANTIATE_ONLINE Function Returns

Return Value	Description
<*system-generated number*>	Specifies the generated system number for the output_id when you select from the USER_REPCAT_TEMP_OUTPUT view to retrieve the generated instantiation script.

DBMS_REPCAT_RGT Package

Summary of Subprograms

Table 8–231 DBMS_REPCAT_RGT Package Subprograms **(Page 1 of 3)**

Subprogram	Description
ALTER_REFRESH_TEMPLATE procedure on page 8-174	Allows the DBA to alter existing deployment templates.
ALTER_TEMPLATE_OBJECT procedure on page 8-176	Alters objects that have been added to a specified deployment template.
ALTER_TEMPLATE_PARM procedure on page 8-179	Allows the DBA to alter the parameters for a specific deployment template.
ALTER_USER_AUTHORIZATION procedure on page 8-181	Alters the contents of the DBA_REPCAT_USER_AUTHORIZATIONS view.
ALTER_USER_PARM_VALUE procedure on page 8-182	Changes existing parameter values that have been defined for a specific user.
COMPARE_TEMPLATES function on page 8-185	Allows the DBA to compare the contents of two deployment templates.
COPY_TEMPLATE function on page 8-186	Allows the DBA to copy a deployment template.
CREATE_OBJECT_FROM_EXISTING function on page 8-188	Creates a template object definition from existing database objects and adds it to a target deployment template.
CREATE_REFRESH_TEMPLATE function on page 8-190	Creates the deployment template, which allows the DBA to define the template name, private/public status, and target refresh group.
CREATE_TEMPLATE_OBJECT function on page 8-192	Adds object definitions to a target deployment template container.
CREATE_TEMPLATE_PARM function on page 8-196	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 8-198	Authorizes specific users to instantiate private deployment templates.
CREATE_USER_PARM_VALUE function on page 8-199	Predefines deployment template parameter values for specific users.

Table 8–231 DBMS_REPCAT_RGT Package Subprograms **(Page 2 of 3)**

Subprogram	Description
DELETE_RUNTIME_PARMS procedure on page 8-202	Deletes a runtime parameter value that you defined using the INSERT_RUNTIME_PARMS procedure.
DROP_ALL_OBJECTS procedure on page 8-203	Allows the DBA to drop all objects or specific object types from a deployment template.
DROP_ALL_TEMPLATE_PARMS procedure on page 8-204	Allows the DBA to drop template parameters for a specified deployment template.
DROP_ALL_TEMPLATE_SITES procedure on page 8-205	Removes all entries from the DBA_REPCAT_TEMPLATE_SITES view.
DROP_ALL_TEMPLATES procedure on page 8-206	Removes all deployment templates at the site where the procedure is called.
DROP_ALL_USER_AUTHORIZATIONS procedure on page 8-206	Allows the DBA to drop all user authorizations for a specified deployment template.
DROP_ALL_USER_PARM_VALUES procedure on page 8-207	Drops user parameter values for a specific deployment template.
DROP_REFRESH_TEMPLATE procedure on page 8-209	Drops a deployment template.
DROP_SITE_INSTANTIATION procedure on page 8-210	Removes the target site from the DBA_REPCAT_TEMPLATE_SITES view.
DROP_TEMPLATE_OBJECT procedure on page 8-211	Removes a template object from a specific deployment template.
DROP_TEMPLATE_PARM procedure on page 8-213	Removes an existing template parameter from the DBA_REPCAT_TEMPLATE_PARMS view.
DROP_USER_AUTHORIZATION procedure on page 8-214	Removes a user authorization entry from the DBA_REPCAT_USER_AUTHORIZATIONS view.
DROP_USER_PARM_VALUE procedure on page 8-215	Removes a predefined user parameter value for a specific deployment template.
GET_RUNTIME_PARM_ID function on page 8-216	Retrieves an ID to be used when defining a runtime parameter value.
INSERT_RUNTIME_PARMS procedure on page 8-217	Defines runtime parameter values prior to instantiating a template.

Table 8–231 DBMS_REPCAT_RGT Package Subprograms **(Page 3 of 3)**

Subprogram	Description
INSTANTIATE_OFFLINE function on page 8-219	Generates a script at the master site that is used to create the snapshot environment at the remote snapshot site while offline.
INSTANTIATE_OFFLINE_REPAPI function on page 8-221	Generates a binary file at the master site that is used to create the snapshot environment at a RepAPI remote snapshot site while offline.
INSTANTIATE_ONLINE function on page 8-224	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 227	Prevents users from reading or instantiating the template when a deployment template is being updated or modified.
LOCK_TEMPLATE_SHARED procedure on page 8-227	Makes a specified deployment template read-only.

ALTER_REFRESH_TEMPLATE procedure

This procedure allows the DBA to 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 8–232 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 to which the template objects will be added. 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 8–233 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 are altering the object DDL and assigning the object to a different deployment template.

Changes made to the template are reflected only at new sites instantiating the deployment template. Remote sites that have already instantiated the template must reinstantiate 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 8–234 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 to which you want to reassign this object. 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	This parameter is for internal use only. Do not set this parameter unless directed to do so by Oracle Worldwide Support.

Exceptions

Table 8–235 ALTER_TEMPLATE_OBJECT Procedure Exceptions

Exception	Description
miss_refresh_ template	Deployment template name specified is invalid or does not exist.
miss_flavor_id	If you receive this exception, contact Oracle Worldwide Support.
bad_object_type	Object type is specified incorrectly. See Table 8–234 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 must use 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 include renaming the parameter and 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 8–236 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 reassigned 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 whether 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 prevent an override.

Exceptions

Table 8–237 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 must use 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_USER_AUTHORIZATIONS view. Specifically, you can change user / deployment template authorization assignments. This procedure is helpful, for example, if an employee is reassigned 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 8–238 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 8–239 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_USER_ AUTHORIZATIONS 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_USER_AUTHORIZATIONS 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. Change a user parameter value to quickly and securely change the data set of a remote snapshot site.

> **See Also:** "Deployment Template Design" in *Oracle8i Replication* 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 8–240 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 reassigned 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 8–241 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 must use 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, 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 8–242 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 8–243 COMPARE_TEMPLATES Function Exceptions

Exception	Description
miss_refresh_template	The deployment template name to be compared is invalid or does not exist.

Returns

Table 8–244 COMPARE_TEMPLATES Function Returns

Return Value	Description
<system-generated number>	Specifies 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 uses many of the objects 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 8–245 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 8–246 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 8–247 COPY_TEMPLATES Function Returns

Return Value	Description
<system-generated number>	System-generated number 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 8–248 CREATE_OBJECT_FROM_EXISTING Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to which you want to add this object.
object_name	Optionally, 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 (that is, PROCEDURE, TRIGGER, and so on). 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 8–249 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_TEMPLATES view for a list of existing deployment templates.
bad_object_type	The object type is specified incorrectly.
dupl_template_object	An object of the same name and type has already been added to the specified deployment template.
objectmissing	The object specified does not exist.

Returns

Table 8–250 CREATE_OBJECT_FROM_EXISTING Function Returns

Return Value	Description
<system-generated number>	System-generated number 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 8–251 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 parameter 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 8–252 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 8–253 CREATE_REFRESH_TEMPLATE Function Returns

Return Value	Description
<system-generated number>	System-generated number 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 8–254 CREATE_TEMPLATE_OBJECT Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to which you want to add this object.
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 (that is, SNAPSHOT, TRIGGER, PROCEDURE, and so on). Objects of the following type may be specified: SNAPSHOT PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM MATERIALIZED VIEW SEQUENCE DATABASE LINK TRIGGER
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. You can use a colon (:) to create a template parameter for your template object. See Chapter 4, "Create Deployment Template" for more information. When you add a snapshot with a CREATE SNAPSHOT statement, make sure you specify the schema name of the owner of the master table in the snapshot query.
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	This parameter is for internal use only. Do not set this parameter unless directed to do so by Oracle Worldwide Support.

Exceptions

Table 8–255 CREATE_TEMPLATE_OBJECT Function Exceptions

Exception	Description
miss_refresh_ template	Specified refresh template name is invalid or missing. Query the DBA_REPCAT_REFRESH_TEMPLATES view for a list of existing deployment templates.
bad_object_type	Object type is specified incorrectly. See Table 8–254 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 8–256 CREATE_TEMPLATE_OBJECT Function Returns

Return Value	Description
<system-generated number>	System-generated number used internally by Oracle.

Usage Notes

Because CREATE_TEMPLATE_OBJECT utilizes a CLOB, you must use 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 8-179 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 8–257 CREATE_TEMPLATE_PARM Function Parameters

Parameter	Description
refresh_template_name	Name of the deployment template for which you want to create the parameter.
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 whether 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 8–258 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 8–259 CREATE_TEMPLATE_PARM Function Returns

Return Value	Description
<system-generated number>	System-generated number used internally by Oracle.

Usage Notes

Because the CREATE_TEMPLATE_PARM function utilizes a CLOB, you must use 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_USER_AUTHORIZATIONS 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 8–260 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 (for example, 'john, mike, bob')
refresh_template_name	Name of the template that you want to authorize the specified user to instantiate.

Exceptions

Table 8–261 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_USER_AUTHORIZATIONS view for a listing of template authorizations.

Returns

Table 8–262 CREATE_USER_AUTHORIZATION Function Returns

Return Value	Description
<system-generated number>	System-generated number used internally by Oracle.

CREATE_USER_PARM_VALUE function

This function predefines deployment template parameter values for specific users. For example, if you want to predefine 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 8–263 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 parameter 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 predefining a user parameter value for.
`parm_value`	The predefined parameter value that will be used during the instantiation process initiated by the specified user.

Exceptions

Table 8–264 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_PARM_VALUES 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 8–265 CREATE_USER_PARM_VALUE Function Returns

Return Value	Description
<*system-generated number*>	System-generated number used internally by Oracle.

Usage Notes

Because the CREATE_USER_PARM_VALUE function utilizes a CLOB, you must use 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 8–266 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 8–267 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 8–268 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 8–269 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 8–268 for a list of valid object types.

DROP_ALL_TEMPLATE_PARMS procedure

This procedure lets you drop template parameters for a specified deployment template. You can use this procedure to drop all parameters that are not referenced by a template object or to drop from the template all objects that reference any parameter, along with all of 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 8–270 DROP_ALL_TEMPLATE_PARMS Procedure Parameters

Parameter	Description
refresh_template_ name	Name of the deployment template that contains the parameters and objects that you want to drop.
drop_objects	If no value is specified, then this parameter defaults to N, which drops all parameters not referenced by a template object. If Y is specified, then all objects that reference any template parameter and the template parameters themselves are dropped. The objects are dropped from the template, not from the database.

Exceptions

Table 8–271 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, Oracle8*i* 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 8–272 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 8–273 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_USER_AUTHORIZATIONS 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 8–274 DROP_ALL_USER_AUTHORIZATIONS Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the user authorizations that you want to drop.

Exceptions

Table 8–275 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 has the effect described:

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 8–276 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 8–277 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_VALUES 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 8–278 DROP_REFRESH_TEMPLATE Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be dropped.

Exceptions

Table 8–279 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_TEMPLATES view for a list of deployment templates.

DROP_SITE_INSTANTIATION procedure

This procedure drops a template instantiation at any target site, including RepAPI sites. This procedure removes all related metadata at the master site and disables the specified site from refreshing its 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,}
     process_repapi_site    IN   VARCHAR2 := 'N');
```

Note: This procedure is overloaded. The `site_name` and `repapi_site_id` parameters are mutually exclusive.

Table 8–280 DROP_SITE_INSTANTIATION Procedure Parameters

Parameter	Description
refresh_template_name	The name of the deployment template to be dropped.
user_name	The name of the user who originally instantiated the template at the remote snapshot site. Query the ALL_REPCAT_TEMPLATE_SITES view to see the users that instantiated templates. See the "ALL_REPCAT_TEMPLATE_SITES" section on page 9-13 for more information.
site_name	Identifies the Oracle server site where you want to drop the specified template instantiation. If you specify a SITE_NAME, 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, do not specify a SITE_NAME.
process_repapi_site	If set to 'Y' then the SITE_NAME is assumed to be a RepAPI SITE_NAME. The default value is 'N'. This parameter has no relevance if REPAPI_SITE_ID is non-NULL.

Exceptions

Table 8–281 DROP_SITE_INSTANTIATION Procedure Exceptions

Exception	Description
miss_refresh_ template	The deployment template name specified is invalid or does not exist.
miss_user	The username specified does not exist.
miss_template_site	The deployment template has not been instantiated for user and site.

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 must reinstantiate 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 8–282 DROP_TEMPLATE_OBJECT Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template from which you are dropping the object.
object_name	Name of the template object to be dropped.
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 8–283 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_OBJECTS 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 8–284 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 8–285 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_USER_AUTHORIZATIONS 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 8-206 for more information.

Syntax

```
DBMS_REPCAT_RGT.DROP_USER_AUTHORIZATION (
   refresh_template_name   IN   VARCHAR2,
   user_name               IN   VARCHAR2);
```

Parameters

Table 8–286 DROP_USER_AUTHORIZATION Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template from which the user's authorization is being removed.
user_name	Name of the user whose authorization is being removed.

Exceptions

Table 8–287 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_USER_AUTHORIZATIONS 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 predefined 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 8–288 DROP_USER_PARM_VALUE Procedure Parameters

Parameter	Description
refresh_template_name	Deployment template name that contains the parameter value that you want to drop.
parameter_name	Parameter name that contains the predefined value that you want to drop.
user_name	Name of the user whose parameter value you want to drop.

Exceptions

Table 8–289 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_VALUES 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 8–290 GET_RUNTIME_PARM_ID Function Returns

Return Value	Corresponding Datatype
<*system-generated number*>	Runtime parameter values are assigned to the system-generated number and are 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 templates.

Syntax

```
DBMS_REPCAT_RGT.INSERT_RUNTIME_PARMS (
   runtime_parm_id    IN   NUMBER,
   parameter_name     IN   VARCHAR2,
   parameter_value    IN   CLOB);
```

Parameters

Table 8–291 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 for which you are defining a runtime parameter value. 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 8–292 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.
miss_user_parm_ values	The deployment template, user, and parameter combination does not exist in the DBA_REPCAT_USER_PARM_VALUES view.

Usage Notes

Because the this procedure utilizes a CLOB, you must use 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 not 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 function must 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 function is used in performing an offline instantiation of a deployment template. Additionally, this function is for replication administrators who are instantiating for another user. Users wanting to perform their own instantiation should use the public version of the INSTANTIATE_OFFLINE function. See the INSTANTIATE_OFFLINE function on page 8-164 for more information.

This function 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',
     use_default_gowner       IN   BOOLEAN   := TRUE)
     return NUMBER;
```

Parameters

Table 8–293 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 who 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.
use_default_gowner	If TRUE, then any snapshot object groups created are owned by the default user PUBLIC. If FALSE, then any snapshot object groups created are owned by the user performing the instantiation.

Exceptions

Table 8–294 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_USER_AUTHORIZATIONS view. If user is not listed, then the specified user is not authorized to instantiate the target deployment template.

Returns

Table 8–295 INSTANTIATE_OFFLINE Function Returns

Return Value	Description
<system-generated number>	Specifies 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_OFFLINE_REPAPI function

This function generates a file at the master site that is used to create the snapshot environment at a remote RepAPI snapshot site while offline. This offline instantiation file should be used at remote RepAPI 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 running Oracle8*i* Lite (which includes RepAPI). The generated file can be posted on an FTP site or loaded to a CD-ROM, floppy disk, and so on.

The file generated by this function is stored at the master site in the directory specified by the parameter `OFFLINE_DIRPATH`. The file is named based on the `USER_NAME`, `REFRESH_TEMPLATE_NAME`, and `SITE_ID` and is identified with the file type extension `.oli`. For example, an offline instantiation for the user SCOTT of the template named MYTEMPLATE at site 1234 is named the following:

`scott_mytemplate_1234.oli`.

Note: This function is used in performing an offline instantiation of a deployment template. Additionally, this function 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_REPAPI function. See the INSTANTIATE_OFFLINE_REPAPI function on page 8-167 for information.

This function 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_REPAPI(
  refresh_template_name   IN   VARCHAR2,
  site_id                 IN   VARCHAR2,
  user_name               IN   VARCHAR2   := USER,
  master                  IN   VARCHAR2   := NULL,
  url                     IN   VARCHAR2   := NULL,
  ssl                     IN   NUMBER     := 0,
  trace_vector            IN   NUMBER     := DBMS_REPCAT_RGT.NO_TRACE_DUMP,
  resultset_threshold     IN   NUMBER     := DBMS_REPCAT_INSTANTIATE.
                                             RESULTSET_THRESHOLD,
  lob_threshold           IN   NUMBER     := DBMS_REPCAT_INSTANTIATE.
                                             LOB_THRESHOLD);
```

Table 8–296 INSTANTIATE_OFFLINE_REPAPI Function Parameters

Parameter	Description
refresh_template_name	The name of the deployment template to be instantiated.
site_id	The identification number of the remote site that is instantiating the deployment template. This number is the site identifier for the snapshot site. Because the value provided for this parameter is usually temporary, it may be updated by the RepAPI client in subsequent operations.
user_name	The name of the user for whom the instantiation file is being generated.
master	An optional alias used for the server by the RepAPI client. If specified, then the RepAPI client must always refer to the server by this alias.
url	The published URL at the master site for access to the database. If specified, then the RepAPI client must always refer to the server by this URL.
ssl	1 indicates that the snapshots use secure sockets layer (SSL) to communicate with the master site. 0 indicates that SSL is not used.
trace_vector	The trace level for debugging.
resultset_threshold	The maximum size of non-LOB row data sent during the snapshot refresh process.
lob_threshold	The maximum size of LOB row data sent during the snapshot refresh process.

Table 8–297 INSTANTIATE_OFFLINE_REPAPI Function Exceptions

Exception	Description
miss_refresh_template	The template does not exist.
miss_user	The username does not exist in the database.
miss_template_site	The template has not been instantiated for the user and site.

Returns

Table 8–298 INSTANTIATE_OFFLINE_REPAPI Function Returns

Return Value	Description
0	An error was encountered.
1	No errors were encountered.

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 function must 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 function is for replication administrators who are instantiating for another user. Users wanting to perform their own instantiation should use the public version of the INSTANTIATE_ONLINE function, described in the "INSTANTIATE_ONLINE function" section on page 8-170.

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',
   use_default_gowner      IN   BOOLEAN   := TRUE)
   return NUMBER;
```

Parameters

Table 8–299 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 who 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.
use_default_gowner	If TRUE, then any snapshot object groups created are owned by the default user PUBLIC. If FALSE, then any snapshot object groups created are owned by the user performing the instantiation.

Exceptions

Table 8–300 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_USER_ AUTHORIZATIONS view. If user is not listed, then the specified user is not authorized to instantiate the target deployment template.
bad_parms	Not all of the template parameters were populated by the defined user parameter values and / or template default values. The number of predefined values may not have matched the number of template parameters or a predefined value was invalid for the target parameter (that is, type mismatch).

Returns

Table 8–301 INSTANTIATE_ONLINE Function Returns

Return Value	Description
<system-generated number>	Specifies the system-generated 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 is 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

The LOCK_TEMPLATE_SHARED procedure is used to make a specified deployment template "read-only." This procedure should be called before instantiating a template, as this ensures that nobody can change the deployment template while it is being instantiated.

The lock is released when a `ROLLBACK` or `COMMIT` is performed.

Syntax

```
DBMS_REPCAT_RGT.LOCK_TEMPLATE_SHARED();
```

Parameters

None

DBMS_REPUTIL Package

Summary of Subprograms

Table 8–302 DBMS_REPUTIL Package Subprograms

Subprogram	Description
REPLICATION_OFF procedure on page 8-229	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 8-229	Re-enables replication of changes after replication has been temporarily suspended.
REPLICATION_IS_ON function on page 8-230	Determines whether or not replication is running.
FROM_REMOTE function on page 8-230	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 8-231	Determines the global database name of the local database (the global name is the returned value).
MAKE_INTERNAL_PKG procedure on page 8-231	Synchronizes internal packages and tables in the replication catalog. This procedure is executed under the direction of Oracle Worldwide Support only.
SYNC_UP_REP procedure on page 8-232	Synchronizes internal triggers and tables/snapshots in the replication catalog. This procedure is executed under the direction of Oracle Worldwide Support only.

REPLICATION_OFF procedure

This procedure lets you modify 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. A return value of FALSE indicates that replication is disabled at the current site for the replicated master 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

MAKE_INTERNAL_PKG procedure

This procedure synchronizes the existence of an internal package with a table in the replication catalog. If the table has replication support, execute this procedure to create the internal package. If replication support does not exist, destroy any related internal package.

> **Caution:** This procedure should only be executed under the guidance of Oracle Worldwide Support.

Syntax

```
DBMS_REPUTIL.MAKE_INTERNAL_PKG (
   canon_sname    IN   VARCHAR2,
   canon_oname    IN   VARCHAR2);
```

Parameters

Table 8–303 MAKE_INTERNAL_PKG Procedure Parameters

Parameter	Description
canon_sname	Schema containing the table to be synchronized. This parameter value must be canonically defined (capitalization must match object and must not be enclosed in double quotes).
canon_oname	Name of the table to be synchronized. This parameter value must be canonically defined (capitalization must match object and must not be enclosed in double quotes).

SYNC_UP_REP procedure

This procedure synchronizes the existence of an internal trigger with a table or snapshot in the replication catalog. If the table or snapshot has replication support, execute this procedure to create the internal replication trigger. If replication support does not exist, destroy any related internal trigger.

Caution: This procedure should only be executed under the guidance of Oracle Worldwide Support.

Syntax

```
DBMS_REPUTIL.SYNC_UP_REP (
   canon_sname     IN   VARCHAR2,
   canon_oname     IN   VARCHAR2);
```

Parameters

Table 8–304 SYNC_UP_REP Procedure Parameters

Parameter	Description
canon_sname	Schema containing the table or snapshot to be synchronized. This parameter value must be canonically defined (capitalization must match object and must not be enclosed in double quotes).
canon_oname	Name of the table or snapshot to be synchronized. This parameter value must be canonically defined (capitalization must match object and must not be enclosed in double quotes).

DBMS_SNAPSHOT Package

Summary of Subprograms

Table 8–305 DBMS_SNAPSHOT Package Subprograms

Subprogram	Description
BEGIN_TABLE_REORGANIZATION procedure on page 8-234	Performs a process to preserve snapshot data needed for refresh.
END_TABLE_REORGANIZATION procedure procedure on page 8-234	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 8-235	Returns the value of the I_AM_REFRESH package state.
PURGE_DIRECT_LOAD_LOG procedure on page 8-235	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 8-236	Purges rows from the snapshot log.
PURGE_SNAPSHOT_FROM_LOG procedure on page 8-237	Purges rows from the snapshot log.
REFRESH procedure on page 8-239	Consistently refreshes one or more snapshots that are not members of the same refresh group.
REFRESH_ALL_MVIEWS procedure on page 8-242	Refreshes all snapshots that do not reflect changes to their master table.
REFRESH_DEPENDENT procedure on page 8-243	Refreshes all table-based snapshots that depend on a specified master table or list of master tables.
REGISTER_SNAPSHOT procedure on page 8-245	Enables the administration of individual snapshots.
UNREGISTER_SNAPSHOT procedure on page 8-247	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:** "Reorganizing Master Tables that Have Snapshot Logs" on page 7-20 for more information.

Syntax

```
DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION (
   tabowner     IN   VARCHAR2,
   tabname      IN   VARCHAR2);
```

Parameters

Table 8–306 BEGIN_TABLE_REORGANIZATION Procedure Parameters

Parameter	Description
tabowner	Owner of the table being reorganized.
tabname	Name of the table being reorganized.

END_TABLE_REORGANIZATION procedure

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:** "Reorganizing Master Tables that Have Snapshot Logs" on page 7-20 for more information.

Syntax

```
DBMS_SNAPSHOT.END_TABLE_REORGANIZATION (
   tabowner     IN   VARCHAR2,
   tabname      IN   VARCHAR2);
```

Parameters

Table 8–307 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. 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.

Syntax

```
DBMS_SNAPSHOT.I_AM_A_REFRESH()
  RETURN BOOLEAN;
```

Parameters

None

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 usually is used in environments using Oracle's data warehousing technology.

See Also: *Oracle8i Data Warehousing Guide* for more information.

Syntax

```
DBMS_SNAPSHOT.PURGE_DIRECT_LOAD_LOG();
```

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 8–308 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);
```

Note: This procedure is overloaded. The `snapshot_id` parameter is mutually exclusive with the three remaining parameters: `snapowner`, `snapname`, and `snapsite`.

Parameters

Table 8–309 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, 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).
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, 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);
```

Note: This procedure is overloaded. The `list` and `tab` parameters are mutually exclusive.

Parameters

Table 8–310 REFRESH Procedure Parameters **(Page 1 of 2)**

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.
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.
refresh_after_errors	If this parameter is TRUE, an updatable snapshot continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master table. If this parameter is TRUE and atomic_refresh is FALSE, this procedure continues to refresh other snapshots if it fails while refreshing a snapshot.

Table 8–310 REFRESH Procedure Parameters **(Page 2 of 2)**

Parameter	Description
purge_option	If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), 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, set this parameter to 0 and occasionally execute PUSH with this parameter set to 2 to reduce the queue.
parallelism	0 means serial propagation, *n* > 1 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 parameter is set to TRUE, then the list of snapshots is refreshed in a single transaction. All of the refreshed snapshots are updated to a single point in time. If the refresh fails for any of the snapshots, none of the snapshots are updated. If this parameter is set to FALSE, then each of the snapshots is 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 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 master table on which it depends.
- The snapshot and all of the master tables on which it depends are local.
- The snapshot is in the view DBA_MVIEWS.

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 8–311 REFRESH_ALL_MVIEWS Procedure Parameters **(Page 1 of 2)**

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, 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 parameter is TRUE, an updatable snapshot continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master table. If this parameter is TRUE and atomic_refresh is FALSE, this procedure continues to refresh other snapshots if it fails while refreshing a snapshot.

Table 8–311 REFRESH_ALL_MVIEWS Procedure Parameters **(Page 2 of 2)**

Parameter	Description
atomic_refresh	If this parameter is set to TRUE, then the refreshed snapshots are refreshed in a single transaction. All of the refreshed snapshots are updated to a single point in time. If the refresh fails for any of the snapshots, none of the snapshots are updated. If this parameter is set to FALSE, then each of the refreshed snapshots is refreshed in a separate transaction. The number of job queue processes must be set to 1 or greater if this parameter is FALSE.

REFRESH_DEPENDENT procedure

This procedure refreshes all snapshots (materialized views) with the following properties:

- The snapshot depends on a master table in the list of specified master tables.
- The snapshot has not been refreshed since the most recent change to a master table on which it depends.
- The snapshot and all of the master tables on which it depends are local.
- The snapshot is in the view DBA_MVIEWS.

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);
```

Note: This procedure is overloaded. The list and tab parameters are mutually exclusive.

Parameters

Table 8–312 REFRESH_DEPENDENT Procedure Parameters **(Page 1 of 2)**

Parameter	Description
number_of_failures	Returns the number of failures that occurred during processing.
list | tab	Comma-separated list of master 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 parameter is TRUE, an updatable snapshot continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master table. If this parameter is TRUE and atomic_refresh is FALSE, this procedure continues to refresh other snapshots if it fails while refreshing a snapshot.

Table 8–312 REFRESH_DEPENDENT Procedure Parameters **(Page 2 of 2)**

Parameter	Description
atomic_refresh	If this parameter is set to TRUE, then the refreshed snapshots are refreshed in a single transaction. All of the refreshed snapshots are updated to a single point in time. If the refresh fails for any of the snapshots, none of the snapshots are updated. If this parameter is set to FALSE, then each of the refreshed snapshots is 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 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 8–313 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 name should not contain any double quotes.
snapshot_id	The identification number of the snapshot. Specify a version 8 snapshot as a BINARY_INTEGER. Specify a version 7 snapshot registering at an version 8 or greater master sites as a DATE.
flag	PL/SQL package variable indicating whether subsequent CREATE or MOVE statements are registered in the query text.
query_txt	The first 32,000 bytes of the snapshot definition query.
rep_type	Version of the snapshot. Valid constants that can be assigned include reg_unknown (the default), reg_v7_group, reg_v8_group, and reg_repapi_group.

Usage Notes

This procedure is invoked at the master site by a remote snapshot site using 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. It is 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 8–314 UNREGISTER_SNAPSHOT Procedure Parameters

Parameters	Description
snapowner	Owner of the snapshot.
snapname	Name of the snapshot.
snapsite	Name of the snapshot site.

9

Data Dictionary Views

This chapter describes data dictionary views that can be useful to users of the advanced replication facility. The views are alphabetized within the following categories:

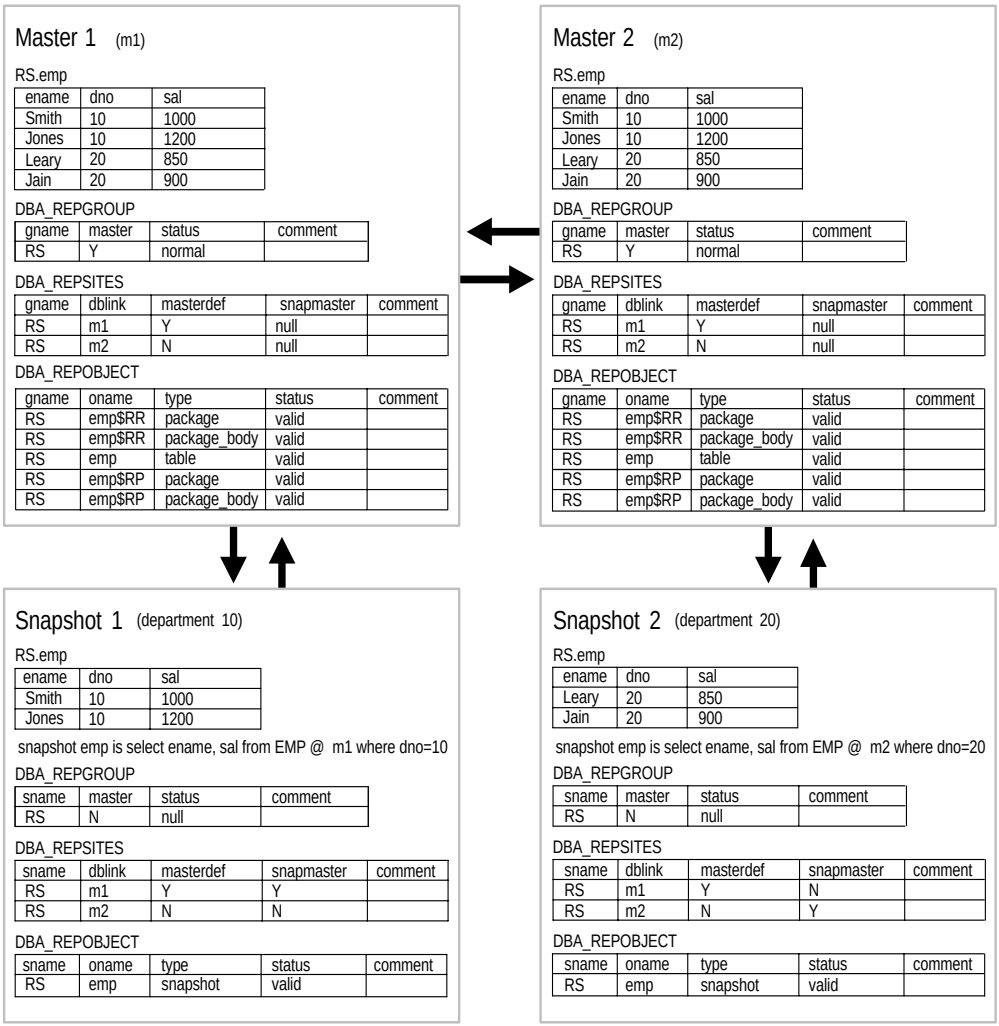
- Replication Catalog Views
- Deferred Transaction Views
- Snapshots and Snapshot Refresh Group Views

Replication Catalog Views

When you install replication capabilities at a site, Oracle installs the replication catalog, which consists of tables and views, at that site. As shown in Figure 9–1, the views are used by master and snapshot sites to determine such information as what objects are being replicated, where they are being replicated, and if any errors have occurred during replication.

Caution: Do not modify the replication catalog tables directly. Instead, use the procedures provided in the DBMS_REPCAT package.

Figure 9–1 Replication Catalog Views



Many data dictionary tables have three corresponding views:

- An ALL_ view displays all the information accessible to the current user, including information from the current user's schema as well as information from objects in other schemas, if the current user has access to those objects by way of grants of privileges or roles.
- A DBA_ view displays all relevant information in the entire database. DBA_ views are intended only for administrators. They can be accessed only by users with the SELECT_ANY_TABLE privilege. This privilege is assigned to the DBA role when Oracle is initially installed.
- A USER_ view displays all the information from the schema of the current user. No special privileges are required to query these views.

The columns of the ALL_, DBA_, and USER_ views corresponding to a single data dictionary table are usually nearly identical. Therefore, these views are described in full only once in this chapter (for the ALL_ view). The views are listed without the full description for DBA_ and USER_ views, but differences are noted.

This section contains information about the replication catalog views listed in Table 9–1.

Table 9–1 Replication Catalog Views

ALL_ Views	DBA_ Views	USER_ Views
ALL_REPCATLOG	DBA_REPCATLOG	USER_REPCATLOG
ALL_REPCAT_REFRESH_TEMPLATES	DBA_REPCAT_REFRESH_TEMPLATES	USER_REPCAT_REFRESH_TEMPLATES
ALL_REPCAT_TEMPLATE_OBJECTS	DBA_REPCAT_TEMPLATE_OBJECTS	USER_REPCAT_TEMPLATE_OBJECTS
ALL_REPCAT_TEMPLATE_PARMS	DBA_REPCAT_TEMPLATE_PARMS	USER_REPCAT_TEMPLATE_PARMS
ALL_REPCAT_TEMPLATE_SITES	DBA_REPCAT_TEMPLATE_SITES	USER_REPCAT_TEMPLATE_SITES
ALL_REPCAT_USER_AUTHORIZATIONS	DBA_REPCAT_USER_AUTHORIZATIONS	USER_REPCAT_USER_AUTHORIZATIONS
ALL_REPCAT_USER_PARM_VALUES	DBA_REPCAT_USER_PARM_VALUES	USER_REPCAT_USER_PARM_VALUES
ALL_REPCOLUMN	DBA_REPCOLUMN	USER_REPCOLUMN
ALL_REPCOLUMN_GROUP	DBA_REPCOLUMN_GROUP	USER_REPCOLUMN_GROUP
ALL_REPCONFLICT	DBA_REPCONFLICT	USER_REPCONFLICT
ALL_REPDDL	DBA_REPDDL	USER_REPDDL
ALL_REPGENOBJECTS	DBA_REPGENOBJECTS	USER_REPGENOBJECTS
ALL_REPGROUP	DBA_REPGROUP	USER_REPGROUP
ALL_REPGROUP_PRIVILEGES	DBA_REPGROUP_PRIVILEGES	USER_REPGROUP_PRIVILEGES
ALL_REPGROUPED_COLUMN	DBA_REPGROUPED_COLUMN	USER_REPGROUPED_COLUMN
ALL_REPKEY_COLUMNS	DBA_REPKEY_COLUMNS	USER_REPKEY_COLUMNS
ALL_REPOBJECT	DBA_REPOBJECT	USER_REPOBJECT
ALL_REPPARAMETER_COLUMN	DBA_REPPARAMETER_COLUMN	USER_REPPARAMETER_COLUMN
ALL_REPPRIORITY	DBA_REPPRIORITY	USER_REPPRIORITY
ALL_REPPRIORITY_GROUP	DBA_REPPRIORITY_GROUP	USER_REPPRIORITY_GROUP
ALL_REPPROP	DBA_REPPROP	USER_REPPROP
ALL_REPRESOL_STATS_CONTROL	DBA_REPRESOL_STATS_CONTROL	USER_REPRESOL_STATS_CONTROL
ALL_REPRESOLUTION	DBA_REPRESOLUTION	USER_REPRESOLUTION
ALL_REPRESOLUTION_METHOD	DBA_REPRESOLUTION_METHOD	USER_REPRESOLUTION_METHOD
ALL_REPRESOLUTION_STATISTICS	DBA_REPRESOLUTION_STATISTICS	USER_REPRESOLUTION_STATISTICS
ALL_REPSITES	DBA_REPSITES	USER_REPSITES

ALL_REPCATLOG

Contains the interim status of any asynchronous administrative requests and any error messages generated at each master site. This view contains administrative requests and error messages that are accessible to the current user. All messages encountered while executing a request are eventually transferred to the ALL_REPCATLOG view at the master that originated the request. If an administrative request completes without error, ultimately all traces of this request are removed from the ALL_REPCATLOG view.

Related views:

- DBA_REPCATLOG describes the status for all asynchronous administrative requests and all error messages in the database.
- USER_REPCATLOG describes the status for all asynchronous administrative requests and all error messages owned by the current user.

Column	Datatype	NULL	Description
ID	NUMBER		A sequence number. Together, the ID and SOURCE columns identify all log records at all master sites that pertain to a single administrative request.
SOURCE	VARCHAR2(128)		Location where the request originated.
USERID	VARCHAR2(30)		Name of the user making the request.
TIMESTAMP	DATE		When the request was made.
ROLE	VARCHAR2(9)		Indicates if site is the master definition site (masterdef) or a master site (master).
MASTER	VARCHAR2(128)		If the role is 'masterdef' and the task is remote, indicates which master is performing the task.
SNAME	VARCHAR2(30)		The name of the schema for the replicated object, if applicable.
REQUEST	VARCHAR2(29)		The name of the DBMS_REPCAT administrative procedure that was run.
ONAME	VARCHAR2(30)		The name of the replicated object, if applicable.
TYPE	VARCHAR2(12)		The type of replicated object.
STATUS	VARCHAR2(14)		The status of the administrative request: ready, do_ callback, await_callback, or error.
MESSAGE	VARCHAR2(200)		Any error message that has been returned.
ERRNUM	NUMBER		The Oracle error number for the message.
GNAME	VARCHAR2(30)		The name of the replicated object group.

ALL_REPCAT_REFRESH_TEMPLATES

Contains global information about each deployment template accessible to the current user, such as the template name, template owner, what refresh group the template objects belong to, and the type of template (private vs. public).

When the DBA adds snapshot definitions to the template container, the DBA references the appropriate REFRESH_TEMPLATE_NAME. Any snapshots added to a specific template are added to the refresh group specified in REFRESH_GROUP_NAME.

Furthermore, deployment templates created as public are available to all users who can connect to the master site. Deployment templates created as private are limited to those users listed in the ALL_REPCAT_USER_AUTHORIZATIONS view.

Related views:

- DBA_REPCAT_REFRESH_TEMPLATES describes all deployment templates in the database.
- USER_REPCAT_REFRESH_TEMPLATES describes all deployment templates owned by the current user.

Column	Datatype	NULL	Description
REFRESH_TEMPLATE_NAME	VARCHAR2(30)		Name of the deployment template.
OWNER	VARCHAR2(30)		Owner of the deployment template.
REFRESH_GROUP_NAME	VARCHAR2(30)		Name of the refresh group to which the template objects are added during the instantiation process.
TEMPLATE_COMMENT	VARCHAR2(2000)		User supplied comment.
PUBLIC_TEMPLATE	VARCHAR2(1)		If Y then the deployment template is public. If N then the deployment template is private.

ALL_REPCAT_TEMPLATE_OBJECTS

Contains the individual object definitions that are contained in each deployment template accessible to the current user. Individual objects are added to a template by specifying the target template in REFRESH_TEMPLATE_NAME.

DDL_TEXT can contain variables to create parameterized templates. Variables are created by placing a colon (:) at the beginning of the variable name (that is, :region). Parameterized templates allow for greater flexibility during the template instantiation process (that is, in defining data sets specific for a snapshot site).

When the object is added to the template, the specified DDL is examined and if any parameters have been defined, Oracle automatically adds the parameter to the ALL_REPCAT_TEMPLATE_PARMS view.

Related views:

- DBA_REPCAT_TEMPLATE_OBJECTS describes the object definitions for all deployment templates in the database.
- USER_REPCAT_TEMPLATE_OBJECTS describes the object definitions for each deployment template owned by the current user.

Column	Datatype	NULL	Description
REFRESH_TEMPLATE_NAME	VARCHAR2(30)	NOT NULL	The name of the deployment template.
OBJECT_NAME	VARCHAR2(30)	NOT NULL	The name of the deployment template object.
OBJECT_TYPE	VARCHAR2(17)		The object type of the deployment template object.
DDL_TEXT	CLOB(4000)		The DDL that is executed to create the deployment template object.
MASTER_ROLLBACK_SEGMENT	VARCHAR2(30)		The name of the rollback segment that is used during the instantiation of the deployment template object.
DERIVED_FROM_SNAME	VARCHAR2(30)		If applicable, displays the schema that contains the object from which the template object was created.
DERIVED_FROM_ONAME	VARCHAR2(30)		If applicable, displays the name of the object from which the template object was created.
FLAVOR_ID	NUMBER		The flavor ID of the deployment template object.

Because the DDL_TEXT parameter is defined as a CLOB, you receive an error if you simply try to perform a SELECT on the ALL_REPCAT_TEMPLATE_OBJECTS view. If you do not need to see the object DDL, use the following select statement (be sure to exclude the DDL_TEXT parameter):

```
SELECT refresh_template_name, object_name, object_type, master_rollback_seg,
flavor_id FROM dba_repcat_template_objects;
```

The following script uses cursors and the DBMS_LOB package to view the entire contents of the ALL_REPCAT_TEMPLATE_OBJECTS view. Use this script to view the entire contents of the ALL_REPCAT_TEMPLATE_OBJECTS view, including the DDL_TEXT column:

```
SET SERVEROUTPUT ON

DECLARE
  CURSOR mycursor IS
        SELECT refresh_template_name, object_name, object_type, ddl_text,
          master_rollback_seg, flavor_id
          FROM dba_repcat_template_objects;
  tempstring VARCHAR2(1000);
  len NUMBER;
BEGIN
  FOR myrec IN mycursor LOOP
        len := DBMS_LOB.GETLENGTH(myrec.ddl_text);
        DBMS_LOB.READ(myrec.ddl_text, len, 1, tempstring);
        DBMS_OUTPUT.PUT_LINE(myrec.refresh_template_name||' '||
          myrec.object_name||' '||myrec.object_type||' '||tempstring||' '||
          myrec.master_rollback_seg||' '||myrec.flavor_id);
  END LOOP;
END;
/
```

> **See Also:** *Oracle8i Application Developer's Guide - Fundamentals* for more information on using cursors. Also, see *Oracle8i Application Developer's Guide - Large Objects (LOBs)* for more information on using the DBMS_LOB package and LOBs in general.

ALL_REPCAT_TEMPLATE_PARMS

Contains parameters defined in the object DDL for all templates accessible to the current user. When an object is added to a template, the DDL is examined for variables. Any found parameters are automatically added to this view.

You can also define default parameter values and a prompt string in this view. These can make the templates easier to use during the instantiation process.

See Also: ALL_REPCAT_TEMPLATE_OBJECTS on page 9-9.

Related views:

- DBA_REPCAT_TEMPLATE_PARMS describes the template parameters for all deployment templates in the database.
- USER_REPCAT_TEMPLATE_PARMS describes the template parameters for all deployment templates owned by the current user.

Column	Datatype	NULL	Description
REFRESH_TEMPLATE_NAME	VARCHAR2(30)	NOT NULL	The name of the deployment template.
OWNER	VARCHAR2(30)	NOT NULL	The owner of the deployment template.
REFRESH_GROUP_NAME	VARCHAR2(30)	NOT NULL	Name of the refresh group to which the template objects are added to during the instantiation process.
TEMPLATE_COMMENTS	VARCHAR2(2000)		User specified comments.
PUBLIC_TEMPLATE	VARCHAR2(1)		If Y then the deployment template is public. If N then the deployment template is private.
PARAMETER_NAME	VARCHAR2(30)	NOT NULL	The name of the parameter.
DEFAULT_PARM_VALUE	CLOB(4000)		The default parameter value.
PROMPT_STRING	VARCHAR2(2000)		The prompt string for the parameter.
USER_OVERRIDE	VARCHAR2(1)		If Y then the user can override the default parameter value. If N then the user can not override the default parameter value.

Because DEFAULT_PARM_VALUE is defined as a CLOB, you receive an error if you simply try to perform a SELECT on the ALL_REPCAT_TEMPLATE_PARMS view. If you do not need to see the default parameter value, use the following select statement (be sure to exclude DEFAULT_PARM_VALUE):

```
SELECT refresh_template_name, owner, refresh_group_name, template_comment,
  public_template, parameter_name, prompt_string, user_override
  FROM dba_repcat_template_parms;
```

The following script uses cursors and the DBMS_LOB package to view the entire contents of the ALL_REPCAT_TEMPLATE_PARMS view. Use this script to view the entire contents of the ALL_REPCAT_TEMPLATE_PARMS view, including the DEFAULT_PARM_VALUE column:

```
SET SERVEROUTPUT ON

DECLARE
  CURSOR mycursor IS
        SELECT refresh_template_name, owner, refresh_group_name,
          template_comment, public_template, parameter_name, default_parm_value,
          prompt_string, user_override
          FROM dba_repcat_template_parms;
  tempstring VARCHAR2(1000);
  len NUMBER;
BEGIN
  FOR myrec IN mycursor LOOP
        len := DBMS_LOB.GETLENGTH(myrec.default_parm_value);
        DBMS_LOB.READ(myrec.default_parm_value, len, 1, tempstring);
        DBMS_OUTPUT.PUT_LINE(myrec.refresh_template_name||' '||
          myrec.owner||' '||myrec.refresh_group_name||' '||
          myrec.template_comment||' '||myrec.public_template||' '||
          myrec.parameter_name||' '||tempstring||' '||myrec.prompt_string||' '||
          myrec.user_override);
  END LOOP;
END;
/
```

See Also: *Oracle8i Application Developer's Guide - Fundamentals* for more information on using cursors. Also, see *Oracle8i Application Developer's Guide - Large Objects (LOBs)* for more information on using the DBMS_LOB package and LOBs in general.

ALL_REPCAT_TEMPLATE_SITES

Contains information about the current status of template instantiation among the sites of an enterprise network. This view contains information about instantiation sites for deployment templates that are accessible to the current user. Specifically, the DBA can monitor the installation and deletion of templates at specific sites, including RepAPI sites.

Related views:

- DBA_REPCAT_TEMPLATE_SITES describes all remote instantiation sites for all templates in the database.
- USER_REPCAT_TEMPLATE_SITES describes remote instantiation sites for all templates owned by the current user.

Column	Datatype	NULL	Description
REFRESH_TEMPLATE_NAME	VARCHAR2(30)	NOT NULL	Name of the deployment template.
REFRESH_GROUP_NAME	VARCHAR2(30)		Name of the refresh group to which template objects are added during the instantiation process.
TEMPLATE_OWNER	VARCHAR2(30)		Name of the user who is considered the owner of the deployment template.
USER_NAME	VARCHAR2(30)	NOT NULL	The name of the user who instantiated the deployment template.
SITE_NAME	VARCHAR2(128)		Target snapshot site of the deployment template. This field is NULL for RepAPI sites.
REPAPI_SITE_NAME	VARCHAR2(128)		The ID of the RepAPI site that has instantiated the displayed deployment template. This field is NULL for non-RepAPI clients.
STATUS	VARCHAR2(10)		Displays the status of the deployment template at the target snapshot site: 0 = Not Installed 1 = Installed -1 = Installed with errors

ALL_REPCAT_USER_AUTHORIZATIONS

Lists the authorized users for private deployment templates accessible to the current user. Users listed in this view have the ability to instantiate the specified template. Users not contained in this view cannot instantiate the deployment template.

Related views:

- DBA_REPCAT_USER_AUTHORIZATIONS lists the authorized users for all the private deployment templates in the database.
- USER_REPCAT_USER_AUTHORIZATIONS lists the authorized users for private deployment templates owned by the current user.

Column	Datatype	NULL	Description
REFRESH_TEMPLATE_NAME	VARCHAR2(30)	NOT NULL	Name of the deployment template that a user has been authorized to instantiate.
OWNER	VARCHAR2(30)	NOT NULL	Name of the owner of the deployment template.
REFRESH_GROUP_NAME	VARCHAR2(30)	NOT NULL	Name of the refresh group to which template objects are added during the instantiation process.
TEMPLATE_COMMENT	VARCHAR2(2000)		User specified comment.
PUBLIC_TEMPLATE	VARCHAR2(1)		If Y then the deployment template is public. If N then the deployment template is private.
USER_NAME	VARCHAR2(30)	NOT NULL	Name of the user who has been authorized to instantiate the deployment template.

ALL_REPCAT_USER_PARM_VALUES

This view describes the template parameters for all deployment templates accessible to the current user. The DBA has the option of building a table of user parameters prior to distributing the template for instantiation. When a template is instantiated by a specified user, the values stored in the ALL_REPCAT_USER_PARM_VALUES view for the specified user are used automatically.

Related views:

- DBA_REPCAT_USER_PARM_VALUES describes the template parameters for all deployment templates in the database.
- USER_REPCAT_USER_PARM_VALUES describes the template parameters for all deployment templates owned by the current user.

Column	Datatype	NULL	Description
REFRESH_TEMPLATE_NAME	VARCHAR2(30)	NOT NULL	The name of the deployment template for which a user parameter value has been defined.
OWNER	VARCHAR2(30)	NOT NULL	The name of the owner of the deployment template.
REFRESH_GROUP_NAME	VARCHAR2(30)	NOT NULL	Name of the refresh group to which the template objects are added to during the instantiation process.
TEMPATE_COMMENT	VARCHAR2(2000)		User specified comment.
PUBLIC_TEMPLATE	VARCHAR2(1)		If Y then the deployment template is public. If N then the deployment template is private.
PARAMETER_NAME	VARCHAR2(30)	NOT NULL	The name of the parameter for which a user parameter value has been defined.
DEFAULT_PARM_VALUE	CLOB(4000)		The default value for the parameter.
PROMPT_STRING	VARCHAR2(2000)		The prompt string for the parameter.
PARM_VALUE	CLOB(4000)		The parameter value that has been defined for the specified user.
USER_NAME	VARCHAR2(30)	NOT NULL	The username of the user for whom the specified parameter value has been defined.

Because DEFAULT_PARM_VALUE and PARM_VALUE are defined as CLOBs, you receive an error if you simply try to perform a SELECT on the ALL_REPCAT_USER_PARM_VALUES view. If you do not need to see the default or user parameter values, use the following select statement (be sure to exclude DEFAULT_PARM_VALUE and PARM_VALUE):

```
SELECT refresh_template_name, owner, refresh_group_name, template_comment,
  public_template, parameter_name, prompt_string, user_name
  FROM dba_repcat_user_parm_values;
```

The following script uses cursors and the DBMS_LOB package to view the entire contents of the ALL_REPCAT_USER_PARM_VALUES view. Use this script to view the entire contents of the ALL_REPCAT_TEMPLATE_PARMS view, including the DEFAULT_PARM_VALUE and PARM_VALUE columns:

```
SET SERVEROUTPUT ON

DECLARE
  CURSOR mycursor IS
        SELECT refresh_template_name, owner, refresh_group_name,
          template_comment, public_template, parameter_name, default_parm_value,
          prompt_string, parm_value, user_name
          FROM dba_repcat_user_parm_values;
  tempstring VARCHAR2(1000);
  tempstring2 varchar2(1000);
  len NUMBER;
BEGIN
  FOR myrec IN mycursor LOOP
        len := DBMS_LOB.GETLENGTH(myrec.default_parm_value);
        DBMS_LOB.READ(myrec.default_parm_value, len, 1, tempstring);
        DBMS_OUTPUT.PUT_LINE(myrec.refresh_template_name||' '||
          myrec.owner||' '||myrec.refresh_group_name||' '||
          myrec.template_comment||' '||myrec.public_template||' '||
          myrec.parameter_name||' '||tempstring||' '||myrec.prompt_string||' '||
          tempstring2||' '||myrec.user_name);
  END LOOP;
END;
/
```

See Also: *Oracle8i Application Developer's Guide - Fundamentals* for more information on using cursors. Also, see *Oracle8i Application Developer's Guide - Large Objects (LOBs)* for more information on using the DBMS_LOB package and LOBs in general.

ALL_REPCOLUMN

Lists the replicated columns for the tables accessible to the current user.

Related views:

- DBA_REPCOLUMN describes the replicated columns for all the tables in the database.
- USER_REPCOLUMN describes the replicated columns for all the tables owned by the current user.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the object owner.
ONAME	VARCHAR2(30)	NOT NULL	The name of the object.
TYPE	VARCHAR2(8)		The type of the object.
CNAME	VARCHAR2(30)	NOT NULL	The name of the replicated column.
ID	NUMBER		The ID number of the replicated column.
POS	NUMBER	NOT NULL	The ordering of the replicated column.
COMPARE_OLD_ON_DELETE	VARCHAR2(1)		Indicates whether Oracle compares the old value of the column in replicated deletes.
COMPARE_OLD_ON_UPDATE	VARCHAR2(1)		Indicates whether Oracle compares the old value of the column in replicated updates.
SEND_OLD_ON_DELETE	VARCHAR2(1)		Indicates whether Oracle sends the old value of the column in replicated deletes.
SEND_OLD_ON_UPDATE	VARCHAR2(1)		Indicates whether Oracle sends the old value of the column in replicated updates.
CTYPE	VARCHAR2(9)		Displays the column type.
DATA_LENGTH	NUMBER		Displays the length of the column in bytes.
DATA_PRECISION	NUMBER		Displays the column precision in terms of decimal digits for NUMBER columns or binary digits for FLOAT columns.
DATA_SCALE	NUMBER		Displays the digits to right of decimal point in a number.
NULLABLE	VARCHAR2(1)		Indicates if the column allow NULL values.
CHARACTER_SET_NAME	VARCHAR2(44)		If applicable, displays the name of character set for the column.

ALL_REPCOLUMN_GROUP

Describes the column groups for each replicated table accessible to the current user.

Related views:

- DBA_REPCOLUMN_GROUP describes the column groups for all the tables in the database.
- USER_REPCOLUMN_GROUP describes the column groups for all the tables owned by the current user.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the schema containing the replicated table.
ONAME	VARCHAR2(30)	NOT NULL	The name of the replicated table.
GROUP_NAME	VARCHAR2(30)	NOT NULL	The column group name.
GROUP_COMMENT	VARCHAR2(80)		Any user-supplied comments.

Note: The SNAME column is not present in the USER_REPCOLUMN_GROUP view.

ALL_REPCONFLICT

Contains the name of each table accessible to the current user for which a conflict resolution method has been defined and the type of conflict that the method is used to resolve.

Related views:

- DBA_REPCONFLICT describes the conflict resolution method for all the tables in the database on which a conflict resolution method has been defined.
- USER_REPCONFLICT describes the conflict resolution method for all the tables owned by the current user on which a conflict resolution method has been defined.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the schema containing the replicated table.
ONAME	VARCHAR2(30)	NOT NULL	The name of the table for which a conflict resolution method has been defined.
CONFLICT_TYPE	VARCHAR2(10)		The type of conflict that the conflict resolution method is used to resolve: delete, uniqueness, or update.
REFERENCE_NAME	VARCHAR2(30)	NOT NULL	The object to which the method applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name.

Note: The SNAME column is not present in the USER_REPCONFLICT view.

ALL_REPDDL

Contains the DDL for each replication object accessible to the current user.

Related views:

- DBA_REPDDL contains the DDL for each replicated object in the database.
- USER_REPDDL contains the DDL for each replicated object owned by the current user.

Column	Datatype	NULL	Description
LOG_ID	NUMBER		Identifying number of the ALL_REPCATLOG record.
SOURCE	VARCHAR2(128)		Name of the database at which the request originated.
ROLE	VARCHAR2(1)		If Y then this database is the master definition site (masterdef) for the request. If N then this database is a master.
MASTER	VARCHAR2(128)		Name of the database that processes this request.
LINE	NUMBER(38)		Ordering of records within a single request.
TEXT	VARCHAR2(2000)		Portion of an argument or DDL text.

ALL_REPGENOBJECTS

Describes each object accessible to the current user that was generated to support replication.

Related views:

- DBA_REPGENOBJECTS describes each object in the database that was generated to support replication.
- USER_REPGENOBJECTS describes each object owned by the current user that was generated to support replication.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)		The name of the replicated schema.
ONAME	VARCHAR2(30)		The name of the generated object.
TYPE	VARCHAR2(12)		The type of the generated object.
BASE_SNAME	VARCHAR2(30)		The base object's owner.
BASE_ONAME	VARCHAR2(30)		The object for which this object was generated.
BASE_TYPE	VARCHAR2(12)		The type of the base object.
PACKAGE_PREFIX	VARCHAR2(30)		The prefix for the package wrapper.
PROCEDURE_PREFIX	VARCHAR2(30)		The procedure prefix for the package wrapper.
DISTRIBUTED	VARCHAR2(1)		This column is obsolete.
REASON	VARCHAR2(30)		The reason the object was generated.

ALL_REPGROUP

Describes all of the object groups accessible to the current user that are being replicated. The members of each object group are listed in a different view: ALL_REPOBJECT.

Related views:

- DBA_REPGROUP describes all of the object groups in the database that are being replicated.
- USER_REPGROUP describes all of the object groups owned by the current user that are being replicated.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the replicated schema. Obsolete with release 7.3 or later.
MASTER	VARCHAR2(1)		Y indicates that the current site is a master site. N indicates the current site is a snapshot site.
STATUS	VARCHAR2(9)		Used at master sites only. Status can be: normal, quiescing, or quiesced.
SCHEMA_COMMENT	VARCHAR2(80)		Any user-supplied comments.
GNAME	VARCHAR2(30)	NOT NULL	The name of the replicated object group.
FNAME	VARCHAR2(30)		Flavor name.
RPC_PROCESSING_DISABLED	VARCHAR2(1)		N indicates that this site can receive and apply deferred RPCs. Y indicates that this site can NOT receive and apply deferred RPCs.
OWNER	VARCHAR2(30)	NOT NULL	Owner of the replication group.

ALL_REPGROUP_PRIVILEGES

Contains information about the users who are registered for privileges in the object groups accessible to the current user.

Related views:

- DBA_REPGROUP_PRIVILEGES contains information about the users who are registered for privileges in all the object groups in the database.
- USER_REPGROUP_PRIVILEGES contains information about the users who are registered for privileges in the object groups owned by the current user.

Column	Datatype	NULL	Description
USERNAME	VARCHAR2(30)	NOT NULL	Displays the name of the user.
GNAME	VARCHAR2(30)		Displays the name of the replicated object group.
CREATED	DATE	NOT NULL	Displays the date that the object group was registered.
RECEIVER	VARCHAR2(1)		Indicates whether the user has receiver privileges.
PROXY_SNAPADMIN	VARCHAR2(1)		Indicates whether the user has proxy_snapadmin privileges.
OWNER	VARCHAR2(30)		Owner of the replication group.

ALL_REPGROUPED_COLUMN

Describes all of the columns that make up the column groups for each table accessible to the current user.

Related views:

- DBA_REPGROUPED_COLUMN describes all of the columns that make up the column groups for each table in the database.
- USER_REPGROUPED_COLUMN describes all of the columns that make up the column groups for each table owned by the current user.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the schema containing the replicated table.
ONAME	VARCHAR2(30)	NOT NULL	The name of the replicated table.
GROUP_NAME	VARCHAR2(30)	NOT NULL	The name of the column group.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	The name of the column in the column group.

Note: The SNAME column is not present in the USER_REPGROUPED_COLUMN version of the view.

ALL_REPKEY_COLUMNS

Describes the primary key column(s) in each table accessible to the current user.

Related views:

- DBA_REPKEY_COLUMNS describes the primary key column(s) in each table in the database.
- USER_REPKEY_COLUMNS describes the primary key column(s) in each table owned by the current user.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	Owner of the replicated table.
ONAME	VARCHAR2(30)	NOT NULL	Name of the replicated table.
COL	VARCHAR2(30)	NOT NULL	Primary key column(s) in the table.

ALL_REPOBJECT

Contains information about the objects in each replicated object group accessible to the current user. An object can belong to only one object group. A replicated object group can span multiple schemas.

Related views:

- DBA_REPOBJECT contains information about the objects in each replicated object group in the database.
- USER_REPOBJECT contains information about the objects in each replicated object group owned by the current user.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)		The name of the schema containing the replicated object.
ONAME	VARCHAR2(30)		The name of the replicated object.
TYPE	VARCHAR2(16)		The type of replicated object: table, view, package, package body, procedure, function, index, synonym, trigger, or snapshot.
STATUS	VARCHAR2(9)		CREATE indicates that Oracle is applying user supplied or Oracle-generated DDL to the local database in an attempt to create the object locally. When a local replica exists, Oracle COMPAREs the replica with the master definition to ensure that they are consistent. When creation or comparison complete successfully, Oracle updates the status to VALID. Otherwise, it updates the status to ERROR. If you drop an object, Oracle updates its status to DROPPED before deleting the row from the ALL_REPOBJECT view.
GENERATION_STATUS	VARCHAR2(9)		Whether the object needs to generate replication packages.
ID	NUMBER		The identifier of the local database object, if one exists.
OBJECT_COMMENT	VARCHAR2(80)		Any user supplied comments.
GNAME	VARCHAR2(30)		The name of the replicated object group to which the object belongs.
MIN_COMMUNICATION	VARCHAR2(1)		If Y then send only new values for an updated view. If N then send both OLD and NEW values for an updated view.
REPLICATION_TRIGGER_EXISTS	VARCHAR2(1)		If Y then internal replication trigger exists. If N then internal replication trigger does not exist.
INTERNAL_PACKAGE_EXISTS	VARCHAR2(1)		If Y then internal package exists. If N then internal package does not exist.
GROUP_OWNER	VARCHAR2(30)		Owner of the replication group.

ALL_REPPARAMETER_COLUMN

In addition to the information contained in the ALL_REPRESOLUTION view, the ALL_REPPARAMETER_COLUMN view contains information about the columns that are used to resolve conflicts for each replicated table accessible to the current user. These are the column values that are passed as the LIST_OF_COLUMN_NAMES argument to the ADD_*conflicttype*_RESOLUTION procedures in the DBMS_REPCAT package.

Related views:

- DBA_REPPARAMETER COLUMN contains information about the columns that are used to resolve conflicts for each replicated table in the database.
- USER_REPPARAMETER_COLUMN contains information about the columns that are used to resolve conflicts for each replicated table owned by the current user.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the schema containing the replicated table.
ONAME	VARCHAR2(30)	NOT NULL	The name of the replicated table.
CONFLICT_TYPE	VARCHAR2(10)		The type of conflict that the method is used to resolve: delete, uniqueness, or update.
REFERENCE_NAME	VARCHAR2(30)	NOT NULL	The object to which the method applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name.
SEQUENCE_NO	NUMBER	NOT NULL	The order in which resolution methods are applied, with 1 applied first.
METHOD_NAME	VARCHAR2(80)	NOT NULL	The name of an Oracle-supplied conflict resolution method. For user-supplied methods, this value is 'user function'.
FUNCTION_NAME	VARCHAR2(92)	NOT NULL	For methods of type 'user function', the name of the user-supplied conflict resolution method.
PRIORITY_GROUP	VARCHAR2(30)		For methods of name 'priority group', the name of the priority group.
PARAMETER_TABLE_NAME	VARCHAR2(30)	NOT NULL	Displays the name of the table to which the parameter column belongs.
PARAMETER_COLUMN_NAME	VARCHAR2(30)	NOT NULL	The name of the column used as the IN parameter for the conflict resolution method.
PARAMETER_SEQUENCE_NO	NUMBER	NOT NULL	Ordering of column used as IN parameter.

Note: The SNAME column is not present in the USER_REPPARAMETER_COLUMN view.

ALL_REPPRIORITY

Contains the value and priority level of each priority group member in each priority group accessible to the current user. Priority group names must be unique within a replicated object group. Priority levels and values must each be unique within a given priority group.

Related views:

- DBA_REPPRIORITY contains the value and priority level of each priority group member in each priority group in the database.
- USER_REPPRIORITY contains the value and priority level of each priority group member in each priority group owned by the current user.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the replicated schema. Obsolete with release 7.3 or later.
PRIORITY_GROUP	VARCHAR2(30)	NOT NULL	The name of the priority group or site priority group.
PRIORITY	NUMBER	NOT NULL	The priority level of the member. The highest number has the highest priority.
DATA_TYPE	VARCHAR2(9)		The datatype of the values in the priority group.
FIXED_DATA_LENGTH	NUMBER(38)		The maximum length of values of datatype CHAR.
CHAR_VALUE	CHAR(255)		The value of the priority group member, if DATA_TYPE = CHAR.
VARCHAR2_VALUE	VARCHAR2(4000)		The value of the priority group member, if DATA_TYPE = VARCHAR2.
NUMBER_VALUE	NUMBER		The value of the priority group member, if DATA_TYPE = NUMBER.
DATE_VALUE	DATE		The value of the priority group member, if DATA_TYPE = DATE.
RAW_VALUE	RAW(2000)		The value of the priority group member, if DATA_TYPE = RAW.
GNAME	VARCHAR2(30)	NOT NULL	The name of the replicated object group.
NCHAR_VALUE	CHAR(500)		The value of the priority group member, if DATA_TYPE = NCHAR.
NVARCHAR2_VALUE	VARCHAR2(1000)		The value of the priority group member, if DATA_TYPE = NVARCHAR2.
LARGE_CHAR_VALUE	CHAR(2000)		The value of the priority group member, for blank-padded character strings over 255 characters.

Note: The SNAME and GNAME columns are not present in the USER_REPPRIORITY view.

ALL_REPPRIORITY_GROUP

Describes the priority group or site priority group defined for each replicated object group accessible to the current user.

Related views:

- DBA_REPPRIORITY_GROUP describes the priority group or site priority group defined for each replicated object group in the database.
- USER_REPPRIORITY_GROUP describes the priority group or site priority group defined for each replicated object group owned by the current user.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the replicated schema. Obsolete with release 7.3 or later.
PRIORITY_GROUP	VARCHAR2(30)	NOT NULL	The name of the priority group or site priority group.
DATA_TYPE	VARCHAR2(9)		The datatype of each value in the priority group.
FIXED_DATA_LENGTH	NUMBER(38)		The maximum length for values of datatype CHAR.
PRIORITY_COMMENT	VARCHAR2(80)		Any user-supplied comments.
GNAME	VARCHAR2(30)	NOT NULL	The name of the replicated object group.

Note: The SNAME and GNAME columns are not present in the USER_REPPRIORITY view.

ALL_REPPROP

Indicates the technique used to propagate operations on each replicated object to the same object at another master site. This view shows objects accessible to the current user. These operations may have resulted from a call to a stored procedure or procedure wrapper, or may have been issued against a table directly.

Related views:

- DBA_REPPROP indicates the technique used to propagate operations on each replicated object to the same object at another master site. This view shows all objects in the database.
- USER_REPPROP indicates the technique used to propagate operations on each replicated object to the same object at another master site. This view shows objects owned by the current user

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the schema containing the replicated object.
ONAME	VARCHAR2(30)	NOT NULL	The name of the replicated object.
TYPE	VARCHAR2(16)		The type of object being replicated.
DBLINK	VARCHAR2(128)	NOT NULL	The fully qualified database name of the master site to which changes are being propagated.
HOW	VARCHAR2(13)		How propagation is performed. Values recognized are 'none' for the local master site, and 'synchronous' or 'asynchronous' for all others.
PROPAGATE_COMMENT	VARCHAR2(80)		Any user-supplied comments.

ALL_REPRESOL_STATS_CONTROL

Describes statistics collection for conflict resolutions for all replicated tables accessible to the current user.

Related views:

- DBA_REPRESOL_STATS_CONTROL describes statistics collection for conflict resolutions for all replicated tables in the database.
- USER_REPRESOL_STATS_CONTROL describes statistics collection for conflict resolutions for all replicated tables owned by the current user.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	Owner of the table.
ONAME	VARCHAR2(30)	NOT NULL	Table name.
CREATED	DATE	NOT NULL	Timestamp for when statistics collection was first started.
STATUS	VARCHAR2(9)		Status of statistics collection: ACTIVE or CANCELLED.
STATUS_UPDATE_DATE	DATE	NOT NULL	Timestamp for when the status was last updated.
PURGED_DATE	DATE		Timestamp for the last purge of statistics data.
LAST_PURGE_START_DATE	DATE		The last start date of the statistics purging date range.
LAST_PURGE_END_DATE	DATE		The last end date of the statistics purging date range.

Note: The SNAME column is not present in the USER_REPRESOL_STATS_CONTROL view.

ALL_REPRESOLUTION

Indicates the methods used to resolve update, uniqueness, or delete conflicts for each table accessible to the current user that is replicated using row-level replication for a given schema.

Related views:

- DBA_REPRESOLUTION indicates the methods used to resolve update, uniqueness, or delete conflicts for each table in the database that is replicated using row-level replication for a given schema.
- USER_REPRESOLUTION indicates the methods used to resolve update, uniqueness, or delete conflicts for each table owned by the current user that is replicated using row-level replication.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the replicated schema.
ONAME	VARCHAR2(30)	NOT NULL	The name of the replicated table.
CONFLICT_TYPE	VARCHAR2(10)		The type of conflict that the method is used to resolve: delete, uniqueness, or update.
REFERENCE_NAME	VARCHAR2(30)	NOT NULL	The object to which the method applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name.
SEQUENCE_NO	NUMBER	NOT NULL	The order that resolution methods are applied, with 1 applied first.
METHOD_NAME	VARCHAR2(80)	NOT NULL	The name of an Oracle-supplied conflict resolution method. For user-supplied methods, this value is 'user function'.
FUNCTION_NAME	VARCHAR2(92)	NOT NULL	For methods of type 'user function', the name of the user-supplied conflict resolution method.
PRIORITY_GROUP	VARCHAR2(30)		For methods of type 'priority group', the name of the priority group.
RESOLUTION_COMMENT	VARCHAR2(80)		Any user-supplied comments.

Note: The SNAME column is not present in the USER_REPRESOLUTION view.

ALL_REPRESOLUTION_METHOD

Lists all of the conflict resolution methods available in the database. Initially, this view lists the standard methods provided with Oracle replication. As you create new user functions and add them as conflict resolution methods for an object in the database, these functions are added to this view.

Related views:

- DBA_REPRESOLUTION_METHOD lists all of the conflict resolution methods available in the database.
- USER_REPRESOLUTION_METHOD lists all of the conflict resolution methods available in the database.

Column	Datatype	NULL	Description
CONFLICT_TYPE	VARCHAR2(10)		The type of conflict that the resolution method is designed to resolve: update, uniqueness, or delete.
METHOD_NAME	VARCHAR2(80)	NOT NULL	The name of the Oracle-supplied method, or the name of the user-supplied method.

ALL_REPRESOLUTION_STATISTICS

Lists information about successfully resolved update, uniqueness, and delete conflicts for all replicated tables accessible to the current user. These statistics are gathered for a table only if you have called the DBMS_REPCAT.REGISTER_ STATISTICS procedure.

Related views:

- DBA_REPRESOLUTION_STATISTICS lists information about successfully resolved update, uniqueness, and delete conflicts for all replicated tables in the database.
- USER_REPRESOLUTION_STATISTICS lists information about successfully resolved update, uniqueness, and delete conflicts for all replicated tables owned by the current user.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the replicated schema.
ONAME	VARCHAR2(30)	NOT NULL	The name of the replicated table.
CONFLICT_TYPE	VARCHAR2(10)		The type of conflict that was successfully resolved: delete, uniqueness, or update.
REFERENCE_NAME	VARCHAR2(30)	NOT NULL	The object to which the conflict resolution method applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name.
METHOD_NAME	VARCHAR2(80)	NOT NULL	The name of an Oracle-supplied conflict resolution method. For user-supplied methods, this value is 'user function'.
FUNCTION_NAME	VARCHAR2(92)		For methods of type 'user function', the name of the user supplied conflict resolution method.
PRIORITY_GROUP	VARCHAR2(30)		For methods of type 'priority group', the name of the priority group.
RESOLVED_DATE	DATE	NOT NULL	Date on which the conflict for this row was resolved.
PRIMARY_KEY_VALUE	VARCHAR2(2000)	NOT NULL	A concatenated representation of the row's primary key.

Note: The SNAME column is not present in the USER_REPRESOLUTION_STATISTICS view.

ALL_REPSITES

Lists the members of each replicated object group accessible to the current user.

Related views:

- DBA_REPSITES lists the members of each replicated object group in the database.
- USER_REPSITES lists the members of each replicated object group owned by the current user.

Column	Datatype	NULL	Description
GNAME	VARCHAR2(30)	NOT NULL	The name of the replicated object group.
DBLINK	VARCHAR2(128)	NOT NULL	The database link to a master site for this object group.
MASTERDEF	VARCHAR2(1)		Indicates which of the DBLINKs is the master definition site.
SNAPMASTER	VARCHAR2(1)		Used by snapshot sites to indicate which of the DBLINKs to use when refreshing.
MASTER_COMMENT	VARCHAR2(80)		User-supplied comments.
MASTER	VARCHAR2(1)		If Y then the site is a master site for the replicated group. If N then the site is not a master site for the replicated group.
GROUP_OWNER	VARCHAR2(30)	NOT NULL	Owner of the replication group.

The DBA_REPSITES view has the following additional columns:

PROP_UPDATES	NUMBER		Encoding of propagating technique for master.
MY_DBLINK	VARCHAR2(1)		Used to detect problems after import. If Y then the DBLINK is the global name.

DBA_REPCATLOG

The DBA_REPCATLOG view at each master site contains the interim status of any asynchronous administrative requests and any error messages generated. All messages encountered while executing a request are eventually transferred to the DBA_REPCATLOG view at the master that originated the request. If an administrative request completes without error, ultimately all traces of this request are removed from the DBA_REPCATLOG view. Its columns are the same as those in "ALL_REPCATLOG" on page 9-6.

DBA_REPCAT_REFRESH_TEMPLATES

This view contains global information about each deployment template in the database, such as the template name, template owner, what refresh group the template objects belong to, and the type of template (private vs. public).

Its columns are the same as those in ALL_REPCAT_REFRESH_TEMPLATES. For detailed information about this view and its columns, see "ALL_REPCAT_REFRESH_TEMPLATES" on page 9-8.

DBA_REPCAT_TEMPLATE_OBJECTS

The DBA_REPCAT_TEMPLATE_OBJECTS view contains the individual object definitions that are contained in all deployment templates in the database. Individual objects are added to a template by specifying the target template in REFRESH_TEMPLATE_NAME.

Its columns are the same as those in ALL_REPCAT_TEMPLATE_OBJECTS. For detailed information about this view and its columns, see "ALL_REPCAT_TEMPLATE_OBJECTS" on page 9-9.

DBA_REPCAT_TEMPLATE_PARMS

Parameters defined in the object DDL for all templates in the database are stored in the DBA_REPCAT_TEMPLATE_PARMS table. When an object is added to a template, the DDL is examined for variables. Any found parameters are automatically added to this view.

Its columns are the same as those in ALL_REPCAT_TEMPLATE_PARMS. For detailed information about this view and its columns, see "ALL_REPCAT_TEMPLATE_PARMS" on page 9-11.

DBA_REPCAT_TEMPLATE_SITES

The DBA_REPCAT_TEMPLATE_SITES view provides the DBA with information about the current status of template instantiation for all the sites of a enterprise network. This view contains information about instantiation sites for all deployment templates in the database. Specifically, the DBA can monitor the installation and deletion of templates at specific sites, including RepAPI sites. Its columns are the same as those in "ALL_REPCAT_TEMPLATE_SITES" on page 9-13.

DBA_REPCAT_USER_AUTHORIZATIONS

The DBA_REPCAT_USER_AUTHORIZATIONS view lists the authorized users for all templates in the database specified for private use. Users listed in this view have the ability to instantiate the specified template. Users not contained in this view cannot instantiate the template. Its columns are the same as those in "ALL_REPCAT_USER_AUTHORIZATIONS" on page 9-14.

DBA_REPCAT_USER_PARM_VALUES

The DBA_REPCAT_USER_PARM_VALUES view describes the template parameters for all deployment templates in the database. The DBA has the option of building a table of user parameters prior to distributing the template for instantiation. When a template is instantiated by a specified user, the values stored in the DBA_REPCAT_USER_PARM_VALUES table for the specified user are used automatically.

Its columns are the same as those in ALL_REPCAT_USER_PARM_VALUES. For detailed information about this view and its columns, see "ALL_REPCAT_USER_PARM_VALUES" on page 9-15.

DBA_REPCOLUMN

The DBA_REPCOLUMN view lists the replicated columns for all the tables in the database. Its columns are the same as those in "ALL_REPCOLUMN" on page 9-17.

DBA_REPCOLUMN_GROUP

The DBA_REPCOLUMN_GROUP view lists all the column groups each replicated table in the database. Its columns are the same as those in "ALL_REPCOLUMN_GROUP" on page 9-18.

DBA_REPCONFLICT

The DBA_REPCONFLICT view displays the name of each table in the database on which a conflict resolution method has been defined and the type of conflict that the method is used to resolve. Its columns are the same as those in "ALL_REPCONFLICT" on page 9-18.

DBA_REPDDL

The DBA_REPDDL contains the DDL for each replication object in the database. Its columns are the same as those in "ALL_REPDDL" on page 9-19.

DBA_REPGENOBJECTS

The DBA_REPGENOBJECTS view describes each object in the database that was generated to support replication. Its columns are the same as those in "ALL_REPGENOBJECTS" on page 9-20.

DBA_REPGROUP

The DBA_REPGROUP view describes all of the object groups in the database that are being replicated. The members of each object group are listed in a different view, DBA_REPOBJECT. The DBA_REPGROUP view's columns are the same as those in "ALL_REPGROUP" on page 9-21.

DBA_REPGROUP_PRIVILEGES

The DBA_REPGROUP_PRIVILEGES view contains information about the users who are registered for privileges in all the object groups in the database. Its columns are the same as those in "ALL_REPGROUP_PRIVILEGES" on page 9-22.

DBA_REPGROUPED_COLUMN

The DBA_REPGROUPED_COLUMN view lists all of the columns that make up the column groups for each table in the database. Its columns are the same as those in "ALL_REPGROUPED_COLUMN" on page 9-22.

DBA_REPKEY_COLUMNS

The DBA_REPKEY_COLUMNS view describes the primary key column(s) in each table in the database. Its columns are the same as those in "ALL_REPKEY_COLUMNS" on page 9-23.

DBA_REPOBJECT

The DBA_REPOBJECT view contains information about the objects in each replicated object group in the database. An object can belong to only one object group. A replicated object group can span multiple schemas. Its columns are the same as those in "ALL_REPOBJECT" on page 9-23.

DBA_REPPARAMETER_COLUMN

In addition to the information contained in the DBA_REPRESOLUTION view, the DBA_REPPARAMETER_COLUMN view contains information about the columns that are used to resolve conflicts for each replicated table in the database. These are the column values that are passed as the LIST_OF_COLUMN_NAMES argument to the ADD_*conflicttype*_RESOLUTION procedures in the DBMS_REPCAT package. The DBA_REPPARAMETER_COLUMN view's columns are the same as those in "ALL_REPPARAMETER_COLUMN" on page 9-25.

DBA_REPPRIORITY

The DBA_REPPRIORITY view contains the value and priority level of each priority group member in each priority group in the database. Priority group names must be unique within a replicated object group. Priority levels and values must each be unique within a given priority group. Its columns are the same as those in "ALL_REPPRIORITY" on page 9-26.

DBA_REPPRIORITY_GROUP

The DBA_REPPRIORITY_GROUP view describes the priority group or site priority group defined for each replicated object group in the database. Its columns are the same as those in "ALL_REPPRIORITY_GROUP" on page 9-28.

DBA_REPPROP

The DBA_REPPROP view indicates the technique used to propagate operations on each replicated object to the same object at another master site. This view shows all objects in the database. These operations may have resulted from a call to a stored procedure or procedure wrapper, or may have been issued against a table directly. Its columns are the same as those in "ALL_REPPROP" on page 9-28.

DBA_REPRESOL_STATS_CONTROL

The DBA_REPRESOL_STATS_CONTROL view describes statistics collection for conflict resolutions for all replicated tables in the database. Its columns are the same as those in "ALL_REPRESOL_STATS_CONTROL" on page 9-29.

DBA_REPRESOLUTION

The DBA_REPRESOLUTION view indicates the methods used to resolve update, uniqueness, or delete conflicts for each table in the database that is replicated using row-level replication for a given schema. Its columns are the same as those in "ALL_REPRESOLUTION" on page 9-30.

DBA_REPRESOLUTION_METHOD

The DBA_REPRESOLUTION_METHOD view lists all of the conflict resolution methods available in the database. Initially, this view lists the standard methods provided with the advanced replication facility. As you create new user functions and add them as conflict resolution methods for an object in the database, these functions are added to this view. Its columns are the same as those in "ALL_REPRESOLUTION_METHOD" on page 9-32.

DBA_REPRESOLUTION_STATISTICS

The DBA_REPRESOLUTION_STATISTICS view lists information about successfully resolved update, uniqueness, and delete conflicts for all replicated tables in the database. These statistics are only gathered for a table if you have called the DBMS_REPCAT.REGISTER_STATISTICS procedure. The DBA_REPRESOLUTION_STATISTICS view's columns are the same as those in "ALL_REPRESOLUTION_STATISTICS" on page 9-32.

DBA_REPSITES

The DBA_REPSITES view lists the members of each replicated object group in the database.

This view has the following additional columns that are not included in the ALL_REPSITES and USER_REPSITES views:

Column	Datatype	NULL	Description
PROP_UPDATES	NUMBER		Encoding of propagating technique for master.
MY_DBLINK	VARCHAR2(1)		Used to detect problem after import. If Y then the dblink is the global name.

Except for these additional columns, its columns are the same as those in "ALL_REPSITES" on page 9-33.

USER_REPCATLOG

The USER_REPCATLOG view at each master site contains the interim status of any asynchronous administrative requests and any error messages generated. This view contains asynchronous administrative requests and error messages that are owned by the current user. All messages encountered while executing a request are eventually transferred to the USER_REPCATLOG view at the master that originated the request. If an administrative request completes without error, ultimately all traces of this request are removed from the USER_REPCATLOG view. Its columns are the same as those in "ALL_REPCATLOG" on page 9-6.

USER_REPCAT_REFRESH_TEMPLATES

This view contains global information about each deployment template owned by the current user, such as the template name, template owner, what refresh group the template objects belong to, and the type of template (private vs. public).

Its columns are the same as those in ALL_REPCAT_REFRESH_TEMPLATES. For detailed information about this view and its columns, see "ALL_REPCAT_REFRESH_TEMPLATES" on page 9-8.

USER_REPCAT_TEMPLATE_OBJECTS

The USER_REPCAT_TEMPLATE_OBJECTS view contains the individual object definitions that are contained in each deployment template owned by the current user. Individual objects are added to a template by specifying the target template in REFRESH_TEMPLATE_NAME.

Its columns are the same as those in ALL_REPCAT_TEMPLATE_OBJECTS. For detailed information about this view and its columns, see "ALL_REPCAT_TEMPLATE_OBJECTS" on page 9-9.

USER_REPCAT_TEMPLATE_PARMS

Parameters defined in the object DDL for all templates owned by the current user are stored in the USER_REPCAT_TEMPLATE_PARMS table. When an object is added to a template, the DDL is examined for variables; any found parameters are automatically added to this view.

Its columns are the same as those in ALL_REPCAT_TEMPLATE_PARMS. For detailed information about this view and its columns, see "ALL_REPCAT_TEMPLATE_PARMS" on page 9-11.

USER_REPCAT_TEMPLATE_SITES

The USER_REPCAT_TEMPLATE_SITES view provides the user with information about the current status of template instantiation amongst the sites of a enterprise network. This view contains information about instantiation sites for deployment templates that are owned by the current user. Specifically, the user can monitor the installation and deletion of templates at specific sites, including RepAPI sites. Its columns are the same as those in "ALL_REPCAT_TEMPLATE_SITES" on page 9-13.

USER_REPCAT_USER_AUTHORIZATIONS

The USER_REPCAT_USER_AUTHORIZATIONS view lists the authorized users for all of the templates that are owned by the current user and specified for private use. Users listed in this view have the ability to instantiate the specified template. Users not contained in this view cannot instantiate the template. Its columns are the same as those in "ALL_REPCAT_USER_AUTHORIZATIONS" on page 9-14.

USER_REPCAT_USER_PARM_VALUES

The USER_REPCAT_USER_PARM_VALUES view describes the template parameters for all deployment templates owned by the current user. The DBA has the option of building a table of user parameters prior to distributing the template for instantiation. When a template is instantiated by a specified user, the values stored in the USER_REPCAT_USER_PARM_VALUES table for the specified user are used automatically.

Its columns are the same as those in ALL_REPCAT_USER_PARM_VALUES. For detailed information about this view and its columns, see "ALL_REPCAT_USER_PARM_VALUES" on page 9-15.

USER_REPCOLUMN

The USER_REPCOLUMN view lists the replicated columns for all the tables owned by the current user. Its columns are the same as those in "ALL_REPCOLUMN" on page 9-17.

USER_REPCOLUMN_GROUP

The USER_REPCOLUMN_GROUP view lists the column groups for each replicated table owned by the current user. Its columns are the same as those in "ALL_REPCOLUMN_GROUP" on page 9-18.

Note: The SNAME column is not present in the USER_REPCOLUMN_GROUP view. This column is available in the ALL_REPCOLUMN_GROUP and DBA_REPCOLUMN_GROUP views.

USER_REPCONFLICT

The USER_REPCONFLICT view displays the name of each table owned by the current user on which a conflict resolution method has been defined and the type of conflict that the method is used to resolve. Its columns are the same as those in "ALL_REPCONFLICT" on page 9-18.

Note: The SNAME column is not present in the USER_REPCONFLICT view. This column is available in the ALL_REPCONFLICT and DBA_REPCONFLICT views.

USER_REPDDL

The USER_REPDDL contains the DDL for each replication object owned by the current user. Its columns are the same as those in "ALL_REPDDL" on page 9-19.

USER_REPGENOBJECTS

The USER_REPGENOBJECTS view describes each object owned by the current user that was generated to support replication. Its columns are the same as those in "ALL_REPGENOBJECTS" on page 9-20.

USER_REPGROUP

The USER_REPGROUP view describes all of the object groups owned by the current user that are being replicated. The members of each object group are listed in a different view, USER_REPOBJECT. The USER_REPGROUP view's columns are the same as those in "ALL_REPGROUP" on page 9-21.

USER_REPGROUP_PRIVILEGES

The USER_REPGROUP_PRIVILEGES view contains information about the users who are registered for privileges in the object groups owned by the current user. Its columns are the same as those in "ALL_REPGROUP_PRIVILEGES" on page 9-22.

USER_REPGROUPED_COLUMN

The USER_REPGROUPED_COLUMN view lists all of the columns that make up the column groups for each table. Its columns are the same as those in "ALL_REPGROUPED_COLUMN" on page 9-22.

Note: The SNAME column is not present in the USER_REPGROUPED_COLUMN view. This column is available in the ALL_REPGROUPED_COLUMN and DBA_REPGROUPED_COLUMN views.

USER_REPKEY_COLUMNS

The USER_REPKEY_COLUMNS view describes the primary key column(s) in each table owned by the current user. Its columns are the same as those in "ALL_REPKEY_COLUMNS" on page 9-23.

USER_REPOBJECT

The USER_REPOBJECT view contains information about the objects in each replicated object group owned by the current user. An object can belong to only one object group. A replicated object group can span multiple schemas. Its columns are the same as those in "ALL_REPOBJECT" on page 9-23.

USER_REPPARAMETER_COLUMN

In addition to the information contained in the USER_REPRESOLUTION view, the USER_REPPARAMETER_COLUMN view contains information about the columns that are used to resolve conflicts for each replicated table owned by the current user. These are the column values that are passed as the LIST_OF_COLUMN_NAMES argument to the ADD_*conflicttype*_RESOLUTION procedures in the DBMS_REPCAT package. The USER_REPPARAMETER_COLUMN view's columns are the same as those in "ALL_REPPARAMETER_COLUMN" on page 9-25.

Note: The SNAME column is not present in the USER_REPPARAMETER_COLUMN view. This column is available in the ALL_REPPARAMETER_COLUMN and DBA_REPPARAMETER_COLUMN views.

USER_REPPRIORITY

The USER_REPPRIORITY view contains the value and priority level of each priority group member in each priority group owned by the current user. Priority group names must be unique within a replicated object group. Priority levels and values must each be unique within a given priority group. Its columns are the same as those in "ALL_REPPRIORITY" on page 9-26.

Note: The SNAME column is not present in the USER_REPPRIORITY view. This column is available in the ALL_REPPRIORITY and DBA_REPPRIORITY views.

USER_REPPRIORITY_GROUP

The USER_REPPRIORITY_GROUP view describes the priority group or site priority group defined for each replicated object group owned by the current user. Its columns are the same as those in "ALL_REPPRIORITY_GROUP" on page 9-28.

USER_REPPROP

The USER_REPPROP view indicates the technique used to propagate operations on each replicated object to the same object at another master site. This view shows objects owned by the current user. These operations may have resulted from a call to a stored procedure or procedure wrapper, or may have been issued against a table directly. Its columns are the same as those in "ALL_REPPROP" on page 9-28.

USER_REPRESOL_STATS_CONTROL

The USER_REPRESOL_STATS_CONTROL view describes statistics collection for conflict resolutions for all replicated tables owned by the current user. Its columns are the same as those in "ALL_REPRESOL_STATS_CONTROL" on page 9-29.

Note: The SNAME column is not present in the USER_REPRESOL_STATS_CONTROL view. This column is available in the ALL_REPRESOL_STATS_CONTROL and DBA_REPRESOL_STATS_CONTROL views.

USER_REPRESOLUTION

The USER_REPRESOLUTION view indicates the methods used to resolve update, uniqueness, or delete conflicts for each table owned by the current user that is replicated using row-level replication for a given schema. Its columns are the same as those in "ALL_REPRESOLUTION" on page 9-30.

Note: The SNAME column is not present in the USER_REPREPRESOLUTION view. This column is available in the ALL_REPREPRESOLUTION and DBA_REPREPRESOLUTION views.

USER_REPRESOLUTION_METHOD

The USER_REPRESOLUTION_METHOD view lists all of the conflict resolution methods available in the database. Initially, this view lists the standard methods provided with the advanced replication facility. As you create new user functions and add them as conflict resolution methods for an object in the database, these functions are added to this view. Its columns are the same as those in "ALL_REPRESOLUTION_METHOD" on page 9-32.

USER_REPRESOLUTION_STATISTICS

The USER_REPRESOLUTION_STATISTICS view lists information about successfully resolved update, uniqueness, and delete conflicts for all replicated tables owned by the current user. These statistics are only gathered for a table if you have called the DBMS_REPCAT.REGISTER_STATISTICS procedure. The USER_REPRESOLUTION_STATISTICS view's columns are the same as those in "ALL_REPRESOLUTION_STATISTICS" on page 9-32.

Note: The SNAME column is not present in the USER_REPRESOLUTION_STATISTICS view. This column is available in the ALL_REPRESOLUTION_STATISTICS and DBA_REPRESOLUTION_STATISTICS views.

USER_REPSITES

The USER_REPSITES view lists the members of each replicated object group owned by the current user. Its columns are the same as those in "ALL_REPSITES" on page 9-33.

Deferred Transaction Views

Oracle provides several views for you to use in administering deferred transactions. These views provide information about each deferred transaction, such as the transaction destinations, the deferred calls that make up the transactions, and any errors encountered during attempted execution of the transaction.

- DEFCALL
- DEFDEFAULTDEST
- DEFERRCOUNT
- DEFCALLDEST
- DEFERROR
- DEFLOB
- DEFPROPAGATOR
- DEFSCHEDULE
- DEFTRAN
- DEFTRANDEST

Caution: You should not modify the tables directly. Instead, use the procedures provided in the DBMS_DEFER and DBMS_DEFER_SYS packages.

DEFCALL

Records all deferred remote procedure calls.

Column	Datatype	NULL	Description
CALLNO	NUMBER		Unique ID of a call within a transaction.
DEFERRED_TRAN_ID	VARCHAR2(30)		The unique ID of the associated transaction.
SCHEMANAME	VARCHAR2(30)		The schema name of the deferred call.
PACKAGENAME	VARCHAR2(30)		The package name of the deferred call. For a replicated table, this may refer to the table name.
PROCNAME	VARCHAR2(30)		The procedure name of the deferred call. For a replicated table, this may refer to an operation name.
ARGCOUNT	NUMBER		The number of arguments in the deferred call.

DEFCALLDEST

Lists the destinations for each deferred remote procedure call.

Column	Datatype	NULL	Description
CALLNO	NUMBER	NOT NULL	Unique ID of a call within a transaction.
DEFERRED_TRAN_ID	VARCHAR2(30)	NOT NULL	Corresponds to the DEFERRED_TRAN_ID column in the DEFTRAN view. Each deferred transaction is made up of one or more deferred calls.
DBLINK	VARCHAR2(128)	NOT NULL	The fully qualified database name of the destination database.

DEFDEFAULTDEST

If you are not using Oracle replication and do not supply a destination for a deferred transaction or the calls within that transaction, Oracle uses the DEFDEFAULTDEST view to determine the destination databases to which you want to defer a remote procedure call.

Column	Datatype	NULL	Description
DBLINK	VARCHAR2(128)	NOT NULL	The fully qualified database name to which a transaction is replicated.

DEFERRCOUNT

Contains information about the error transactions for a destination.

Column	Datatype	NULL	Description
ERRCOUNT	NUMBER		Number of existing transactions that caused an error for the destination.
DESTINATION	VARCHAR2(128)		Database link used to address destination.

DEFERROR

Contains the ID of each transaction that could not be applied. You can use this ID to locate the queued calls associated with this transaction. These calls are stored in the DEFCALL view. You can use the procedures in the DBMS_DEFER_QUERY package to determine the arguments to the procedures listed in the DEFCALL view.

Column	Datatype	NULL	Description
DEFERRED_TRAN_ID	VARCHAR2(22)	NOT NULL	The ID of the transaction causing the error.
ORIGIN_TRAN_DB	VARCHAR2(128)		The database originating the deferred transaction.
ORIGIN_TRAN_ID	VARCHAR2(22)		The original ID of the transaction.
CALLNO	NUMBER		Unique ID of the call at DEFERRED_TRAN_ID.
DESTINATION	VARCHAR2(128)		Database link used to address destination.
START_TIME	DATE		Time when the original transaction was enqueued.
ERROR_NUMBER	NUMBER		Oracle error number.
ERROR_MSG	VARCHAR2(2000)		Error message text.
RECEIVER	VARCHAR2(30)		Original receiver of the deferred transaction.

DEFLOB

Contains the LOB parameters to deferred RPCs.

Column	Datatype	NULL	Description
ID	RAW(16)	NOT NULL	Identifier of the LOB parameter.
DEFERRED_TRAN_ID	VARCHAR2(22)		Transaction ID for deferred RPC with this LOB parameter.
BLOB_COL	BLOB(4000)		The binary LOB parameter.
CLOB_COL	CLOB(4000)		The character LOB parameter.
NCLOB_COL	CLOB(4000)		The national character LOB parameter.

DEFPROPAGATOR

Contains information about the local propagator.

Column	Datatype	NULL	Description
USERNAME	VARCHAR2(30)	NOT NULL	Username of the propagator.
USERID	NUMBER	NOT NULL	User ID of the propagator.
STATUS	VARCHAR2(7)		Status of the propagator.
CREATED	DATE	NOT NULL	Time when the propagator was registered.

DEFSCHEDULE

Contains information about when a job is next scheduled to be executed.

Column	Datatype	NULL	Description
DBLINK	VARCHAR2(128)	NOT NULL	Fully qualified pathname to the master database site for which you have scheduled periodic execution of deferred remote procedure calls.
JOB	NUMBER		Number assigned to job when you created it by calling DBMS_DEFER_SYS.SCHEDULE_PUSH. Query the WHAT column of the USER_JOBS view to determine what is executed when the job is run.
INTERVAL	VARCHAR2(200)		Function used to calculate the next time to push the deferred transaction queue to destination.
NEXT_DATE	DATE		Next date that job is scheduled to be executed.
LAST_DATE	DATE		Last time the queue was pushed (or attempted to push) remote procedure calls to this destination.
DISABLED	CHAR(1)		If Y then propagation to destination is disabled. If N then propagation to the destination is enabled.
LAST_TXN_COUNT	NUMBER		Number of transactions pushed during last attempt.
LAST_ERROR_NUMBER	NUMBER		Oracle error number from last push.
LAST_ERROR_MESSAGE	VARCHAR2(2000)		Error message from last push.

DEFTRAN

Records all deferred transactions.

Column	Datatype	NULL	Description
DEFERRED_TRAN_ID	VARCHAR2(30)		The transaction ID that enqueued the calls.
DELIVERY_ORDER	NUMBER		An identifier that determines the order of deferred transactions in the queue. The identifier is derived from the system commit number of the originating transaction.
DESTINATION_LIST	VARCHAR2(1)		R indicates that the destinations are determined by the ALL_REPSITES view. D indicates that the destinations were determined by the DEFDEFAULTDEST view or the NODE_LIST argument to the TRANSACTION or CALL procedures.
START_TIME	DATE		The time that the original transaction was enqueued.

DEFTRANDEST

Lists the destinations for a deferred transaction.

Column	Datatype	NULL	Description
DEFERRED_TRAN_ID	VARCHAR2(30)	NOT NULL	The transaction ID of the transaction to replicate to the given database link.
DELIVERY_ORDER	NUMBER		An identifier that determines the order of deferred transactions in the queue. The identifier is derived from the system commit number of the originating transaction.
DBLINK	VARCHAR2(128)	NOT NULL	The fully qualified database name of the destination database.

Snapshots and Snapshot Refresh Group Views

The following views provide information about snapshots and snapshot refresh groups.

Table 9–2 Snapshots and Snapshot Refresh Group Views

ALL_ Views	DBA_ Views	USER_ Views
ALL_REFRESH	DBA_REFRESH	USER_REFRESH
ALL_REFRESH_CHILDREN	DBA_REFRESH_CHILDREN	USER_REFRESH_CHILDREN
	DBA_REGISTERED_SNAPSHOT_ GROUPS	
ALL_REGISTERED_SNAPSHOTS	DBA_REGISTERED_SNAPSHOTS	USER_REGISTERED_SNAPSHOTS
ALL_SNAPSHOT_LOGS	DBA_SNAPSHOT_LOGS	USER_SNAPSHOT_LOGS
	DBA_SNAPSHOT_LOG_FILTER_COLS	
ALL_SNAPSHOT_REFRESH_TIMES	DBA_SNAPSHOT_REFRESH_TIMES	USER_SNAPSHOT_REFRESH_TIMES
ALL_SNAPSHOTS	DBA_SNAPSHOTS	USER_SNAPSHOTS

ALL_REFRESH

Describes all the refresh groups accessible to the current user.

Related views:

- DBA_REFRESH describes all refresh groups in the database.
- USER_REFRESH describes all refresh groups owned by the current user.

Column	Datatype	NULL	Description
ROWNER	VARCHAR2(30)	NOT NULL	Name of the owner of the refresh group.
RNAME	VARCHAR2(30)	NOT NULL	Name of the refresh group.
REFGROUP	NUMBER		Internal identifier of refresh group.
IMPLICIT_DESTROY	VARCHAR2(1)		Y or N; if Y, then destroy the refresh group when its last item is subtracted.
PUSH_DEFERRED_RPC	VARCHAR2(1)		Y or N; if Y then push changes from snapshot to master before refresh.
REFRESH_AFTER _ERRORS	VARCHAR2(1)		If Y, proceed with refresh despite error when pushing deferred RPCs.
ROLLBACK_SEG	VARCHAR2(30)		Name of the rollback segment to use while refreshing.
JOB	NUMBER		Identifier of job used to refresh the group automatically.
NEXT_DATE	DATE		Date that this job will next be refreshed automatically, if not broken.
INTERVAL	VARCHAR2(200)		A date function used to compute the next NEXT_DATE.
BROKEN	VARCHAR2(1)		Y or N; Y means the job is broken and will never be run.
PURGE_OPTION	NUMBER(38)		The method for purging the transaction queue after each push. 1=quick purge option; 2=precise purge option.
PARALLELISM	NUMBER(38)		The level of parallelism for transaction propagation.
HEAP_SIZE	NUMBER(38)		The size of the heap.

ALL_REFRESH_CHILDREN

Lists all the objects in refresh groups that are accessible to the current user.

Related views:

- DBA_REFRESH_CHILDREN describes the objects in all refresh groups in the database.
- USER_REFRESH_CHILDREN describes the objects in all refresh groups owned by the current user.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the object in the refresh group.
NAME	VARCHAR2(30)	NOT NULL	Name of the object in the refresh group.
TYPE	VARCHAR2(30)		Type of the object in the refresh group.
ROWNER	VARCHAR2(30)	NOT NULL	Name of the owner of the refresh group.
RNAME	VARCHAR2(30)	NOT NULL	Name of the refresh group.
REFGROUP	NUMBER		Internal identifier of refresh group.
IMPLICIT_DESTROY	VARCHAR2(1)		Y or N; if Y, then destroy the refresh group when its last item is subtracted.
PUSH_DEFERRED_RPC	VARCHAR2(1)		Y or N; if Y then push changes from snapshot to master before refresh.
REFRESH_AFTER_ERRORS	VARCHAR2(1)		If Y, proceed with refresh despite error when pushing deferred RPCs.
ROLLBACK_SEG	VARCHAR2(30)		Name of the rollback segment to use while refreshing.
JOB	NUMBER		Identifier of job used to refresh the group automatically.
NEXT_DATE	DATE		Date that this job will next be refreshed automatically, if not broken.
INTERVAL	VARCHAR2(200)		A date function used to compute the next NEXT_DATE.
BROKEN	VARCHAR2(1)		Y or N; Y means the job is broken and will never be run.
PURGE_OPTION	NUMBER(38)		The method for purging the transaction queue after each push. 1=quick purge option; 2=precise purge option.
PARALLELISM	NUMBER(38)		The level of parallelism for transaction propagation.
HEAP_SIZE	NUMBER(38)		The size of the heap.

ALL_REGISTERED_SNAPSHOTS

Describes all registered snapshots accessible to the current user.

Related views:

- DBA_REGISTERED_SNAPSHOTS describes all registered snapshots in the database.
- USER_REGISTERED_SNAPSHOTS describes all registered snapshots owned by the current user.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the snapshot.
NAME	VARCHAR2(30)	NOT NULL	Name of the snapshot.
SNAPSHOT_SITE	VARCHAR2(128)	NOT NULL	Global name of the snapshot site.
CAN_USE_LOG	VARCHAR2(3)		YES if this snapshot can use a snapshot log, NO if this snapshot is too complex to use a log.
UPDATABLE	VARCHAR2(3)		Specifies whether the snapshot is updatable. YES if it is, NO if it is not. If set to NO, the snapshot is read only.
REFRESH_METHOD	VARCHAR2(11)		Whether the snapshot uses rowids or primary key for fast refresh.
SNAPSHOT_ID	NUMBER(38)		Identifier for the snapshot used by the master for fast refresh.
VERSION	VARCHAR2(17)		Version of snapshot.
QUERY_TXT	LONG		Original query of which this snapshot is an instantiation.

ALL_SNAPSHOT_LOGS

Lists all snapshot logs accessible to the current user.

This view shows one row for each snapshot using the snapshot log. However, there is only one log for all the snapshots located at the master site. To find out which logs are used, query USER_SNAPSHOT_LOGS using unique without selecting SNAPSHOT_ID and CURRENT_SNAPSHOTS.

Related views:

- DBA_SNAPSHOT_LOGS describes all snapshot logs in the database.
- USER_SNAPSHOT_LOGS describes all snapshot logs owned by the current user.

Column	Datatype	NULL	Description
LOG_OWNER	VARCHAR2(30)	NOT NULL	Owner of the log
MASTER	VARCHAR2(30)	NOT NULL	Name of the master table whose changes are logged.
LOG_TABLE	VARCHAR2(30)	NOT NULL	Name of the table where the changes to the master table are recorded.
LOG_TRIGGER	VARCHAR2(30)		Obsolete with the release of Oracle8*i* Server. Set to NULL. Formerly, this parameter was an after-row trigger on the master which inserted rows into the log.
ROWIDS	VARCHAR2(3)		If YES, records ROWID information.
PRIMARY_KEY	VARCHAR2(3)		If YES, records primary key information.
FILTER_COLUMNS	VARCHAR2(3)		If YES, snapshot log records filter columns.
CURRENT_SNAPSHOTS	DATE		One date per snapshot; the date the snapshot of the master was last refreshed.
SNAPSHOT_ID	NUMBER(38)		Unique identifier of the snapshot.

ALL_SNAPSHOT_REFRESH_TIMES

Describes refresh times of snapshots accessible to the current owner.

Related views:

- DBA_SNAPSHOT_REFRESH_TIMES describes refresh times of all snapshots in the database.
- USER_SNAPSHOT_REFRESH_TIMES describes refresh times of all snapshots owned by the current user.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the snapshot.
NAME	VARCHAR2(30)	NOT NULL	Name of the snapshot view.
MASTER_OWNER	VARCHAR2(30)		Owner of the master tabl.e
MASTER	VARCHAR2(30)		Name of the master tabl.e
LAST_REFRESH	DATE		The last refresh.

ALL_SNAPSHOTS

Describes all snapshots accessible to the user.

Related views:

- DBA_SNAPSHOTS describes all snapshots in the database.
- USER_SNAPSHOTS describes all snapshots owned by the current user.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the snapshot.
NAME	VARCHAR2(30)	NOT NULL	Name of the view used by users and applications for viewing the snapshot. For databases with the COMPATIBLE parameter set to 8.1.0 or higher, the name of the view is the same as the TABLE_NAME. For databases with the COMPATIBLE parameter set below 8.1.0, the name of the view is different than the TABLE_NAME.
TABLE_NAME	VARCHAR2(30)	NOT NULL	Table the in which the snapshot is stored.
MASTER_VIEW	VARCHAR2(30)		View of the master table, owned by the snapshot owner, used for refreshes. This is obsolete in Oracle8*i* and is set to NULL.
MASTER_OWNER	VARCHAR2(30)		Owner of the master table.
MASTER	VARCHAR2(30)		Name of the master table of which this snapshot is a copy.
MASTER_LINK	VARCHAR2(128)		Database link name to the master site.
CAN_USE_LOG	VARCHAR2(3)		YES if this snapshot can use a snapshot log, NO if this snapshot is too complex to use a log.
UPDATABLE	VARCHAR2(3)		Specifies whether the snapshot is updatable. YES if it is, NO if it is not. If set to YES, the snapshot is read only.
REFRESH_METHOD	VARCHAR2(11)		Values used to drive a refresh of the snapshot (PRIMARY KEY / ROWID / COMPLEX). If PRIMARY KEY, then the snapshot uses primary keys to drive a fast refresh. If ROWID, then it uses RowIDs to drive a fast refresh. If COMPLEX, then fast refreshes are not allowed and the snapshot can only perform complete refreshes.
LAST_REFRESH	DATE		Date and time at the master site of the last refresh.
ERROR	NUMBER		The number of failed automatic refreshes since last successful refresh.

Column	Datatype	NULL	Description
FR_OPERATIONS	VARCHAR2(10)		Status of generated fast refresh operations: (REGENERATE, VALID).
CR_OPERATIONS	VARCHAR2(10)		Status of generated complete refresh operations: (REGENERATE, VALID).
TYPE	VARCHAR2(8)		Type of refresh for all automatic refreshes: COMPLETE, FAST, FORCE.
NEXT	VARCHAR2(200)		Date function used to compute next refresh dates.
START_WITH	DATE		Date function used to compute next refresh dates.
REFRESH_GROUP	NUMBER		All snapshots in a given refresh group get refreshed in the same transaction.
UPDATE_TRIG	VARCHAR2(30)		Obsolete. It is NULL for Oracle8*i* snapshots. Formerly, the name of the trigger that fills the UPDATE_LOG.
UPDATE_LOG	VARCHAR2(30)		The table that logs changes made to an updatable snapshots.
QUERY	LONG		Original query of which this snapshot is an instantiation.
MASTER_ROLLBACK_SEG	VARCHAR2(30)		Rollback segment to use at the master site.
STATUS	VARCHAR2(7)		The status of the contents of the snapshot.
REFRESH_MODE	VARCHAR2(8)		This indicates how and when the snapshot will be refreshed.
PREBUILT	VARCHAR2(3)		If YES, this snapshot uses a prebuilt table as the base table.

DBA_REFRESH

The DBA_REFRESH view describes all refresh groups in the database. Its columns are the same as those in "ALL_REFRESH" on page 9-54.

DBA_REFRESH_CHILDREN

The DBA_REFRESH_CHILDREN lists all of the objects in all refresh groups in the database. Its columns are the same as those in "ALL_REFRESH_CHILDREN" on page 9-55.

DBA_REGISTERED_SNAPSHOT_GROUPS

DBA_REGISTERED_SNAPSHOT_GROUPS lists all the snapshot groups at this site. This view is available only if you have installed the Oracle replication packages.

Column	Datatype	NULL	Description
NAME	VARCHAR2(30)		Name of the snapshot replication group.
SNAPSHOT_SITE	VARCHAR2(128)		Site of the master of the snapshot repgroup.
GROUP_COMMENT	VARCHAR2(80)		Description of the snapshot repgroup.
VERSION	VARCHAR2(8)		Version of the snapshot repgroup.
FNAME	VARCHAR2(30)		Name of the flavor of the snapshot object group.

DBA_REGISTERED_SNAPSHOTS

DBA_REGISTERED_SNAPSHOTS describes all registered snapshots in the database. Its columns are the same as those in "ALL_REFRESH" on page 9-54.

DBA_SNAPSHOT_LOGS

DBA_SNAPSHOT_LOGS describes all snapshot logs in the database. Its columns are the same as those in "ALL_SNAPSHOT_LOGS" on page 9-57.

DBA_SNAPSHOT_LOG_FILTER_COLS

DBA_SNAPSHOT_LOG_FILTER_COLS lists all filter columns (excluding primary key columns) being logged in the snapshot logs.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the master table being logged.
NAME	VARCHAR2(30)	NOT NULL	Name of the master table being logged.
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Filter column being logged.

DBA_SNAPSHOT_REFRESH_TIMES

DBA_SNAPSHOT_REFRESH_TIMES lists refresh times of all snapshots in the database. Its columns are the same as those in "ALL_SNAPSHOT_REFRESH_TIMES" on page 9-58.

DBA_SNAPSHOTS

DBA_SNAPSHOTS describes all snapshots in the database. Its columns are the same as those in "ALL_SNAPSHOTS" on page 9-59.

USER_REFRESH

USER_REFRESH describes all refresh groups owned by the current user. Its columns are the same as those in "ALL_REFRESH" on page 9-54.

USER_REFRESH_CHILDREN

USER_REFRESH_CHILDREN lists all the objects in refresh groups owned by the current user. Its columns are the same as those in "ALL_REFRESH_CHILDREN" on page 9-55.

USER_REGISTERED_SNAPSHOTS

USER_REGISTERED_SNAPSHOTS describes all registered snapshots owned by the current user. Its columns are the same as those in "ALL_REFRESH" on page 9-54.

USER_SNAPSHOTS

USER_SNAPSHOTS describes all snapshots owned by the current user. Its columns are the same as those in "ALL_SNAPSHOTS" on page 9-59.

USER_SNAPSHOT_LOGS

USER_SNAPSHOT_LOGS lists all snapshot logs owned by the current user. Its columns are the same as those in "ALL_SNAPSHOT_LOGS" on page 9-57.

USER_SNAPSHOT_REFRESH_TIMES

USER_SNAPSHOT_REFRESH_TIMES describes refresh times of snapshots owned by the current user. Its columns are the same as those in "ALL_SNAPSHOT_REFRESH_TIMES" on page 9-58.

A

Security Options

Security options include the following:

- Security Setup for Multimaster Replication
- Security Setup for Snapshot Replication

Security Setup for Multimaster Replication

Nearly all users should find it easiest to use the Replication Manager Setup Wizard when configuring multimaster replication security. However, for certain cases you may need to use the replication management API to perform these setup operations.

To configure a replication environment, the database administrator must connect with DBA privileges to grant the necessary privileges to the replication administrator.

First set up user accounts at each master site with the appropriate privileges to configure and maintain the replication environment and to propagate and apply replicated changes. You must also define links for users at each master site.

In addition to the end users who access replicated objects, there are three special categories of "users" in a replication environment:

- Replication administrators, who are responsible for configuring and maintaining a replication environment.
- Propagators, who are responsible for propagating deferred transactions.
- Receivers at remote sites, who are responsible for applying these transactions.

Typically, a single user acts as administrator, propagator, and receiver. However, you can have separate users perform each of these functions. You can choose to have a single, global replication administrator or, if your replication groups do not span schema boundaries, you may prefer to have separate replication administrators for different schemas. Note, however, that you can have only one registered propagator for each database.

Table A–1 on page A-4 describes the necessary privileges that must be assigned to these specialized accounts. Most privileges needed by these users are granted to them through calls to the replication management API. You also must grant certain privileges directly, such as CONNECT and RESOURCE privileges.

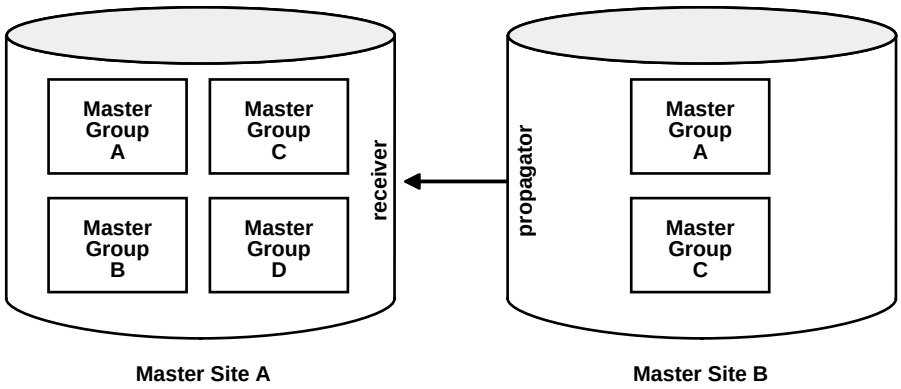
Trusted vs. Untrusted Security

In addition to the different users, you also need to determine which type of security model you will implement: trusted or untrusted. With a trusted security model, the receiver has access to all local master groups. Because the receiver performs database activities at the local master site on behalf of the propagator at the remote site, the propagator also has access to all master groups at the receiver’s site. Remember that a single receiver is used for all incoming transactions.

For example, consider the scenario in Figure A–1. Even though only Master Groups A and C exist at Master Site B, the propagator has access to Master Groups A, B, C, and D at Master Site A because the trusted security model has been used. While this greatly increases the flexibility of database administration, due to the mobility of remote database administration, it also increases the chances of a malicious user at a remote site viewing or corrupting data at the master site.

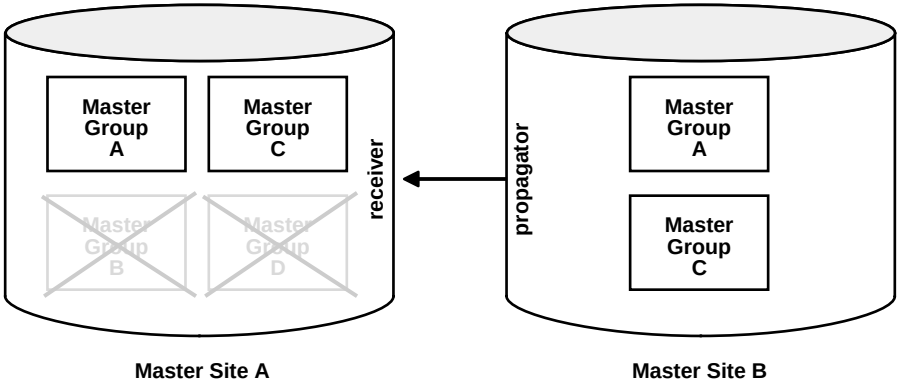
Regardless of the security model used, Oracle8*i* automatically grants the appropriate privileges for objects as they are added to or removed from a replicated environment.

Figure A–1 Trusted Security: Multimaster Replication



Untrusted security assigns only the privileges to the receiver that are required to work with specified master groups. The propagator, therefore, can only access the specified master groups that are local to the receiver. Figure A–2 illustrates an untrusted security model. Because Master Site B contains only Master Groups A and C, the receiver at Master Site A has been granted privileges for Master Groups A and C only, thereby limiting the propagator's access at Master Site A.

Figure A–2 Untrusted Security: Multimaster Replication



Typically, master sites are considered trusted and therefore the trusted security model is used. If, however, your remote master sites are untrusted, you may want to use the untrusted model and assign your receiver limited privileges. A site might be considered untrusted, for example, if a consulting shop performs work for multiple customers. Use the appropriate API call listed for the receiver in Table A–1 to assign the different users the appropriate privileges.

Table A–1 Required User Accounts

User	Privileges
global replication administrator	DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA
schema-level replication administrator	DBMS_REPCAT_ADMIN.GRANT_ADMIN_SCHEMA
propagator	DBMS_DEFER_SYS.REGISTER_PROPAGATOR

receiver	See the DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP procedure on page 8-157 for details.
	Trusted:
	`DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP`
	`privilege => 'receiver'`
	`list_of_gnames => NULL`
	Untrusted:
	`DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP`
	`privilege => 'receiver'`
	`list_of_gnames => 'mastergroupname'`

After you have created these accounts and assigned the appropriate privileges, create the following private database links, including username and password between each site:

- From the local replication administrator to the remote replication administrator.
- From the local propagator to the remote receiver.

Assuming you have designated a single user account to act as replication administrator, propagator, and receiver, you must create N(N-1) links, where N is the number of master sites in your replication environment.

After creating these links, you must call DBMS_DEFER_SYS.SCHEDULE_PUSH and DBMS_DEFER_SYS.SCHEDULE_PURGE, at each location, to define how frequently you want to propagate your deferred transaction queue to each remote location, and how frequently you wish to purge this queue. You must call DBMS_DEFER_SYS.SCHEDULE_PUSH multiple times at each site, once for each remote location.

A sample script for setting up multimaster replication between HQ.WORLD and SALES.WORLD is shown below:

```
/*--- Create global replication administrator at HQ ---*/
connect system/manager@hq.world
create user repadmin identified by repadmin
execute dbms_repcat_admin.grant_admin_any_schema(username => 'repadmin')

/*--- Create global replication administrator at Sales ---*/
connect system/manager@sales.world
create user repadmin identified by repadmin
execute dbms_repcat_admin.grant_admin_any_schema(username => 'repadmin')

/*--- Create single user to act as both propagator and receiver at HQ ---*/
connect system/manager@hq.world
create user prop_rec identified by prop_rec
/*--- Grant privileges necessary to act as propagator ---*/
execute dbms_defer_sys.register_propagator(username => 'prop_rec')
/*--- Grant privileges necessary to act as receiver ---*/
execute dbms_repcat_admin.register_user_repgroup(
        username => 'prop_rec',
        privilege_type => 'receiver',
        list_of_gnames => NULL)

/*--- Create single user to act as both propagator and receiver at Sales ---*/
connect system/manager@sales.world
create user prop_rec identified by prop_rec
```

```
/*--- Grant privileges necessary to act as propagator ---*/execute
dbms_defer_sys.register_propagator(username => 'prop_rec')
/*--- Grant privileges necessary to act as receiver ---*/
execute dbms_repcat_admin.register_user_repgroup(
        username => 'prop_rec',
        privilege_type => 'receiver',
        list_of_gnames => NULL)

/*--- Create public link from HQ to Sales with necessary USING clause ---*/
connect system/manager@hq.world
create public database link sales.world using sales.world

/*--- Create private repadmin to repadmin link ---*/
connect repadmin/repadmin@hq.world
create database link sales.world connect to repadmin identified by repadmin

/*--- Schedule replication from HQ to Sales ---*/
execute dbms_defer_sys.schedule_push(
     destination => 'sales.world',
     interval => 'sysdate + 1/24',
     next_date => sysdate,
     stop_on_error => FALSE,
     parallelism => 1)

/*--- Schedule purge of def tran queue at HQ ---*/
execute dbms_defer_sys.schedule_purge(
     next_date => sysdate,
     interval => 'sysdate + 1',
     delay_seconds => 0,
     rollback_segment => '')

/*--- Create link from propagator to receiver for scheduled push ---*/
connect prop_rec/prop_rec@hq.world
create database link sales.world connect to prop_rec identified by prop_rec

/*--- Create public link from Sales to HQ with necessary USING clause ---*/
connect system/manager@sales.world
create public database link hq.world using hq.world

/*--- Create private repadmin to repadmin link ---*/
connect repadmin/repadmin@sales.world
create database link hq.world connect to repadmin identified by repadmin
```

```
/*--- Schedule replication from Sales to HQ ---*/
execute dbms_defer_sys.schedule_push(
     destination => 'hq.world',
     interval => 'sysdate + 1/24',
     next_date => sysdate,
     stop_on_error => FALSE,
     parallelism => 1)

/*--- Schedule purge of def tran queue at Sales ---*/
execute dbms_defer_sys.schedule_purge(
     next_date => sysdate,
     interval => 'sysdate + 1',
     delay_seconds => 0,
     rollback_segment =>'')

/*--- Create link from propagator to receiver for scheduled push ---*/
connect prop_rec/prop_rec@sales.world
create database link hq.world connect to prop_rec identified by prop_rec
```

Security Setup for Snapshot Replication

Nearly all users should find it easiest to use the Replication Manager Setup Wizard when configuring snapshot replication security. However, for certain specialized cases, you may need to use the replication management API to perform these setup operations. To configure a replication environment, the database administrator must connect with DBA privileges to grant the necessary privileges to the replication administrator.

First set up user accounts at each snapshot site with the appropriate privileges to configure and maintain the replication environment and to propagate replicated changes. You must also define links for these users to the associated master site. You may need to create additional users, or assign additional privileges to users at the associated master site.

In addition to end users who will be accessing replicated objects, there are three special categories of "users" at a snapshot site:

- Replication administrators, who are responsible for configuring and maintaining a replication environment.
- Propagators, who are responsible for propagating deferred transactions.
- Refreshers, who are responsible for pulling down changes to the snapshots from the associated master tables.

Typically, a single user performs each of these functions. However, there may be situations where you need different users performing these functions. For example, snapshots may be created by a snapshot site administrator and refreshed by another end user.

Table A–2 describes the privileges needed to create and maintain a snapshot site.

Table A–2 Required Snapshot Site User Accounts

User	Privileges
snapshot site replication administrator	DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA
propagator	DBMS_DEFER_SYS.REGISTER_PROPAGATOR
refresher	CREATE ANY SNAPSHOT ALTER ANY SNAPSHOT

In addition to creating the appropriate users at the snapshot site, you may need to create additional users at the associated master site, as well. Table A–5 on on page A-11 describes the privileges need by master site users to support a new snapshot site.

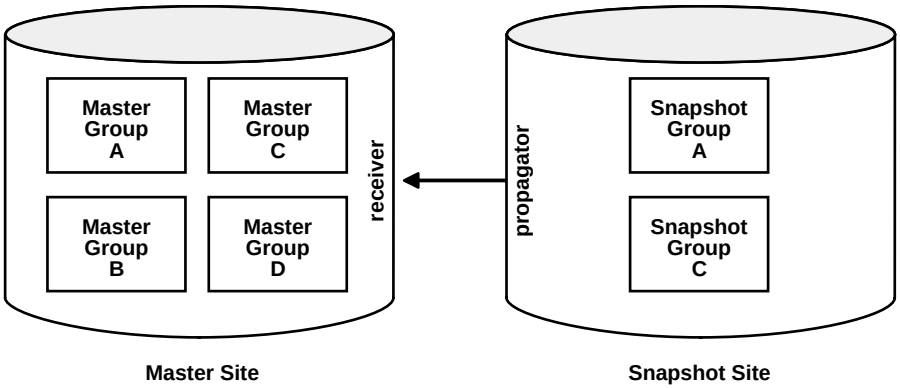
Trusted vs. Untrusted Security

In addition to the different users at the master site, you also need to determine which type of security model you will implement: trusted or untrusted. With a trusted security model, the receiver and proxy snapshot administrator have access to all local master groups. Because the proxy snapshot administrator and receiver perform database activities at the local master site on behalf of the snapshot administrator and propagator, respectively, at the remote snapshot site, the propagator and snapshot administrator also have access to all master groups at the master site. Remember that a single receiver is used for all incoming transactions.

For example, consider the scenario in Figure A–3. Even though Snapshot Groups A and C exist at the snapshot site (based on Master Groups A and C at the Master Site), the propagator and snapshot administrator have access to Master Groups A, B, C, and D at the Master Site because the trusted security model has been used. While this greatly increases the flexibility of database administration, because the DBA can perform administrative functions at any of these remote sites and have these changes propagated to the master sites, it also increases the chances of a malicious user at a remote site viewing or corrupting data at the master site.

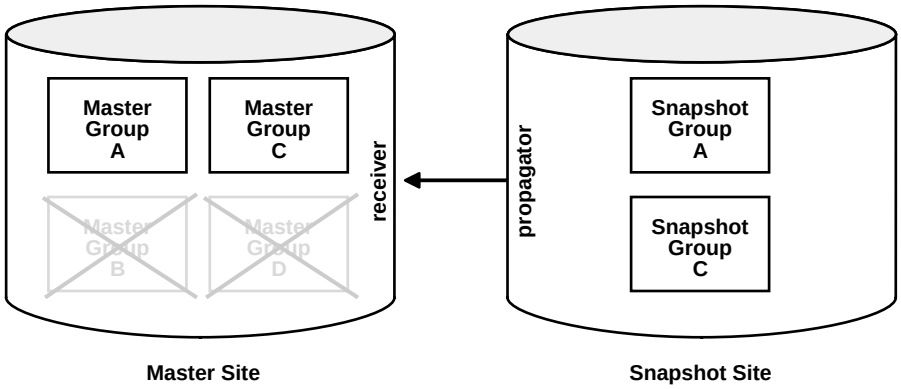
Regardless of the security model used, Oracle8*i* automatically grants the appropriate privileges for objects as they are added to or removed from a replicated environment.

Table A–3 Trusted Security: Snapshot Replication



Untrusted security assigns only the privileges to the proxy snapshot administrator and receiver that are required to work with specified master groups. The propagator and snapshot administrator, therefore, can only access these specified master groups at the Master Site. Figure A–4 illustrates an untrusted security model with snapshot replication. Because the Snapshot Site contains Snapshot Groups A and C, access to only Master Groups A and C are required. Using untrusted security does not allow the propagator or the snapshot administrator at the Snapshot Site to access Master Groups B and D at the Master Site.

Table A–4 Trusted Security: Snapshot Replication



Typically, snapshot sites are more vulnerable to security breaches and therefore the untrusted security model is used. There are very few reasons why you would want to use a trusted security model with your snapshot site and it is recommended that you use the untrusted security model with snapshot sites.

One reason you might choose to use a trusted security model is if your snapshot site is considered a master site in every way (security, constant network connectivity, resources) but is a snapshot only because of data partitioning requirements. Remember that horizontal or vertical partitioning are not supported in a multimaster configuration.

Use the appropriate API calls listed for the proxy snapshot administrator and receiver in Table A–5 to assign the different users the appropriate privileges.

Table A–5 Required Master Site User Accounts

User	Privileges

proxy snapshot site administrator	See the DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP procedure on page 8-157 for details.
	Trusted:
	DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP
	`privilege => 'proxy_snapadmin'`
	`list_of_gnames => NULL`
	Untrusted:
	DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP
	`privilege => 'proxy_snapadmin'`
	`list_of_gnames => 'mastergroupname'`

receiver	See the DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP procedure on page 8-157 for details.
	Trusted:
	DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP
	`privilege => 'receiver'`
	`list_of_gnames => NULL`
	Untrusted:
	DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP
	`privilege => 'receiver'`
	`list_of_gnames => 'mastergroupname'`

proxy refresher	**Trusted**: grant CREATE SESSION grant SELECT ANY TABLE **Untrusted**: grant CREATE SESSION grant SELECT on necessary master tables and snapshot logs

After creating the accounts at both the snapshot and associated master sites, you need to create the following private database links, including username and password, from the snapshot site to the master:

- From the snapshot replication administrator to the proxy snapshot replication administrator.
- From the propagator to the receiver.
- From the refresher to the proxy refresher.
- From the snapshot owner to the master site for refreshes.

Assuming you have designated a single user account to act as snapshot administrator, propagator, and refresher, you must create one link for each snapshot site for those functions. You do not need a link from the master site to the snapshot site.

After creating these links, you must call DBMS_DEFER_SYS.SCHEDULE_PUSH and DBMS_DEFER_SYS.SCHEDULE_PURGE at the snapshot site to define how frequently you want to propagate your deferred transaction queue to the associated master site, and how frequently you wish to purge this queue. You must also call DMBS_REFRESH. REFRESH at the snapshot site to schedule how frequently to pull changes from the associated master site.

B

User-Defined Conflict Resolution Methods

This appendix describes how to build a user-defined conflict resolution and notification functions:

- User-Defined Conflict Resolution Methods
- User-Defined Conflict Notification Methods
- Viewing Conflict Resolution Information

User-Defined Conflict Resolution Methods

Oracle allows you to write your own conflict resolution or notification methods. A user-defined conflict resolution method is a PL/SQL function that returns either TRUE or FALSE. TRUE indicates that the method has successfully resolved all conflicting modifications for a column group. If the method cannot successfully resolve a conflict, it should return FALSE. Oracle continues to evaluate available conflict resolution methods, in sequence order, until either a method returns TRUE or there are no more methods available.

If the conflict resolution method raises an exception, Oracle stops evaluation of the method, and, if any other methods were provided to resolve the conflict with a later sequence number, Oracle does not evaluate them.

Conflict Resolution Method Parameters

The parameters needed by a user-defined conflict resolution method are determined by the type of conflict being resolved (uniqueness, update, or delete) and the columns of the table being replicated. All conflict resolution methods take some combination of old, new, and current column values for the table.

- The old value represents the value of the row at the initiating site before you made the change.
- The new value represents the value of the row at the initiating site after you made the change.
- The current value represents the value of the equivalent row at the receiving site.

> **Note:** Recall that Oracle uses the primary key, or the key specified by SET_COLUMNS, to determine which rows to compare.

The conflict resolution function should accept as parameters the values for the columns specified in the PARAMETER_COLUMN_NAME argument to the DBMS_REPCAT.ADD_*conflicttype*_RESOLUTION procedures. The column parameters are passed to the conflict resolution method in the order listed in the PARAMETER_COLUMN_NAME argument, or in ascending alphabetical order if you specified '*' for this argument. When both old and new column values are passed as parameters (for update conflicts), the old value of the column immediately precedes the new value.

Note: Type checking of parameter columns in user-defined conflict resolution methods is not performed until you regenerate replication support for the associated replicated table.

Resolving Update Conflicts

For update conflicts, a user-defined function should accept the following values for each column in the column group:

- Old column value from the initiating site. The mode for this parameter is IN. This value should not be changed.
- New column value from the initiating site. The mode for this parameter is IN OUT. If the function can resolve the conflict successfully, it should modify the new column value as needed.
- Current column value from the receiving site. The mode for this parameter is IN.

The old, new, and current values for a column are received consecutively. The final argument to the conflict resolution method should be a Boolean flag. If this flag is FALSE, it indicates that you have updated the value of the IN OUT parameter, new, and that you should update the current column value with this new value. If this flag is TRUE, it indicates that the current column value should not be changed.

Resolving Uniqueness Conflicts

Uniqueness conflicts can occur as the result of an INSERT or UPDATE. Your uniqueness conflict resolution method should accept the new column value from the initiating site in IN OUT mode for each column in the column group. The final parameter to the conflict resolution method should be a Boolean flag.

If the method can resolve the conflict, it should modify the new column values so that Oracle can insert or update the current row with the new column values. Your function should set the Boolean flag to TRUE if it wants to discard the new column values, and FALSE otherwise.

Because a conflict resolution method cannot guarantee convergence for uniqueness conflicts, a user-defined uniqueness resolution method should include a notification mechanism.

Resolving Delete Conflicts

Delete conflicts occur when you successfully delete from the local site, but the associated row cannot be found at the remote site (for example, because it had been updated). For delete conflicts, the function should accept old column values in IN OUT mode for the entire row. The final parameter to the conflict resolution method should be a BOOLEAN flag.

If the conflict resolution method can resolve the conflict, it modifies the old column values so that Oracle can delete the current row that matches all old column values. Your function should set the Boolean flag to TRUE if it wants to discard these column values, and FALSE otherwise.

If you perform a delete at the local site and an update at the remote site, the remote site detects the delete conflict, but the local site detects an unresolvable update conflict. This type of conflict cannot be handled automatically. The conflict raises a NO_DATA_FOUND exception and Oracle logs the transaction as an error transaction.

Designing a mechanism to properly handle these types of update / delete conflicts is difficult. It is far easier to avoid these types of conflicts entirely, by simply "marking" deleted rows, and then purging them using procedural replication.

See Also: "Avoiding Delete Conflicts" on page 7-20.

Restrictions

You should avoid the following types of SQL statements in user-defined conflict resolution methods. Use of such statements can result in unpredictable results.

- Data definition language (DDL) statements (such as CREATE, ALTER, DROP)
- Transaction control statements (such as COMMIT, ROLLBACK)
- Session control (such as ALTER SESSION)
- System control (such as ALTER SYSTEM)

Example User-Defined Conflict Resolution Method

The following examples show user-defined methods that are variations on the standard MAXIMUM and ADDITIVE prebuilt conflict resolution methods. Unlike the standard methods, these custom functions can handle nulls in the columns used to resolve the conflict.

Maximum User Function

```
-- User function similar to MAXIMUM method.
-- If curr is null or curr < new, use new values.
-- If new is null or new < curr, use current values.
-- If both are null, no resolution.
-- Does not converge with > 2 masters, unless
-- always increasing.

FUNCTION max_null_loses(old                 IN    NUMBER,
                        new                 IN  OUT  NUMBER,
                        cur                 IN    NUMBER,
                        ignore_discard_flag OUT    BOOLEAN)
  RETURN BOOLEAN IS
BEGIN
   IF (new IS NULL AND cur IS NULL) OR new = cur THEN
       RETURN FALSE;
   END IF;
   IF new IS NULL THEN
       ignore_discard_flag := TRUE;
   ELSIF cur IS NULL THEN
       ignore_discard_flag := FALSE;
   ELSIF new < cur THEN
       ignore_discard_flag := TRUE;
   ELSE
       ignore_discard_flag := FALSE;
   END IF;
   RETURN TRUE;
END max_null_loses;
```

Additive User Function

```
-- User function similar to ADDITIVE method.
-- If old is null, old = 0.
-- If new is null, new = 0.
-- If curr is null, curr = 0.
-- new = curr + (new - old) -> just like ADDITIVE method.

FUNCTION additive_nulls(old                 IN    NUMBER,
                        new                 IN  OUT  NUMBER,
                        cur                 IN    NUMBER,
                        ignore_discard_flag OUT    BOOLEAN)
   RETURN BOOLEAN IS
   old_val NUMBER := 0.0;
   new_val NUMBER := 0.0;
   cur_val NUMBER := 0.0;
```

```
BEGIN
   IF old IS NOT NULL THEN
      old_val := old;
   END IF;
   IF new IS NOT NULL THEN
      new_val := new;
   END IF;
   IF cur IS NOT NULL THEN
      cur_val := cur;
   END IF;
   new := cur_val + (new_val - old_val);
   ignore_discard_flag := FALSE;
   RETURN TRUE;
END additive_nulls;
```

User-Defined Conflict Notification Methods

A conflict notification method is a user-defined function that provides conflict notification rather than or in addition to conflict resolution. For example, you can write your own conflict notification methods to log conflict information in a database table, send an email message, or page an administrator. After you write a conflict notification method, you can assign it to a column group (or constraint) in a specific order so that Oracle notifies you when a conflict happens, before attempting subsequent conflict resolution methods, or after Oracle attempts to resolve a conflict but cannot do so.

To configure a replicated table with a user-defined conflict notification mechanism, you must complete the following steps:

1. Create a conflict notification log.
2. Create the user-defined conflict notification method in a package.

The following sections explain each step.

Creating a Conflict Notification Log

When configuring a replicated table to use a user-defined conflict notification method, the first step is to create a database table that can record conflict notifications. You can create a table to log conflict notifications for one or many tables in a master group.

To create a conflict notification log table at all master sites, use the replication execute DDL facility. For more information, see the "EXECUTE_DDL procedure" of the DBMS_REPCAT package on page 8-129. Do *not* generate replication support for the conflict notification tables because their entries are specific to the site that detects a conflict.

Sample Conflict Notification Log Table

The following CREATE TABLE statement creates a table that you can use to log conflict notifications from several tables in a master group.

```
CREATE TABLE conf_report (
 line          NUMBER(2),    --- used to order message text
 txt           VARCHAR2(80), --- conflict notification message
 timestamp     DATE,         --- time of conflict
 table_name    VARCHAR2(30), --- table in which the
                             --- conflict occurred
 table_owner   VARCHAR2(30), --- owner of the table
 conflict_type VARCHAR2(6)   --- INSERT, DELETE or UNIQUE
)
```

Creating a Conflict Notification Package

To create a conflict notification method, you must define the method in a PL/SQL package and then replicate the package as part of a master group along with the associated replicated table.

A conflict notification method can perform conflict notification only, or both conflict notification and resolution. If possible, you should always use one of Oracle's prebuilt conflict resolution methods to resolve conflicts. When a user-defined conflict notification method performs only conflict notification, assign the user-defined method to a column group (or constraint) along with conflict resolution methods that can resolve conflicts.

Note: If Oracle cannot ultimately resolve a replication conflict, Oracle rolls back the entire transaction, including any updates to a notification table. If notification is necessary independent of transactions, you can design a notification mechanism to use the Oracle DBMS_PIPES package or the database interface to Oracle Office.

Sample Conflict Notification Package

The following package and package body perform a simple form of *conflict notification* by logging uniqueness conflicts for a CUSTOMERS table into the previously defined CONF_REPORT table.

> **Note:** This example of *conflict notification* does not resolve any conflicts. You should either provide a method to resolve conflicts (such as *discard* or *overwrite*), or provide a notification mechanism that will succeed (for example, using e-mail) even if the error is not resolved and the transaction is rolled back. With simple modifications, the following user-defined conflict notification method can take more active steps. For example, instead of just recording the notification message, the package can use the DBMS_OFFICE utility package to send an Oracle Office email message to an administrator.

```
CREATE OR REPLACE PACKAGE notify AS
 -- Report uniqueness constraint violations on CUSTOMERS table
 FUNCTION customers_unique_violation (
   first_name         IN OUT VARCHAR2,
   last_name          IN OUT VARCHAR2,
   discard_new_values IN OUT BOOLEAN)
 RETURN BOOLEAN;
END notify;
/

CREATE OR REPLACE PACKAGE BODY notify AS
 -- Define a PL/SQL table to hold the notification message
 TYPE message_table IS TABLE OF VARCHAR2(80) INDEX BY BINARY_INTEGER;

 PROCEDURE report_conflict (
   conflict_report IN MESSAGE_TABLE,
   report_length   IN NUMBER,
   conflict_time   IN DATE,
   conflict_table  IN VARCHAR2,
   table_owner     IN VARCHAR2,
   conflict_type   IN VARCHAR2) IS
 BEGIN
   FOR idx IN 1..report_length LOOP
     BEGIN
```

```
      INSERT INTO sales.conf_report
        (line, txt, timestamp, table_name, table_owner, conflict_type)
        VALUES (idx, SUBSTR(conflict_report(idx),1,80), conflict_time,
                conflict_table, table_owner, conflict_type);
    EXCEPTION WHEN others THEN NULL;
    END;
  END LOOP;
END report_conflict;

-- This is the conflict resolution method that is called first when
-- a uniqueness constraint violated is detected in the CUSTOMERS table.

FUNCTION customers_unique_violation (
  first_name  IN OUT VARCHAR2,
  last_nameIN OUT VARCHAR2,
  discard_new_valuesIN OUT BOOLEAN)
 RETURN BOOLEAN IS
  local_node  VARCHAR2(128);
  conf_report MESSAGE_TABLE;
  conf_time   DATE := SYSDATE;
 BEGIN
 -- Get the global name of the local site
   BEGIN
     SELECT global_name INTO local_node FROM global_name;
   EXCEPTION WHEN others THEN local_node := '?';
   END;
 -- Generate a message for the DBA
 conf_report(1) := 'UNIQUENESS CONFLICT DETECTED IN TABLE CUSTOMERS ON ' ||
                   TO_CHAR(conf_time, 'MM-DD-YYYY HH24:MI:SS');
 conf_report(2) := '  AT NODE ' || local_node;
 conf_report(3) := 'ATTEMPTING TO RESOLVE CONFLICT USING ' ||
                   'APPEND SEQUENCE METHOD';
 conf_report(4) := 'FIRST NAME: ' || first_name;
 conf_report(5) := 'LAST NAME:  ' || last_name;
 conf_report(6) := NULL;
 --- Report the conflict
 report_conflict(conf_report, 5, conf_time, 'CUSTOMERS',
              'OFF_SHORE_ACCOUNTS', 'UNIQUE');
 --- Do not discard the new column values. They are still needed by
 --- other conflict resolution methods.
 discard_new_values := FALSE;
 --- Indicate that the conflict was not resolved.
   RETURN FALSE;
 END customers_unique_violation;
END notify;
/
```

Viewing Conflict Resolution Information

Oracle provides replication catalog (REPCAT) views that you can use to determine what conflict resolution methods are being used by each of the tables and column groups in your replicated environment. Each view has three versions: USER_*, ALL_*, SYS.DBA_*. The views available include the following:

ALL_REPRESOLUTION_ METHOD	Lists all of the available conflict resolution methods.
ALL_REPCOLUMN_GROUP	Lists all of the column groups defined for the database.
ALL_REPGROUPED_COLUMN	Lists all of the columns in each column group in the database.
ALL_REPPRIORITY_GROUP	Lists all of the priority groups and site priority groups defined for the database.
ALL_REPPRIORITY	Lists the values and corresponding priority levels for each priority or site priority group.
ALL_REPCONFLICT	Lists the types of conflicts (delete, update, or uniqueness) for which you have specified a resolution method, for the tables, column groups, and unique constraints in the database.
ALL_REPRESOLUTION	Shows more specific information about the conflict resolution method used to resolve conflicts on each object.
ALL_REPPARAMETER_COLUMN	Shows which columns are used by the conflict resolution methods to resolve a conflict.

See Also: Chapter 9, "Data Dictionary Views".

Index

A

C

D

E

F

G

I

J

M

N

O

P

Q

R

S

T

U

V