Oracle9*i*

Replication Management API Reference

Release 1 (9.0.1)

June 2001

A87502-01

Oracle9*i* Replication Management API Reference, Release 1 (9.0.1)

A87502-01

Primary Author: Randy Urbano

Graphic Artist: Valarie Moore

Contributors: Nimar Arora, Sukanya Balaraman, Yuen Chan, Al Demers, Alan Downing, Curt Elsbernd, Yong Feng, Jairaj Galagali, Denis Goddard, Lewis Kaplan, Viswanathan Krishnamurthy, Anand Lakshminath, Paul Lane, Jing Liu, Edwina Lu, Silvio Martinez, Pat McElroy, Maria Pratt, Arvind Rajaram, Neeraj Shodhan, Gordon Smith, Wayne Smith, Jim Stamos, Janet Stern, Mahesh Subramaniam, Eric Vandevelde, Lik Wong, David Zhang

Contents

3 Create a Master Group

4 Create a Deployment Template

5 Create Materialized View Group

6 Configure Conflict Resolution

8 Managing a Materialized View Replication Environment

9 Managing Replication Objects and Queues

10 Monitoring a Replication Environment

Part III Replication Management API Packages Reference

11 Introduction to the Replication Management API Reference

12 DBMS_DEFER

13 DBMS_DEFER_QUERY

14 DBMS_DEFER_SYS

15 DBMS_MVIEW

16 DBMS_OFFLINE_OG

23 DBMS_REPCAT_RGT

24 DBMS_REPUTIL

Part IV Replication Data Dictionary Reference

25 Replication Catalog Views

Part V Appendixes

A Security Options

B User-Defined Conflict Resolution Methods

Index

Send Us Your Comments

Oracle9*i* Replication Management API Reference, Release 1 (9.0.1)

A87502-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

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

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:

  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

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

Preface

Oracle9i Replication Management API Reference contains information that describes the features and functionality of the replication management API. Specifically, this *Oracle9i Replication Management API Reference* contains reference information for the packages in the replication management API, as well as examples of their use.

In addition, *Oracle9i Replication Management API Reference* contains reference information about the replication catalog and other data dictionary views that are important for replication.

This preface contains these topics:

Audience

Oracle9i Replication Management API Reference is intended for database administrators and application developers who develop and maintain replication environments. These administrators and application developers perform one or more of the following tasks:

- Configure replication sites
- Create master groups
- Create deployment templates
- Create materialized view groups
- Configure conflict resolution
- Manage replication environments
- Use the replication management API
- Monitor replication environments using data dictionary views
- Plan and configure security options

To use this document, you need to be familiar with relational database concepts, distributed database administration, PL/SQL (if using procedural replication), and the operating system under which you run an Oracle Replication environment.

Organization

This document contains:

Part I, "Configuring Your Replication Environment"

Includes instructions on using the replication management API to set up both multimaster replication and materialized view replication. This part also contains instructions for configuring conflict resolution methods and instructions for managing your replication environment using the replication management API.

Chapter 1, "Replication Overview"

Provides an overview of the process for building a replication environment with the replication management API. This chapter also contains some prerequisites for building a replication environment.

Chapter 2, "Create Replication Site"

Describes in detail the process of setting up both a master and materialized view site. Consult this chapter when building a new replication environment and when adding either a new master or materialized view site to an established replication environment.

Chapter 3, "Create a Master Group"

Describes how to build a master group for use with multimaster replication or as a master for a materialized view 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 a Deployment Template"

Describes how to build a materialized view environment with deployment templates, which are the most effective method of distributing a materialized view environment to any number of materialized view sites.

Chapter 5, "Create Materialized View Group"

Describes how to build a materialized view environment with materialized view groups. If deployment templates do not meet your requirements, then Chapter 5 describes in detail how to build a materialized view environment at the materialized view site.

Chapter 6, "Configure Conflict Resolution"

Describes the conflict resolution methods that can help your data converge at all sites when a data conflict arises.

Part II, "Managing and Monitoring Your Replication Environment"

Includes instructions on managing a replication environment using the replication management API.

Chapter 7, "Managing a Master Replication Environment"

Describes many of the management tasks that you may need to perform to manage a multimaster replication environment. Topics include adding new master sites, master group management, and more.

Chapter 8, "Managing a Materialized View Replication Environment"

Describes many of the management tasks that you may need to perform to manage a materialized view replication environment. Topics include using a group owner, managing materialized view logs, offline instantiation, and more.

Chapter 9, "Managing Replication Objects and Queues"

Describes many of the management tasks that you may need to perform to manage your replication environment. Topics include altering replicated objects, managing the deferred transactions queue, managing the error queue, and more.

Chapter 10, "Monitoring a Replication Environment"

Describes many of the queries you can run to monitor your replication environment.

Part III, "Replication Management API Packages Reference"

Includes reference information about the replication management API, including: the procedures and functions in each package, the parameters for each packaged procedure and function, and exceptions that each procedure or function can raise.

Chapter 11, "Introduction to the Replication Management API Reference"

Introduces the replication management API and includes examples for its use.

Chapter 12, "DBMS_DEFER"

Describes the procedures in the `DBMS_DEFER` package.

Chapter 13, "DBMS_DEFER_QUERY"

Describes the procedures and functions in the `DBMS_DEFER_QUERY` package.

Chapter 14, "DBMS_DEFER_SYS"

Describes the procedures and functions in the `DBMS_DEFER_SYS` package.

Chapter 15, "DBMS_MVIEW"

Describes the procedures and functions in the `DBMS_MVIEW` package.

Chapter 16, "DBMS_OFFLINE_OG"

Describes the procedures in the `DBMS_OFFLINE_OG` package.

Chapter 17, "DBMS_OFFLINE_SNAPSHOT"

Describes the procedures in the `DBMS_OFFLINE_SNAPSHOT` package.

Chapter 18, "DBMS_RECTIFIER_DIFF"

Describes the procedures in the `DBMS_RECTIFIER_DIFF` package.

Chapter 19, "DBMS_REFRESH"

Describes the procedures in the DBMS_REFRESH package.

Chapter 20, "DBMS_REPCAT"

Describes the procedures and functions in the DBMS_REPCAT package.

Chapter 21, "DBMS_REPCAT_ADMIN"

Describes the procedures in the DBMS_REPCAT_ADMIN package.

Chapter 22, "DBMS_REPCAT_INSTANTIATE"

Describes the procedures and functions in the DBMS_REPCAT_INSTANTIATE package.

Chapter 23, "DBMS_REPCAT_RGT"

Describes the procedures and functions in the DBMS_REPCAT_RGT package.

Chapter 24, "DBMS_REPUTIL"

Describes the procedures and functions in the DBMS_REPUTIL package.

Part IV, "Replication Data Dictionary Reference"

Describes data dictionary views that provide information about your replication environment.

Chapter 25, "Replication Catalog Views"

Describes the replication catalog, which contains data dictionary views that are used by master and materialized view sites to determine such information as what objects are being replicated, where they are being replicated, and if any errors have occurred during replication.

Chapter 26, "Replication Dynamic Performance Views"

Describes the dynamic performance views that are used by master and materialized view sites to determine such information as which materialized views are being refreshed currently and statistics about the deferred transaction queue.

Chapter 27, "Deferred Transaction Views"

Describes the data dictionary views that contain information about 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.

Chapter 28, "Materialized View and Refresh Group Views"

Describes data dictionary views that provide information about materialized views and materialized view refresh groups.

Part V, "Appendixes"

Includes the following appendixes:

Appendix A, "Security Options"

Describes setting up security for multimaster and materialized view 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.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i Replication*
- *Oracle9i Database Concepts*
- *Oracle9i Database Administrator's Guide*
- *Oracle9i SQL Reference*
- *PL/SQL User's Guide and Reference* (if you plan to use procedural replication)

You may find more information about a particular topic in the other documents in the Oracle9*i* documentation set.

Many of the examples in this book use the sample schemas of the seed database, which is installed by default when you install Oracle. Refer to *Oracle9i Sample Schemas* for information on how these schemas were created and how you can use them yourself.

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://technet.oracle.com/membership/index.htm`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://technet.oracle.com/docs/index.htm`

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width font)`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
`lowercase monospace (fixed-width font)`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true.` Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase monospace (fixed-width font) italic`	Lowercase monospace italic font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`
...	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS `*`subquery`*`;`
	■ That you can repeat a portion of the code	`SELECT `*`col1`*`, `*`col2`*`, ... , `*`coln`*` FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct    CONSTANT NUMBER(4) := 3;`
`Italics`	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`* `DB_NAME = `*`database_name`*

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Documentation Accessibility

Oracle's goal is to make our products, services, and supporting documentation accessible to the disabled community with good usability. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Part I

Configuring Your Replication Environment

Part I contains instructions for using the replication management API to set up both multimaster replication and materialized view replication. This part also contains instructions for configuring conflict resolution methods using the replication management API.

Part I contains the following chapters:

1

Replication Overview

This chapter reviews the process of building a replication environment with the replication management API.

This chapter contains these topics:

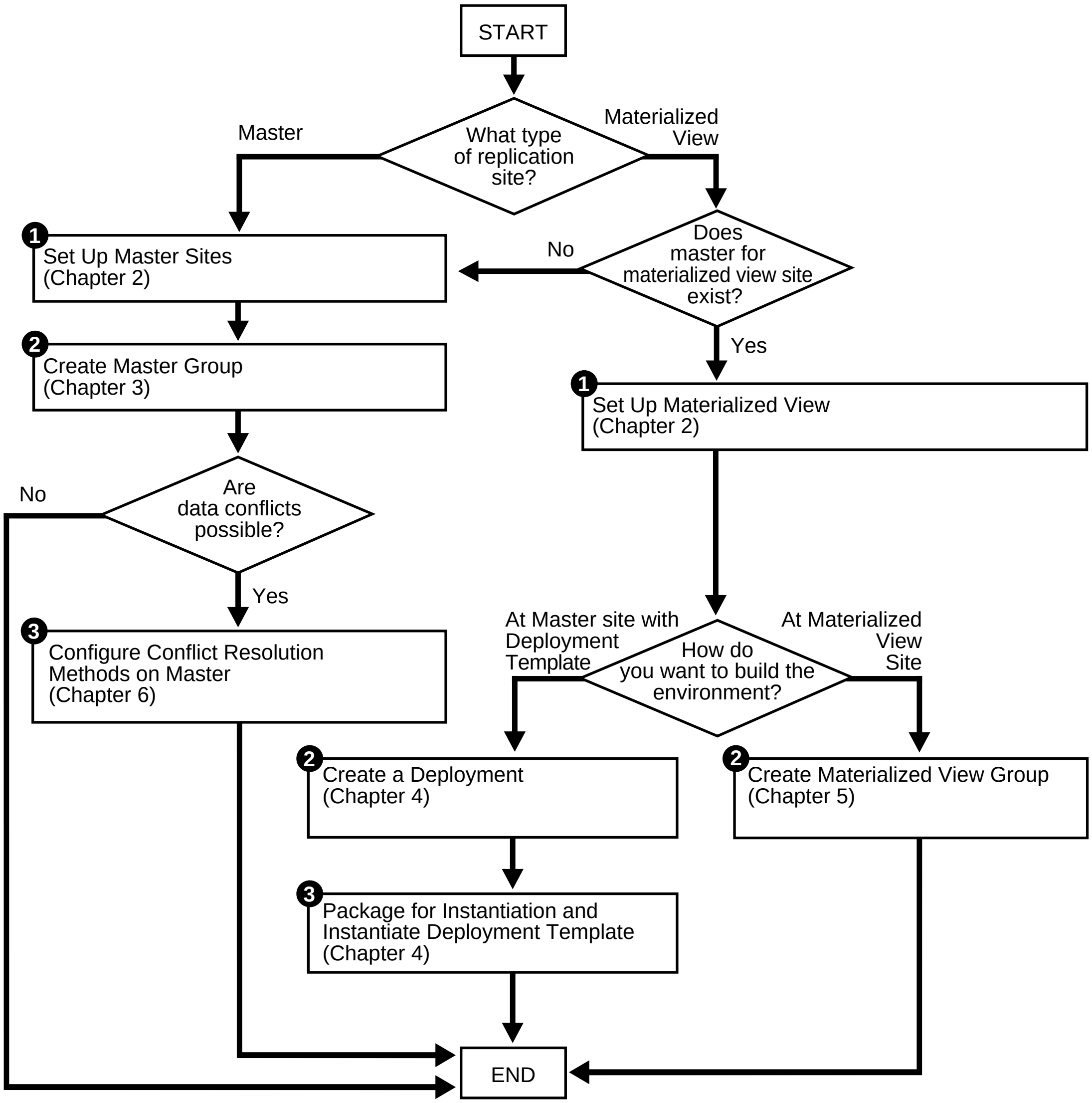
- Creating a Replication Environment Overview
- Before You Start

Creating a Replication Environment Overview

Figure 1–1 illustrates the basic steps required to build a replication 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 groups and materialized view groups. After you have built your replication environment, make sure that you review Chapter 6 and Chapter 9 to learn about conflict resolution and managing your replication environment.

Figure 1–1 Create Replication Environment Process



Before You Start

Before you begin setting up your replication site, make sure you plan your replication environment so that it meets your needs. Planning considerations include:

- Designing your replicated database objects
- Deciding on the settings of initialization parameters that are important for replication
- Deciding whether you want to create a multimaster replication environment or a materialized view replication environment, or if you want to combine both types of replication environments into a hybrid environment
- Deciding how you want to configure your scheduled links
- Deciding how you want to configure your scheduled purges
- Deciding whether you want to use serial or parallel propagation
- If you use parallel propagation, then deciding on the degree of parallelism
- If you plan to create a materialized view environment, then deciding whether you want to use deployment templates to create the environment
- Analyzing your environment for possible conflicts and, if conflicts are possible, then deciding which conflict resolution methods to use
- Configuring security for your replication environment
- Designing your replication environment for survivability

> **See Also:** *Oracle9i Replication* for more information planning your replication environment

2

Create Replication Site

This chapter illustrates how to set up both a master site and a materialized view replication site using the replication management API.

This chapter contains these topics:

Overview of Setting Up Replication Sites

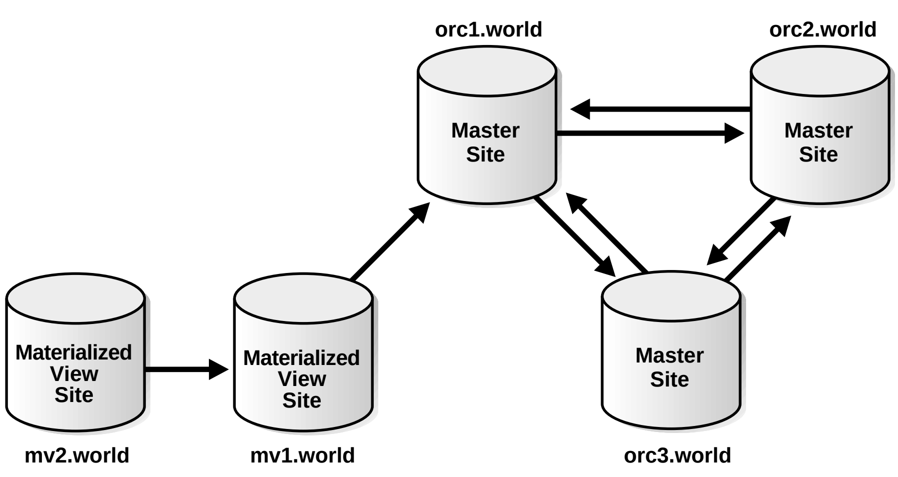
Before you build your replication environment, you need to set up the sites that will participate in the replication environment. As illustrated in Figure 2–2 and Figure 2–3, there are separate processes for setting up a master site versus setting up a materialized view site.

The examples in this chapter, and in other chapters, use the following five databases:

- `orc1.world`
- `orc2.world`
- `orc3.world`
- `mv1.world`
- `mv2.world`

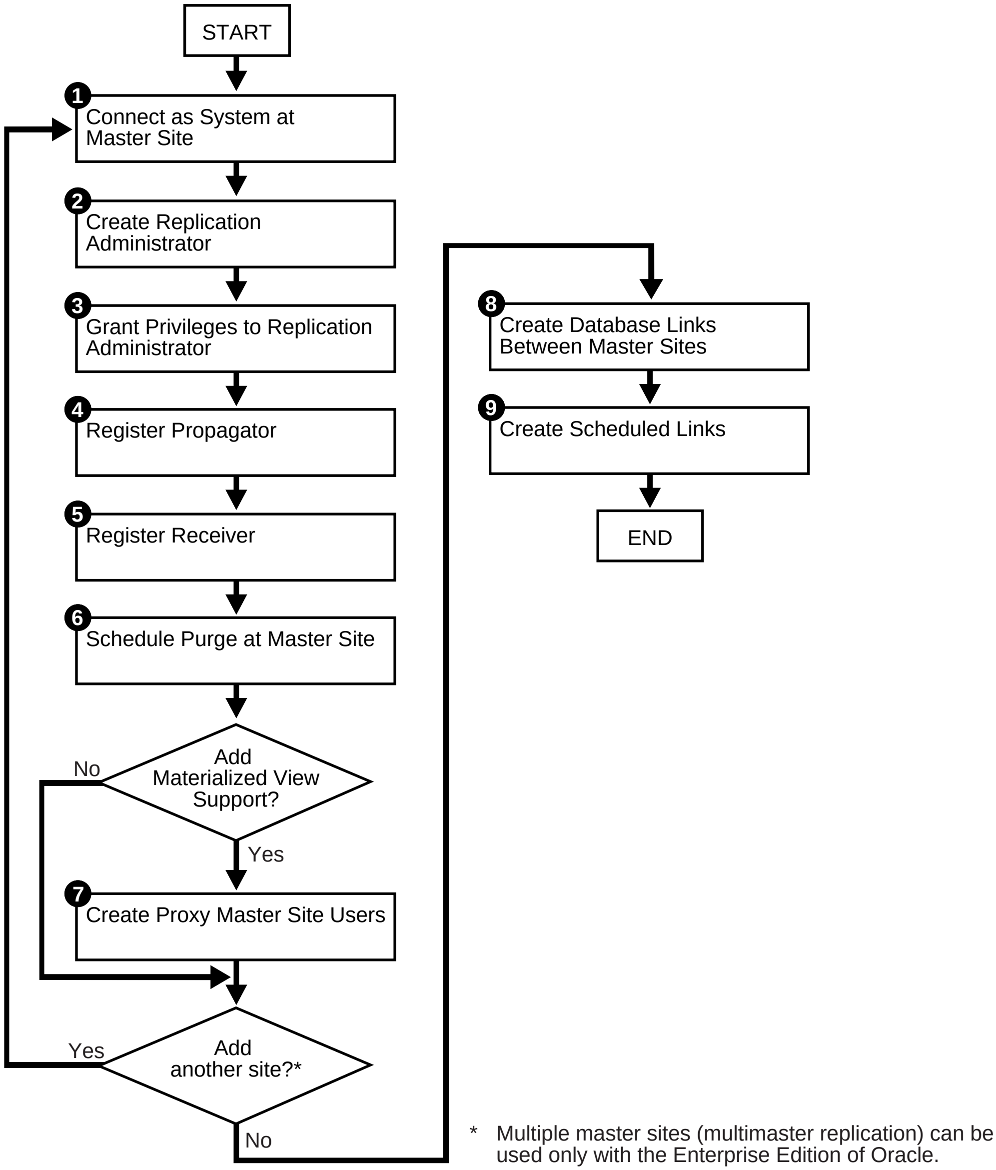
Chapters 2 - 6 work with the replication environment illustrated in Figure 2–1. You start to create this environment using the instructions in this chapter. Notice that `mv2.world` is a materialized view based on the `mv1.world` materialized view, creating a multitier materialized view environment. The arrows in Figure 2–1 represent database links.

Figure 2–1 Three Master Sites and Two Materialized View Sites



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 materialized view site.

Figure 2–2 Setting Up Master Sites



Setting Up Master Sites

The following sections contain step-by-step instructions for setting up the three master sites in our sample replication environment: `orc1.world`, `orc2.world`, and `orc3.world`.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 2-31 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

Setting Up orc1.world

Complete the following steps to set up the `orc1.world` master site.

1. Connect as `SYSTEM` at a master site at `orc1.world`.

   Connect as `SYSTEM` to the database that you want to set up for replication. After you set up `orc1.world`, begin again with Step 1 for site `orc2.world` on page 2-9 and Step 1 for site `orc3.world` on page 2-12.

```
*/

CONNECT system/manager@orc1.world

/*
```

2. Create replication administrator at `orc1.world`.

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

```
*/

CREATE USER repadmin IDENTIFIED BY repadmin;

/*
```

3. Grant privileges to replication administrator at orc1.world by completing the following steps:

   a. Execute the GRANT_ADMIN_ANY_SCHEMA procedure to grant the replication administrator powerful privileges to create and manage a replicated environment.

```
*/

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

/*
```

   See Also: "GRANT_ADMIN_ANY_SCHEMA Procedure" on page 21-3

   b. If you want your repadmin to be able to create materialized view logs for any replicated table, then grant COMMENT ANY TABLE and LOCK ANY TABLE to repadmin:

```
*/

GRANT COMMENT ANY TABLE TO repadmin;
GRANT LOCK ANY TABLE TO repadmin;

/*
```

   c. If you want your repadmin to be able to connect to the Replication Management tool, then grant SELECT ANY DICTIONARY to repadmin:

```
*/

GRANT SELECT ANY DICTIONARY TO repadmin;

/*
```

4. Register propagator at `orc1.world`.

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

```
*/

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

/*
```

   See Also: "REGISTER_PROPAGATOR Procedure" on page 14-19

5. Register receiver at `orc1.world`.

   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;
/

/*
```

   See Also: "REGISTER_USER_REPGROUP Procedure" on page 21-5

6. Schedule purge at master site `orc1.world`.

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

Note: Date expressions are used for the `NEXT_DATE` and `INTERVAL` parameters. For example:

- Now is specified as: `SYSDATE`
- An interval of one hour is specified as: `SYSDATE + 1/24`
- An interval of seven days could be specified as: `SYSDATE + 7`

```
*/

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
      next_date => SYSDATE,
      interval => 'SYSDATE + 1/24',
      delay_seconds => 0);
END;
/

/*
```

See Also: "SCHEDULE_PURGE Procedure" on page 14-20

7. If you plan to create materialized view sites based on this master site, then create proxy master site users at `orc1.world` that correspond to users at the materialized view site by completing the following steps:

   a. Create proxy materialized view administrator.

      The proxy materialized view administrator performs tasks at the target master site on behalf of the materialized view administrator at the materialized view site.

```
*/

CONNECT SYSTEM/MANAGER@orc1.world

CREATE USER proxy_mviewadmin IDENTIFIED BY proxy_mviewadmin;
```

```
BEGIN
   DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
      username => 'proxy_mviewadmin',
      privilege_type => 'proxy_snapadmin',
      list_of_gnames => NULL);
END;
/

-- Place GRANT SELECT_CATALOG_ROLE statement here if necessary.

/*
```

If you want your materialized view administrator at materialized view sites to be able to perform administrative operations using the Replication Management tool, then grant `SELECT_CATALOG_ROLE` to `proxy_mviewadmin`:

```
GRANT SELECT_CATALOG_ROLE TO proxy_mviewadmin;
```

Granting this privilege to the `proxy_mviewadmin` is not required if you do not plan to use the Replication Management tool. However, if you plan to use the Replication Management tool, then move the `GRANT` statement to the line directly after the previous `REGISTER_USER_REPGROUP` statement.

See Also: "Security Setup for Materialized View Replication" on page A-7

b. Create proxy refresher.

The proxy refresher performs tasks at the master site on behalf of the refresher at the materialized view site.

```
*/

CREATE USER proxy_refresher IDENTIFIED BY proxy_refresher;

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

/*
```

Setting Up orc2.world

Complete the following steps to set up the `orc2.world` master site.

1. Connect as `SYSTEM` at `orc2.world`.

> **Note:** Multiple master sites (multimaster replication) can only be used with Oracle Enterprise Edition. If you are not using Oracle Enterprise Edition, then skip to "Setting Up Materialized View Sites" on page 2-19.

You must connect as `SYSTEM` to the database that you want to set up for replication. After you set up `orc2.world`, begin with Step 1 for site `orc3.world` on page 2-12.

```
*/

CONNECT system/manager@orc2.world

/*
```

2. Create replication administrator at `orc2.world`.

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

```
*/

CREATE USER repadmin IDENTIFIED BY repadmin;

/*
```

3. Grant privileges to replication administrator at `orc2.world` by completing the following steps:

   a. Execute the `GRANT_ADMIN_ANY_SCHEMA` procedure to grant the replication administrator powerful privileges to create and manage a replicated environment.

   ```
   */

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

   /*
   ```

   See Also: "GRANT_ADMIN_ANY_SCHEMA Procedure" on page 21-3

   b. If you want your `repadmin` to be able to create materialized view logs for any replicated table, then grant `COMMENT ANY TABLE` and `LOCK ANY TABLE` privileges to `repadmin`:

   ```
   */

   GRANT COMMENT ANY TABLE TO repadmin;
   GRANT LOCK ANY TABLE TO repadmin;

   /*
   ```

   c. If you want your `repadmin` to be able to connect to the Replication Management tool, then grant `SELECT ANY DICTIONARY` to `repadmin`:

   ```
   */

   GRANT SELECT ANY DICTIONARY TO repadmin;

   /*
   ```

4. Register propagator at `orc2.world`.

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

```
*/

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

/*
```

See Also: "REGISTER_PROPAGATOR Procedure" on page 14-19

5. Register receiver at `orc2.world`.

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;
/

/*
```

See Also: "REGISTER_USER_REPGROUP Procedure" on page 21-5

6. Schedule purge at master site at `orc2.world`.

In order to keep the size of the deferred transaction queue in check, you should purge successfully completed deferred transactions. The `SCHEDULE_PURGE` procedure 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);
END;
/

/*
```

See Also: "SCHEDULE_PURGE Procedure" on page 14-20

Setting Up orc3.world

Complete the following steps to set up the `orc3.world` master site.

1. Connect as `SYSTEM` at `orc3.world`.

   Note: Multiple master sites (multimaster replication) can be used only with Oracle Enterprise Edition. If you are not using Oracle Enterprise Edition, then skip to "Setting Up Materialized View Sites" on page 2-19.

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

   ```
   */

   CONNECT system/manager@orc3.world

   /*
   ```

2. Create replication administrator at `orc3.world`.

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

   ```
   */

   CREATE USER repadmin IDENTIFIED BY repadmin;

   /*
   ```

3. Grant privileges to replication administrator at `orc3.world` by completing the following steps:

   a. Execute the `GRANT_ADMIN_ANY_SCHEMA` procedure to grant the replication administrator powerful privileges to create and manage a replicated environment.

```
*/

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

/*
```

   See Also: "GRANT_ADMIN_ANY_SCHEMA Procedure" on page 21-3

   b. If you want your `repadmin` to be able to create materialized view logs for any replicated table, then grant `COMMENT ANY TABLE` and `LOCK ANY TABLE` to `repadmin`:

```
*/

GRANT COMMENT ANY TABLE TO repadmin;
GRANT LOCK ANY TABLE TO repadmin;

/*
```

   c. If you want your `repadmin` to be able to connect to the Replication Management tool, then grant `SELECT ANY DICTIONARY` to `repadmin`:

```
*/

GRANT SELECT ANY DICTIONARY TO repadmin;

/*
```

4. Register propagator at `orc3.world`.

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

```
*/

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

/*
```

   See Also: "REGISTER_PROPAGATOR Procedure" on page 14-19

5. Register receiver at `orc3.world`.

   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;
/

/*
```

   See Also: "REGISTER_USER_REPGROUP Procedure" on page 21-5

6. Schedule purge at master site at `orc3.world`.

   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);
END;
/

/*
```

See Also: "SCHEDULE_PURGE Procedure" on page 14-20

Creating Scheduled Links Between the Master Sites

Complete the following steps to create scheduled links between the master sites.

1. 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. 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.

See Also: *Oracle9i Database Administrator's Guide* for more information about database links

```
*/

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;

/*
```

2. Define a schedule for each database link to create scheduled links.

Create a scheduled link by defining a database link when you execute the `SCHEDULE_PUSH` procedure. 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 1. The database link is specified in the destination parameter of the `SCHEDULE_PUSH` procedure.

Even when using Oracle's asynchronous replication mechanisms, you can configure a scheduled link to simulate continuous, real-time replication. The scheduled links in this example simulate continuous replication.

See Also:

- "SCHEDULE_PUSH Procedure" on page 14-22
- *Oracle9i Replication* for more information about simulating continuous replication

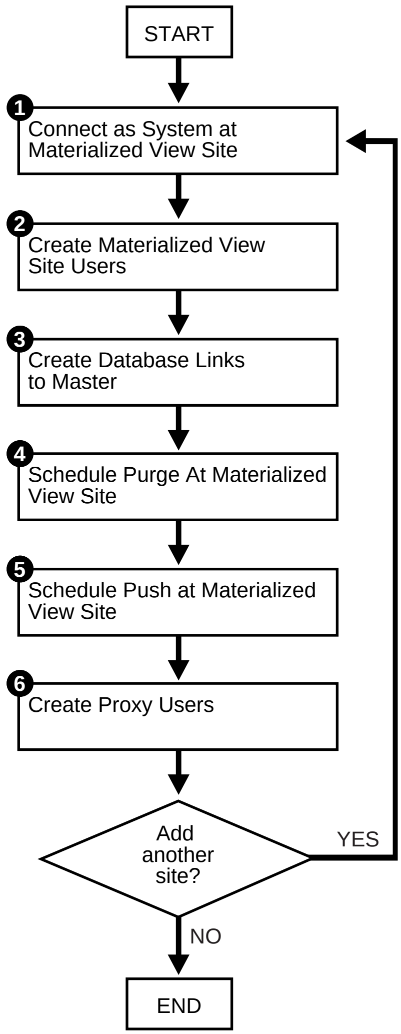
```
*/

CONNECT repadmin/repadmin@orc1.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      destination => 'orc2.world',
      interval => 'SYSDATE + (1/144)',
      next_date => SYSDATE,
      parallelism => 1,
      execution_seconds => 1500,
      delay_seconds => 1200);
END;
/
```

```
BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      destination => 'orc2.world',
      interval => 'SYSDATE + (1/144)',
      next_date => SYSDATE,
      parallelism => 1,
      execution_seconds => 1500,
      delay_seconds => 1200);
END;
/

BEGIN
DBMS_DEFER_SYS.SCHEDULE_PUSH (
      destination => 'orc3.world',
      interval => 'SYSDATE + (1/144)',
      next_date => SYSDATE,
      parallelism => 1,
      execution_seconds => 1500,
      delay_seconds => 1200);
END;
/

CONNECT repadmin/repadmin@orc2.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      destination => 'orc1.world',
      interval => 'SYSDATE + (1/144)',
      next_date => SYSDATE,
      parallelism => 1,
      execution_seconds => 1500,
      delay_seconds => 1200);
END;
/

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      destination => 'orc3.world',
      interval => 'SYSDATE + (1/144)',
      next_date => SYSDATE,
      parallelism => 1,
      execution_seconds => 1500,
      delay_seconds => 1200);
END;
/
```

```
CONNECT repadmin/repadmin@orc3.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      destination => 'orc1.world',
      interval => 'SYSDATE + (1/144)',
      next_date => SYSDATE,
      parallelism => 1,
      execution_seconds => 1500,
      delay_seconds => 1200);
END;
/

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      destination => 'orc2.world',
      interval => 'SYSDATE + (1/144)',
      next_date => SYSDATE,
      parallelism => 1,
      execution_seconds => 1500,
      delay_seconds => 1200);
END;
/

/************************END OF SCRIPT***********************************/
```

Setting Up Materialized View Sites

Figure 2–3 Setting Up Materialized View Sites



Setting Up mv1.world

Complete the following steps to set up the `mv1.world` master materialized view site. `mv1.world` is a master materialized view site because `mv2.world` will be based on it.

> **Note:** If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 2-31 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect as `SYSTEM` at materialized view site at `mv1.world`.

   You must connect as `SYSTEM` to the database that you want to set up as a materialized view site.

   ```
   */

   CONNECT SYSTEM/MANAGER@mv1.world

   /*
   ```

2. Create materialized view site users at `mv1.world`.

   Several users must be created at the materialized view site. These users are:

   - Materialized view administrator
   - Propagator
   - Refresher
   - Receiver (if the site will serve as a master materialized view site for other materialized views, as `mv1.world` is)

   Complete the following steps to create these users.

a. Create materialized view administrator.

The materialized view administrator is responsible for creating and managing the materialized view site. Execute the `GRANT_ADMIN_ANY_SCHEMA` procedure to grant the materialized view administrator the appropriate privileges.

```
*/

CREATE USER mviewadmin IDENTIFIED BY mviewadmin;

BEGIN
   DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
      username => 'mviewadmin');
END;
/

GRANT COMMENT ANY TABLE TO mviewadmin;

GRANT LOCK ANY TABLE TO mviewadmin;

/*
```

b. If you want your `mviewadmin` to be able to connect to the Replication Management tool, then grant `SELECT ANY DICTIONARY` to `mviewadmin`:

```
*/

GRANT SELECT ANY DICTIONARY TO mviewadmin;

/*
```

c. 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;
/
```

```
/*
```

d. Create refresher.

The refresher is responsible for "pulling" changes made to the replicated tables at the target master site to the materialized view site. This user refreshes one or more materialized views. If you want the mviewadmin user to be the refresher, then this step is not required.

```
*/

CREATE USER refresher IDENTIFIED BY refresher;

GRANT CREATE SESSION TO refresher;

GRANT ALTER ANY MATERIALIZED VIEW TO refresher;

/*
```

e. Register receiver.

The receiver receives the propagated deferred transactions sent by the propagator from materialized view sites. The receiver is necessary only if the site will function as a master materialized view site for other materialized view sites.

```
*/

BEGIN
   DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
      username => 'mviewadmin',
      privilege_type => 'receiver',
      list_of_gnames => NULL);
END;
/

/*
```

See Also: "REGISTER_USER_REPGROUP Procedure" on page 21-5

3. Create database links to the master site by completing the following steps.

   a. Create public database link.

```
*/

CONNECT SYSTEM/MANAGER@mv1.world

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

/*
```

   b. Create materialized view administrator database link.

   You need to create a database link from the materialized view administrator at the materialized view site to the proxy materialized view administrator at the master site.

```
*/

CONNECT mviewadmin/mviewadmin@mv1.world;

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

/*
```

   c. Create propagator/receiver database link.

   You need to create a database link from the propagator at the materialized view site to the receiver at the master site. The receiver was defined when you created the master site.

```
*/

CONNECT propagator/propagator@mv1.world

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

/*
```

See Also: Step 5 on page 2-11

4. Schedule purge at the `mv1.world` materialized view site.

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

```
*/

CONNECT mviewadmin/mviewadmin@mv1.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
   next_date => SYSDATE,
   interval => 'SYSDATE + 1/24',
   delay_seconds => 0,
   rollback_segment => '');
END;
/

/*
```

   See Also: "SCHEDULE_PURGE Procedure" on page 14-20

5. If the materialized view site has a constant connection to its master site, then you can optionally schedule push at the `mv1.world` materialized view site. If the materialized view site is disconnected from its master site for extended periods of time, then it is typically better not to schedule push and refresh on demand, which pushes changes to the master site.

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

```
*/

CONNECT mviewadmin/mviewadmin@mv1.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;
/

/*
```

See Also: "SCHEDULE_PUSH Procedure" on page 14-22

6. Create proxy users at the `mv1.world` materialized view site by completing the following steps.

   a. Create proxy materialized view administrator.

      The proxy materialized view administrator performs tasks at the target master materialized view site on behalf of the materialized view administrator at the materialized view sites based on this materialized view site. This user is not required if the site will not function as a master materialized view site for other materialized view sites.

```
*/

CONNECT SYSTEM/MANAGER@mv1.world

CREATE USER proxy_mviewadmin IDENTIFIED BY proxy_mviewadmin;

BEGIN
   DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
      username => 'proxy_mviewadmin',
      privilege_type => 'proxy_snapadmin',
      list_of_gnames => NULL);
END;
/

-- Place GRANT SELECT_CATALOG_ROLE statement here if necessary.

/*
```

If you want your materialized view administrator at materialized view sites based on this materialized view site to be able to perform administrative operations using the Replication Management tool, then grant `SELECT_CATALOG_ROLE` to `proxy_mviewadmin`:

```
GRANT SELECT_CATALOG_ROLE TO proxy_mviewadmin;
```

Granting this privilege to the `proxy_mviewadmin` is not required if you do not plan to use the Replication Management tool. However, if you plan to use the Replication Management tool, then move the `GRANT` statement to the line directly after the previous `REGISTER_USER_REPGROUP` statement.

b. Create proxy refresher.

The proxy refresher performs tasks at the master materialized view site on behalf of the refresher at the materialized view sites based on this materialized view site. This user is not required if the site will not function as a master materialized view site for other materialized view sites.

```
*/

CREATE USER proxy_refresher IDENTIFIED BY proxy_refresher;

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

/*
```

See Also: "Security Setup for Materialized View Replication" on page A-7

Setting Up mv2.world

Complete the following steps to set up the `mv2.world` materialized view site. `mv2.world` is part of a multitier materialized view configuration because it is based on `mv1.world`, another materialized view.

1. Connect as `SYSTEM` at level 2 materialized view site `mv2.world`.

You must connect as `SYSTEM` to the database that you want to set up as a level 2 materialized view site. This site, `mv2.world`, will be a materialized view site that is based on mv1.world.

```
*/

CONNECT SYSTEM/MANAGER@mv2.world

/*
```

2. Create level 2 materialized view site users at `mv2.world`.

Several users must be created at the level 2 materialized view site. These users are:

- Materialized view administrator
- Propagator
- Refresher

Complete the following steps to create these users.

a. Create materialized view administrator.

The materialized view administrator is responsible for creating and managing the level 2 materialized view site. Execute the `GRANT_ADMIN_ANY_SCHEMA` procedure to grant the materialized view administrator the appropriate privileges.

```
*/

CREATE USER mviewadmin IDENTIFIED BY mviewadmin;

BEGIN
   DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
      username => 'mviewadmin');
END;
/
```

```
/*
```

b. If you want your `mviewadmin` to be able to connect to the Replication Management tool, then grant `SELECT ANY DICTIONARY` to `mviewadmin`:

```
*/

GRANT SELECT ANY DICTIONARY TO mviewadmin;

/*
```

c. Create propagator.

The propagator is responsible for propagating the deferred transaction queue to the target master materialized view site.

```
*/

CREATE USER propagator IDENTIFIED BY propagator;

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

/*
```

d. Create refresher.

The refresher is responsible for "pulling" changes made to the replicated materialized views at the target master materialized view site to the level 2 materialized view site.

```
*/

CREATE USER refresher IDENTIFIED BY refresher;

GRANT CREATE SESSION TO refresher;
GRANT ALTER ANY MATERIALIZED VIEW TO refresher;

/*
```

3. Create database links to master materialized view site by completing the following steps.

   a. Create public database link.

```
*/

CONNECT SYSTEM/MANAGER@mv2.world

CREATE PUBLIC DATABASE LINK mv1.world USING 'mv1.world';

/*
```

   b. Create materialized view administrator database link.

   You need to create a database link from the materialized view administrator at the level 2 materialized view site to the proxy materialized view administrator at the master materialized view site.

```
*/

CONNECT mviewadmin/mviewadmin@mv2.world;

CREATE DATABASE LINK mv1.world
  CONNECT TO proxy_mviewadmin IDENTIFIED BY proxy_mviewadmin;

/*
```

   c. Create propagator/receiver database link.

   You need to create a database link from the propagator at the level 2 materialized view site to the receiver at the master materialized view site. The receiver was defined when you created the master materialized view site.

```
*/

CONNECT propagator/propagator@mv2.world

CREATE DATABASE LINK mv1.world
  CONNECT TO mviewadmin IDENTIFIED BY mviewadmin;

/*
```

4. Schedule purge at level 2 materialized view site at `mv2.world`.

   In order to keep the size of the deferred transaction queue in check, you should purge successfully completed deferred transactions. The `SCHEDULE_PURGE` procedure automates the purge process for you. If your level 2 materialized view site only contains "read-only" materialized views, then you do not need to execute this procedure.

```
*/

CONNECT mviewadmin/mviewadmin@mv2.world

BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
     next_date => SYSDATE,
     interval => 'SYSDATE + 1/24',
     delay_seconds => 0,
     rollback_segment => '');
END;
/

/*
```

   See Also: "SCHEDULE_PURGE Procedure" on page 14-20

5. If the materialized view site has a constant connection to its master materialized view site, then you can optionally schedule push at the `mv2.world` materialized view site. If the materialized view site is disconnected from its master materialized view site for extended periods of time, then it is typically better not to schedule push and refresh on demand, which pushes changes to the master materialized view site.

   The `SCHEDULE_PUSH` procedure schedules when the deferred transaction queue should be propagated to the target master materialized view site.

```
*/

CONNECT mviewadmin/mviewadmin@mv2.world
```

```
BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      destination => 'mv1.world',
      interval => 'SYSDATE + 1/24',
      next_date => SYSDATE,
      stop_on_error => FALSE,
      delay_seconds => 0,
      parallelism => 0);
END;
/

/*
```

See Also: "SCHEDULE_PUSH Procedure" on page 14-22

```
/************************* END OF SCRIPT **********************************/
```

3

Create a Master Group

This chapter illustrates how to create a master group at a master replication site. This chapter contains these topics:

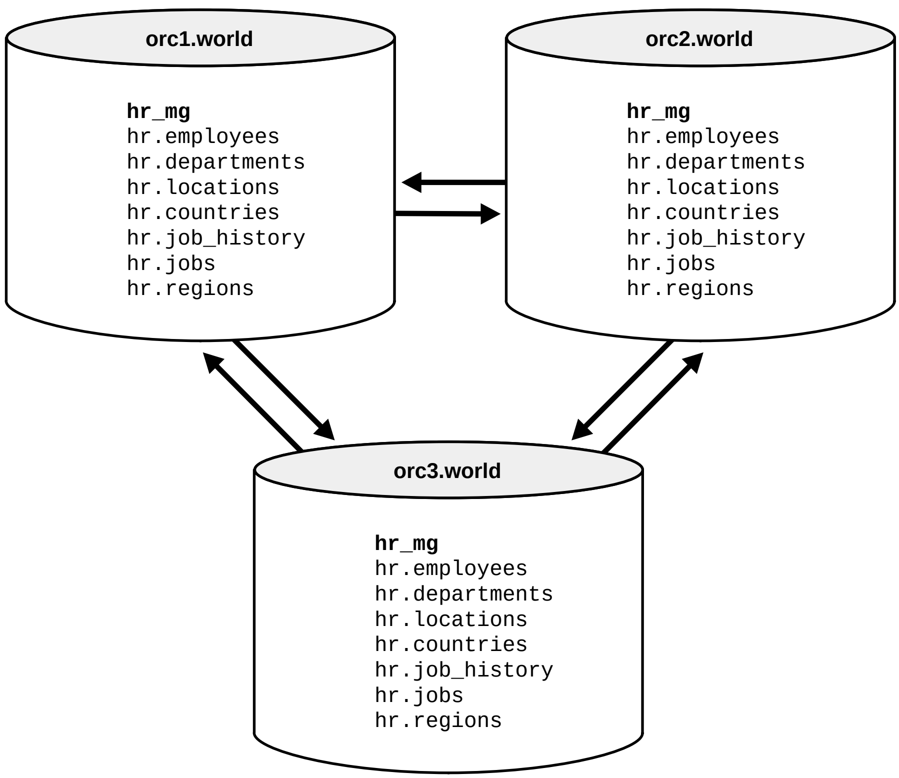
Overview of Creating a Master Group

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

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

In this chapter, you create the `hr_repg` master group and replicate the objects illustrated in Figure 3–1.

Figure 3–1 Replicate the Tables in the hr Schema Between All Sites



Before You Start

In order for the script in this chapter to work as designed, it is assumed that the `hr` schema exists at `orc1.world`, `orc2.world`, and `orc3.world`. The `hr` schema includes the following database objects:

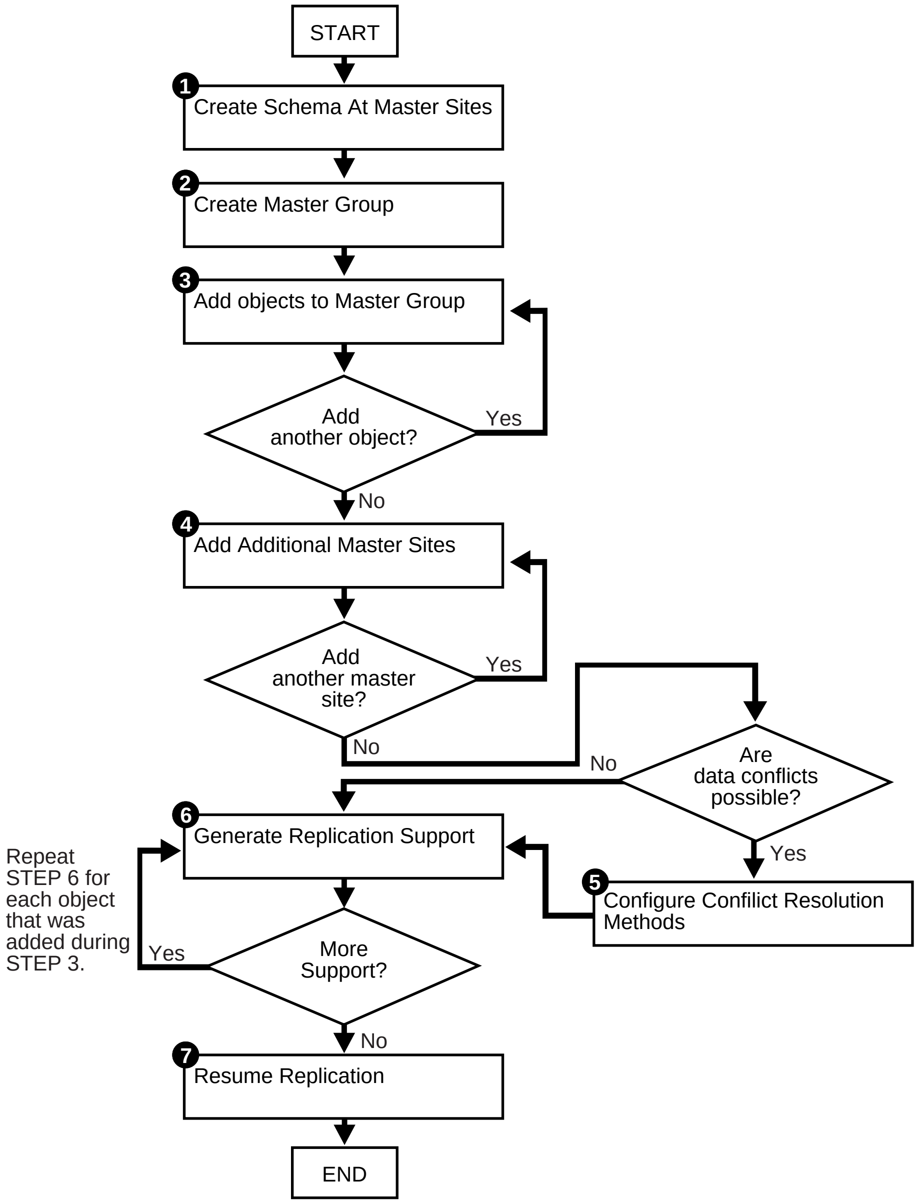
- `countries` table
- `departments` table
- `employees` table
- `jobs` table
- `job_history` table
- `locations` table
- `regions` table
- `dept_location_ix` index
- `emp_department_ix` index
- `emp_job_ix` index
- `emp_manager_ix` index
- `jhist_department_ix` index
- `jhist_employee_ix` index
- `jhist_job_ix` index
- `loc_country_ix` index

The indexes listed are the indexes based on foreign key columns in the `hr` schema. When replicating tables with foreign key referential constraints, Oracle Corporation recommends that you always index foreign key columns and replicate these indexes, unless no updates and deletes are allowed in the parent table. Indexes are not replicated automatically.

By default, the `hr` schema is installed automatically when you install Oracle9*i*. The example script in this chapter assumes that the hr schema exists at all master sites and that the schema contains all of these database objects at each site. The example script also assumes that the tables contain the data that is inserted automatically during Oracle installation. If the `hr` schema is not installed at your replication sites, then you can install it manually.

> **See Also:** *Oracle9i Sample Schemas* for information about the `hr` schema and the other sample schemas, and for information about installing the sample schemas manually

Figure 3–2 Creating a Master Group



Creating a Master Group

Complete the following steps to create the `hr_repg` master group.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 3-14 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Create schema at master sites.

   If the schema does not already exist at all of the master sites participating in the master group, then create the schema now and grant it all of the necessary privileges. This example uses the `hr` schema, which is one of the sample schemas that are installed by default when you install Oracle. So, the `hr` schema should exist at all master sites.

   ```
   */

   PAUSE Press <RETURN> to continue when the schema exists at all master sites.

   /*
   ```

2. Create master group.

   Use the `CREATE_MASTER_REPGROUP` procedure 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. This step must be completed by the replication administrator.

   ```
   */

   CONNECT repadmin/repadmin@orc1.world

   BEGIN
      DBMS_REPCAT.CREATE_MASTER_REPGROUP (
         gname => 'hr_repg');
   END;
   /

   /*
   ```

3. Add objects to master group.

   Use the `CREATE_MASTER_REPOBJECT` procedure to add an object to your master group. In most cases, you probably will be adding tables and indexes to your master group, but you can also add procedures, views, synonyms, and so on.

```
*/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'TABLE',
      oname => 'countries',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'TABLE',
      oname => 'departments',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'TABLE',
      oname => 'employees',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'TABLE',
      oname => 'jobs',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'TABLE',
      oname => 'job_history',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'TABLE',
      oname => 'locations',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'TABLE',
      oname => 'regions',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'INDEX',
      oname => 'dept_location_ix',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'INDEX',
      oname => 'emp_department_ix',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'INDEX',
      oname => 'emp_job_ix',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'INDEX',
      oname => 'emp_manager_ix',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'INDEX',
      oname => 'jhist_department_ix',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'INDEX',
      oname => 'jhist_employee_ix',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'INDEX',
      oname => 'jhist_job_ix',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'INDEX',
      oname => 'loc_country_ix',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/

/*
```

> **See Also:** "CREATE_MASTER_REPOBJECT Procedure" on page 20-53

4. Add additional master sites.

   After you have defined your master group at the master definition 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 replication environment. You might have guessed that you will be adding the `orc2.world` and `orc3.world` sites to the replication environment. This example creates the master group at all master sites, but you have the option of creating the master group at one master site now and adding additional master sites later without quiescing the database. In this case, you can skip this step.

   > **See Also:** "Adding New Master Sites Without Quiescing the Master Group" on page 7-4 for more information

   In this example, the `use_existing_objects` parameter in the `ADD_MASTER_DATABASE` procedure is set to `TRUE` because it is assumed that the hr schema already exists at all master sites. In other words, it is assumed that the objects in the `hr` schema are precreated at all master sites. Also, the `copy_rows` parameter is set to `FALSE` because it is assumed that the identical data is stored in the tables at each master site.

   > **Note:** When adding a master site to a master group that contains tables with circular dependencies or a table that contains a self-referential constraint, you must precreate the table definitions and manually load the data at the new master site. The following is an example of a circular dependency: Table A has a foreign key constraint on table B, and table B has a foreign key constraint on table A.

```
*/
```

```
BEGIN
   DBMS_REPCAT.ADD_MASTER_DATABASE (
      gname => 'hr_repg',
      master => 'orc2.world',
      use_existing_objects => TRUE,
      copy_rows => FALSE,
      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 DBLINK FROM DBA_REPSITES WHERE GNAME = 'HR_REPG';
```

```
*/

PAUSE Press <RETURN> to continue.

BEGIN
   DBMS_REPCAT.ADD_MASTER_DATABASE (
      gname => 'hr_repg',
      master => 'orc3.world',
      use_existing_objects => TRUE,
      copy_rows => FALSE,
      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 `orc2.world` has appeared:

```
SELECT DBLINK FROM DBA_REPSITES WHERE GNAME = 'HR_REPG';
```

```
*/
```

```
PAUSE Press <RETURN> to continue.

/*
```

5. If conflicts are possible, then configure conflict resolution methods.

> **Caution:** If you added one or more tables to a master group during creation of the group, then 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 Also:** Chapter 6, "Configure Conflict Resolution" for information about configuring conflict resolution methods

```
*/

PAUSE Press <RETURN> to continue after configuring conflict resolution
methods or if no conflict resolution methods are required.

/*
```

6. Generate replication support.

```
*/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'countries',
      type => 'TABLE',
      min_communication => TRUE);
END;
/
```

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'departments',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'employees',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'jobs',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'job_history',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'locations',
      type => 'TABLE',
      min_communication => TRUE);
END;
/
```

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'regions',
      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 COUNT(*) FROM DBA_REPCATLOG WHERE GNAME = 'HR_REPG';
```

```
*/

PAUSE Press <RETURN> to continue.

/*
```

7. Resume replication.

After 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 procedure "turns on" replication for the specified master group.

See Also: "RESUME_MASTER_ACTIVITY Procedure" on page 20-100

```
*/

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/************************* END OF SCRIPT **********************************/
```

4

Create a Deployment Template

This chapter illustrates how to build a deployment template using the replication managment API. This chapter contains these topics:

- Oracle Deployment Templates Concepts
- Before Creating the Deployment Template
- Creating a Deployment Template
- Packaging a Deployment Template for Instantiation
- Instantiating a Deployment Template

Before you build materialized view environments, you must set up your master site, create a master group, and set up your intended materialized view sites. Also, if conflicts are possible at the master site due to activity at the materialized view sites you are creating, then configure conflict resolution for the master tables of the materialized views before you create the materialized view group.

See Also:

- "Setting Up Master Sites" on page 2-4
- "Overview of Creating a Master Group" on page 3-2
- "Setting Up Materialized View Sites" on page 2-19
- Chapter 6, "Configure Conflict Resolution"

Oracle Deployment Templates Concepts

Oracle offers deployment templates to allow the database administrator to package a materialized view environment for easy, custom, and secure distribution and installation. A deployment template can be simple (for example, it can contain a single materialized view with a fixed data set), or complex (for example, it can contain hundreds of materialized views 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 materialized view environment
- Data subsetting at remote sites using template parameters
- Authorized user list to control template instantiation and data access

To prepare a materialized view environment for deployment, the DBA creates a deployment template at the master site. This template stores all of the information needed to deploy a materialized view 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 materialized view creation.

You cannot use deployment templates to instantiate the following types of objects:

- User-defined types
- User-defined type bodies
- User-defined operators
- Indextypes

Nor can you use deployment templates to instantiate any objects based on these types of objects.

> **See Also:** *Oracle9i Replication* for more conceptual information about deployment templates

Before Creating the Deployment Template

If you want one of your master sites to support a materialized views that can be fast refreshed, then you need to create materialized view logs for each master table that is replicated to a materialized view.

The example in this chapter uses the `hr` sample schema. Enter the following to create materialized view logs for the tables in the `hr` schema:

```
CONNECT hr/hr@orc3.world

CREATE MATERIALIZED VIEW LOG ON hr.countries;
CREATE MATERIALIZED VIEW LOG ON hr.departments;
CREATE MATERIALIZED VIEW LOG ON hr.employees;
CREATE MATERIALIZED VIEW LOG ON hr.jobs;
CREATE MATERIALIZED VIEW LOG ON hr.job_history;
CREATE MATERIALIZED VIEW LOG ON hr.locations;
CREATE MATERIALIZED VIEW LOG ON hr.regions;
```

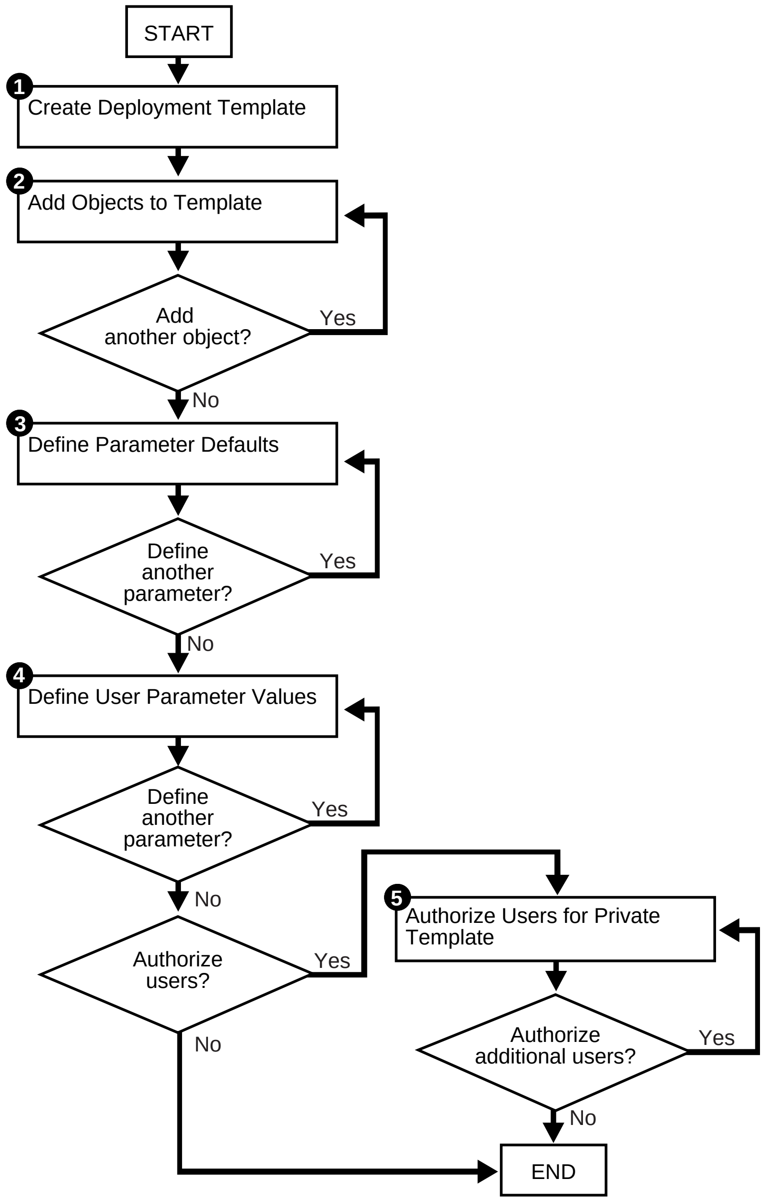
See Also: The `CREATE MATERIALIZED VIEW LOG` statement in the *Oracle9i SQL Reference* for detailed information about this SQL statement

Creating a Deployment Template

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

See Also: *Oracle9i Replication* for conceptual and architectural information about deployment templates

Figure 4–1 Creating a 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: You must use the Replication Management tool if you want to create materialized views with a subset of the columns their master tables. See *Oracle9i Replication* and the Replication Management tool online help for more information about column subsetting.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 4-12 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

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. Complete the following steps to build a template:

1. Create deployment template.

   Before 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).

```
*/

CONNECT repadmin/repadmin@orc3.world
```

```
DECLARE
   a NUMBER;
BEGIN
   a := DBMS_REPCAT_RGT.CREATE_REFRESH_TEMPLATE (
           owner => 'hr',
           refresh_group_name => 'hr_refg',
           refresh_template_name => 'hr_refg_dt',
           template_comment => 'Human Resources Deployment Template',
           public_template => 'N');
END;
/

/*
```

2. Add objects to template by completing the following steps.

   a. Create `countries_mv` materialized view.

```
*/

DECLARE
   tempstring VARCHAR2(3000);
   a NUMBER;
BEGIN
   tempstring := 'CREATE MATERIALIZED VIEW hr.countries_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      country_id, country_name, region_id
      FROM hr.countries@:dblink';
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           refresh_template_name => 'hr_refg_dt',
           object_name => 'countries_mv',
           object_type => 'SNAPSHOT',
           ddl_text => tempstring,
           master_rollback_seg => 'rbs');
END;
/

/*
```

Whenever you create a materialized view, always specify the schema name of the table owner in the query for the materialized view. In the example above, `hr` is specified as the owner of the `countries` table.

b. Create `departments_mv` materialized view.

```
*/

DECLARE
   tempstring VARCHAR2(3000);
   a NUMBER;
BEGIN
   tempstring := 'CREATE MATERIALIZED VIEW hr.departments_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      department_id, department_name, manager_id, location_id
      FROM hr.departments@:dblink';
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           refresh_template_name => 'hr_refg_dt',
           object_name => 'departments_mv',
           object_type => 'SNAPSHOT',
           ddl_text => tempstring,
           master_rollback_seg => 'rbs');
END;
/

/*
```

c. Create `employees_mv` materialized view.

```
*/

DECLARE
   tempstring VARCHAR2(3000);
   a NUMBER;
BEGIN
   tempstring := 'CREATE MATERIALIZED VIEW hr.employees_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      employee_id, first_name, last_name, email, phone_number,
      hire_date, job_id, salary, commission_pct, manager_id,
      department_id
      FROM hr.employees@:dblink WHERE department_id = :dept';
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           refresh_template_name => 'hr_refg_dt',
           object_name => 'employees_mv',
           object_type => 'SNAPSHOT',
           ddl_text => tempstring,
           master_rollback_seg => 'rbs');
END;
/
```

```
/*
```

d. Create `jobs_mv` materialized view.

```
*/

DECLARE
   tempstring VARCHAR2(3000);
   a NUMBER;
BEGIN
   tempstring := 'CREATE MATERIALIZED VIEW hr.jobs_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      job_id, job_title, min_salary, max_salary
      FROM hr.jobs@:dblink';
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           refresh_template_name => 'hr_refg_dt',
           object_name => 'jobs_mv',
           object_type => 'SNAPSHOT',
           ddl_text => tempstring,
           master_rollback_seg => 'rbs');
END;
/

/*
```

e. Create `job_history_mv` materialized view.

```
*/

DECLARE
   tempstring VARCHAR2(3000);
   a NUMBER;
BEGIN
   tempstring := 'CREATE MATERIALIZED VIEW hr.job_history_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      employee_id, start_date, end_date, job_id, department_id
      FROM hr.job_history@:dblink';
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           refresh_template_name => 'hr_refg_dt',
           object_name => 'job_history_mv',
           object_type => 'SNAPSHOT',
           ddl_text => tempstring,
           master_rollback_seg => 'rbs');
 END;
/
```

```
/*
```

f. Create `locations_mv` materialized view.

```
*/

DECLARE
   tempstring VARCHAR2(3000);
   a NUMBER;
BEGIN
   tempstring := 'CREATE MATERIALIZED VIEW hr.locations_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      location_id, street_address, postal_code, city,
      state_province, country_id
      FROM hr.locations@:dblink';
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           refresh_template_name => 'hr_refg_dt',
           object_name => 'locations_mv',
           object_type => 'SNAPSHOT',
           ddl_text => tempstring,
           master_rollback_seg => 'rbs');
END;
/

/*
```

g. Create `regions_mv` materialized view.

```
*/

DECLARE
   tempstring VARCHAR2(3000);
   a NUMBER;
BEGIN
   tempstring := 'CREATE MATERIALIZED VIEW hr.regions_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      region_id, region_name
      FROM hr.regions@:dblink';
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
           refresh_template_name => 'hr_refg_dt',
           object_name => 'regions_mv',
           object_type => 'SNAPSHOT',
           ddl_text => tempstring,
           master_rollback_seg => 'rbs');
END;
/
```

```
/*
```

3. Define parameter defaults.

Rather than using the CREATE_* functions and procedures as in the other steps, 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 1b and 1c. Recall that you defined the :dblink and :dept template 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).

Complete the following steps to define parameter defaults.

a. Define the default value for the dept parameter.

```
*/

BEGIN
   DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM (
      refresh_template_name => 'hr_refg_dt',
      parameter_name => 'dept',
      new_default_parm_value => '30',
      new_prompt_string => 'Enter your department number:',
      new_user_override => 'Y');
END;
/

/*
```

b. Define the default value for the dblink parameter.

```
*/

BEGIN
   DBMS_REPCAT_RGT.ALTER_TEMPLATE_PARM (
      refresh_template_name => 'hr_refg_dt',
      parameter_name => 'dblink',
      new_default_parm_value => 'orc3.world',
      new_prompt_string => 'Enter your master site:',
      new_user_override => 'Y');
END;
/
```

```
/*
```

4. Define user parameter values.

To automate the instantiation of custom data sets at individual remote materialized view 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.

Complete the following steps to define user parameter values.

a. Define `dept` user parameter value for user `hr`.

```
*/

DECLARE
   a NUMBER;
BEGIN
   a := DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE (
           refresh_template_name => 'hr_refg_dt',
           parameter_name => 'dept',
           user_name => 'hr',
           parm_value => '20');
END;
/

/*
```

b. Define `dblink` user parameter value for user `hr`.

```
*/

DECLARE
   a NUMBER;
BEGIN
   a := DBMS_REPCAT_RGT.CREATE_USER_PARM_VALUE (
           refresh_template_name => 'hr_refg_dt',
           parameter_name => 'dblink',
           user_name => 'hr',
           parm_value => 'orc3.world');
END;
/

/*
```

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 on page 4-3), you need to authorize users to instantiate the `dt_personnel` deployment template. Use the `CREATE_USER_AUTHORIZATION` function in the `DBMS_REPCAT_RGT` package to create authorized users.

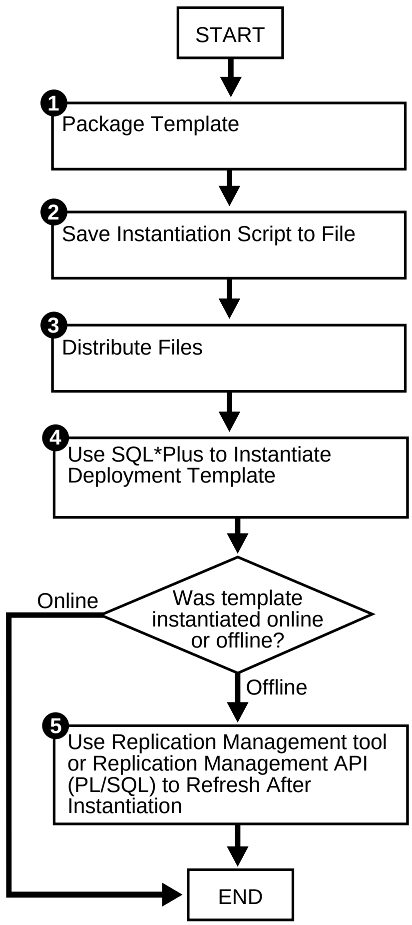
```
*/

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

COMMIT;
```

```
/************************* END OF SCRIPT **********************************/
```

Packaging a Deployment Template for Instantiation

After 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 describes two tasks: create the instantiation script and save the instantiation script to a file.

Figure 4–2 Packaging and Instantiating a Deployment Template



Packaging a Deployment Template

When you execute either the INSTANTIATE_OFFLINE or the INSTANTIATE_ONLINE function, Oracle populates the USER_REPCAT_TEMP_OUTPUT data dictionary view with the script to create the remote materialized view 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 materialized view site connects to the master site to download the data.

Complete the steps in either the "Packaging a Deployment Template for Offline Instantiation" or "Packaging a Deployment Template for Online Instantiation" according to your needs. These sections only apply to packaging templates for materialized view sites running Oracle Enterprise Edition, Oracle Standard Edition, or Oracle Personal Edition.

Note: In you need to execute either the INSTANTIATE_OFFLINE or the INSTANTIATE_ONLINE function more than once for a particular materialized view site, then run the DROP_SITE_INSTANTIATION procedure in the DBMS_REPCAT_RGT package before you attempt to repackage a template for the site. Otherwise, Oracle returns an error stating that there is a duplicate template site.

Packaging a Deployment Template for Offline Instantiation

The INSTANTIATE_OFFLINE function creates a script that creates the materialized view environment according to the contents of a specified deployment template. In addition to containing the DDL (CREATE statements) to create the materialized view environment, this script also contains the DML (INSERT statements) to populate the materialized view 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, then you must create a loopback database link.

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

CONNECT repadmin/repadmin@orc3.world

SET SERVEROUTPUT ON
DECLARE
   dt_num NUMBER;
BEGIN
   dt_num := DBMS_REPCAT_RGT.INSTANTIATE_OFFLINE(
               refresh_template_name => 'hr_refg_dt',
               user_name => 'hr',
               site_name => 'sf.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` data dictionary view to retrieve the generated script. Be sure that you complete the steps in "Saving an Instantiation Script to File" on page 4-16 after you complete this section. This script is unique to an individual materialized view site and cannot be used for other materialized view sites.

Packaging a Deployment Template for Online Instantiation

The `INSTANTIATE_ONLINE` function creates a script that creates the materialized view environment according to the contents of a specified deployment template. When this script is executed at the remote materialized view site, Oracle creates the materialized view 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 materialized view site has a "live" connection to the master site.

> **See Also:** *Oracle9i Replication* for additional materialized view site requirements

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

CONNECT repadmin/repadmin@orc3.world

SET SERVEROUTPUT ON
DECLARE
   dt_num NUMBER;
BEGIN
   dt_num := DBMS_REPCAT_RGT.INSTANTIATE_ONLINE(
               refresh_template_name => 'hr_refg_dt',
               user_name => 'hr',
               site_name => 'sf.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` data dictionary view to retrieve the generated script. Be sure that you complete the steps in "Saving an Instantiation Script to File" after you complete this task.

Saving an Instantiation Script to File

The best way to save the contents of the `USER_REPCAT_TEMP_OUTPUT` data dictionary view is to use the `UTL_FILE` package to save the contents of the `TEXT` column in the `USER_REPCAT_TEMP_OUTPUT` view to a file.

Note: The following action 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` data dictionary view are temporary. If you have not completed the steps in "Packaging a Deployment Template" on page 4-14, then do so now and then complete the following action.

> **See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `UTL_FILE` package

Enter the following to save the deployment template script to a file.

```
DECLARE
  fh UTL_FILE.FILE_TYPE;
  CURSOR ddlcursor(myid NUMBER) IS
  SELECT TEXT FROM USER_REPCAT_TEMP_OUTPUT WHERE OUTPUT_ID = myid ORDER BY LINE;
BEGIN
  fh := UTL_FILE.FOPEN ('file_location', 'file_name', 'w');
  UTL_FILE.PUT_LINE (fh, 'SET ECHO OFF;');
  FOR myrec IN ddlcursor(template_id) LOOP
    UTL_FILE.PUT_LINE(fh, myrec.text);
  END LOOP;
  UTL_FILE.PUT_LINE (fh, 'SET ECHO ON;');
  UTL_FILE.FFLUSH(fh);
  UTL_FILE.FCLOSE(fh);
END;
/
```

Notice that *`file_location`*, *`file_name`*, and *`template_id`* are placeholders. Substitute the correct values for your environment:

- Replace the *`file_location`* placeholder with the full directory path where you want to save the template script.

> **Note:** The location you specify for the template script must be a location listed in the `UTL_FILE_DIR` initialization parameter. If you specify a location that is not listed in the `UTL_FILE_DIR` initialization parameter, then Oracle returns errors when you try to save the template script to a file. See *Oracle9i Replication* for more information about `UTL_FILE_DIR`.

- Replace the *`file_name`* placeholder with name you want to use for the template script.
- Replace the *`template_id`* placeholder with the number returned by the `INSTANTIATE_OFFLINE` or `INSTANTIATE_ONLINE` function when you packaged the template previously.

For example, suppose you have the following values:

Placeholder	Value
file_location	/home/gen_files/
file_name	sf.sql
template_id	18

Given these values, connect to the master site as the replication administrator and run the following procedure to save the template script to a file:

```
CONNECT repadmin/repadmin@orc3.world

DECLARE
  fh UTL_FILE.FILE_TYPE;
  CURSOR ddlcursor(myid NUMBER) IS
  SELECT TEXT FROM USER_REPCAT_TEMP_OUTPUT WHERE OUTPUT_ID = myid
  ORDER BY LINE;
BEGIN
  fh := UTL_FILE.FOPEN ('/home/gen_files/', 'sf.sql', 'w');
  UTL_FILE.PUT_LINE (fh, 'SET ECHO OFF;');
  FOR myrec IN ddlcursor(18) LOOP
    UTL_FILE.PUT_LINE(fh, myrec.text);
  END LOOP;
  UTL_FILE.PUT_LINE (fh, 'SET ECHO ON;');
  UTL_FILE.FFLUSH(fh);
  UTL_FILE.FCLOSE(fh);
END;
/
```

Distributing Instantiation Files

After creating the instantiation script and saving it to a file, you must distribute this file to the remote materialized view 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.

Instantiating a Deployment Template

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

> **See Also:** *Oracle9i Replication* for materialized view site requirements that must be met before instantiating your deployment template

The following script demonstrates how to complete the instantiation process at a remote materialized view site with Oracle Enterprise Edition, Oracle Standard Edition, or Oracle Personal Edition installed.

> **Note:** If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 4-21 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. If it does not already exist, create the schema at the materialized view site.

   Before executing the instantiation script at the remote materialized view site, you must create the schema that contains the replicated objects.

```
*/

CONNECT system/manager@sf.world

CREATE TABLESPACE demo_mv
 DATAFILE 'demo_mv.dbf' SIZE 10M AUTOEXTEND ON
 EXTENT MANAGEMENT LOCAL AUTOALLOCATE;

CREATE TEMPORARY TABLESPACE temp_mv
 TEMPFILE 'temp_mv.dbf' SIZE 5M AUTOEXTEND ON;

CREATE USER hr IDENTIFIED BY hr;

ALTER USER hr DEFAULT TABLESPACE demo_mv
              QUOTA UNLIMITED ON demo_mv;

ALTER USER hr TEMPORARY TABLESPACE temp_mv;
```

```
GRANT
  CREATE SESSION,
  CREATE TABLE,
  CREATE PROCEDURE,
  CREATE SEQUENCE,
  CREATE TRIGGER,
  CREATE VIEW,
  CREATE SYNONYM,
  ALTER SESSION,
  CREATE MATERIALIZED VIEW,
  ALTER ANY MATERIALIZED VIEW,
  CREATE DATABASE LINK
 TO hr;

/*
```

2. If they do not already exist, create the database links for the replicated schema.

   Before instantiating the deployment template, you must make sure that the necessary database links exist for the replicated schema. The owner of the materialized views needs a database link pointing to the `proxy_refresher` that was created when the master site was set up.

```
*/

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

CONNECT hr/hr@sf.world

CREATE DATABASE LINK orc3.world
   CONNECT TO proxy_refresher IDENTIFIED BY proxy_refresher;

/*
```

See Also: Step 7 on page 2-7 for more information about creating proxy master site users

3. Execute the instantiation script.

```
*/

@d:\sf.sql

/*
```

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

```
/************************ END OF SCRIPT **********************************/
```

Refreshing a Refresh Group After Instantiation

If you have just instantiated a deployment template using the offline instantiation method, then you should perform a refresh of the refresh group as soon as possible by issuing the following execute statement:

```
CONNECT hr/hr@sf.world

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

5

Create Materialized View Group

This chapter illustrates how to create a materialized view group at a remote materialized view replication site. This chapter contains these topics:

Before you build materialized view environments, you must set up your master site, create a master group, and set up your intended materialized view sites. Also, if conflicts are possible at the master site due to activity at the materialized view sites you are creating, then configure conflict resolution for the master tables of the materialized views before you create the materialized view group.

See Also:

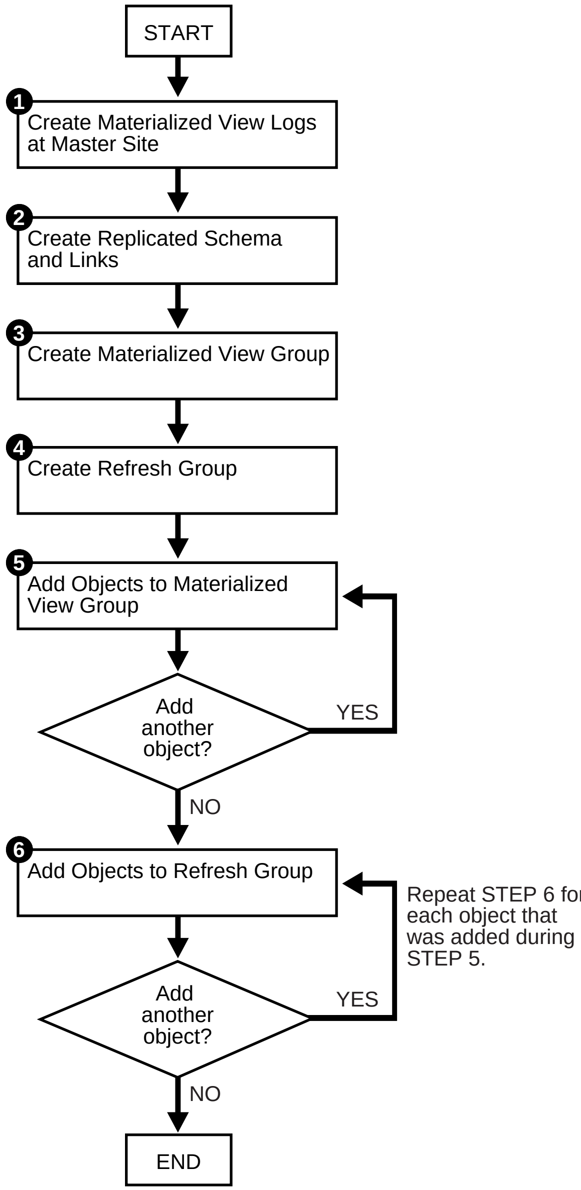
- "Setting Up Master Sites" on page 2-4
- "Overview of Creating a Master Group" on page 3-2
- "Setting Up Materialized View Sites" on page 2-19
- Chapter 6, "Configure Conflict Resolution"

Overview of Creating a Materialized View Group

After setting up your materialized view site and creating at least one master group, you are ready to create a materialized view group at a remote materialized view site. Figure 5–1 illustrates the process of creating a materialized view group.

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

Figure 5–1 Creating a Materialized View Group



Creating a Materialized View Group

This chapter guides you through the process of creating two materialized view groups at two different materialized view sites: `mv1.world` and `mv2.world`:

- The materialized view group at `mv1.world` is based on the objects in the `hr_repg` master group at the `orc1.world` master site.
- The materialized view group at `mv2.world` is based on the objects in the `hr_repg` materialized view group at the `mv1.world` materialized view site.

Therefore, the examples in this chapter illustrate how to create a multitier materialized view environment, where one or more materialized views are based on other materialized views.

Complete the following steps to create these two materialized view groups.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 5-19 into a text editor and then edit the text to create a script for your environment.

```
/************************ BEGINNING OF SCRIPT *****************************
```

Creating the Materialized View Group at mv1.world

Complete the following steps to create the hr_repg materialized view group at the `mv1.world` materialized view site. This materialized view group is based on the `hr_repg` master group at the `orc1.world` master site.

1. Create materialized view logs at the master site.

   If you want one of your master sites to support a materialized view site, then you need to create materialized view logs for each master table that is replicated to a materialized view. Recall from Figure 2–1 on page 2-2 that `orc1.world` serves as the target master site for the `mv1.world` materialized view site. The required materialized view logs must be created at `orc1.world`.

   ```
   */

   CONNECT hr/hr@orc1.world

   CREATE MATERIALIZED VIEW LOG ON hr.countries;
   CREATE MATERIALIZED VIEW LOG ON hr.departments;
   CREATE MATERIALIZED VIEW LOG ON hr.employees;
   ```

```
CREATE MATERIALIZED VIEW LOG ON hr.jobs;
CREATE MATERIALIZED VIEW LOG ON hr.job_history;
CREATE MATERIALIZED VIEW LOG ON hr.locations;
CREATE MATERIALIZED VIEW LOG ON hr.regions;

/*
```

> **See Also:** The CREATE MATERIALIZED VIEW LOG statement in the *Oracle9i SQL Reference* for detailed information about this SQL statement

2. If they do not already exist, then create the replicated schema its database link by completing the following steps. Before building your materialized view group, you must make sure that the replicated schema exists at the remote materialized view site and that the necessary database links have been created.

   a. If the hr schema does not exist, then create the schema. For this example, if the hr schema already exists at the materialized view site, then go to Step b.

   ```
   */

   CONNECT system/manager@mv1.world

   CREATE TABLESPACE demo_mv1
    DATAFILE 'demo_mv1.dbf' SIZE 10M AUTOEXTEND ON
    EXTENT MANAGEMENT LOCAL AUTOALLOCATE;

   CREATE TEMPORARY TABLESPACE temp_mv1
    TEMPFILE 'temp_mv1.dbf' SIZE 5M AUTOEXTEND ON;

   CREATE USER hr IDENTIFIED BY hr;

   ALTER USER hr DEFAULT TABLESPACE demo_mv1
                 QUOTA UNLIMITED ON demo_mv1;

   ALTER USER hr TEMPORARY TABLESPACE temp_mv1;

   GRANT
     CREATE SESSION,
     CREATE TABLE,
     CREATE PROCEDURE,
     CREATE SEQUENCE,
     CREATE TRIGGER,
     CREATE VIEW,
     CREATE SYNONYM,
   ```

```
  ALTER SESSION,
  CREATE MATERIALIZED VIEW,
  ALTER ANY MATERIALIZED VIEW,
  CREATE DATABASE LINK
 TO hr;

/*
```

b. If it does not already exist, then create the database link for the replicated schema.

Before building your materialized view group, you must make sure that the necessary database links exist for the replicated schema. The owner of the materialized views needs a database link pointing to the `proxy_refresher` that was created when the master site was set up.

```
*/

CONNECT hr/hr@mv1.world

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

/*
```

See Also: Step 7 on page 2-7 for more information about creating proxy master site users

3. Create the materialized view group.

The following procedures must be executed by the materialized view administrator at the remote materialized view site.

```
*/

CONNECT mviewadmin/mviewadmin@mv1.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_MVIEW_REPGROUP (
      gname => 'hr_repg',
      master => 'orc1.world',
      propagation_mode => 'ASYNCHRONOUS');
END;
/

/*
```

4. Create the refresh group.

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

```
*/

BEGIN
   DBMS_REFRESH.MAKE (
      name => 'mviewadmin.hr_refg',
      list => '',
      next_date => SYSDATE,
      interval => 'SYSDATE + 1/24',
      implicit_destroy => FALSE,
      rollback_seg => '',
      push_deferred_rpc => TRUE,
      refresh_after_errors => FALSE);
END;
/

/*
```

5. Add objects to the materialized view group by completing the following steps.

   a. Create the materialized views based on the master tables.

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

```
*/

CREATE MATERIALIZED VIEW hr.countries_mv1
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.countries@orc1.world;
```

```
CREATE MATERIALIZED VIEW hr.departments_mv1
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.departments@orc1.world;

CREATE MATERIALIZED VIEW hr.employees_mv1
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.employees@orc1.world;

CREATE MATERIALIZED VIEW hr.jobs_mv1
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.jobs@orc1.world;

CREATE MATERIALIZED VIEW hr.job_history_mv1
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.job_history@orc1.world;

CREATE MATERIALIZED VIEW hr.locations_mv1
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.locations@orc1.world;

CREATE MATERIALIZED VIEW hr.regions_mv1
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.regions@orc1.world;

/*
```

b. Add the objects to the materialized view group.

```
*/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'countries_mv1',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'departments_mv1',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'employees_mv1',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'jobs_mv1',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'job_history_mv1',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'locations_mv1',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'regions_mv1',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/

/*
```

6. Add objects to refresh group.

   All of the materialized view group objects that you add to the refresh group are refreshed at the same time to preserve referential integrity between related materialized views.

```
*/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.countries_mv1',
      lax => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.departments_mv1',
      lax => TRUE);
END;
/
```

```
BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.employees_mv1',
      lax => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.jobs_mv1',
      lax => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.job_history_mv1',
      lax => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.locations_mv1',
      lax => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.regions_mv1',
      lax => TRUE);
END;
/

/*
```

Creating the Materialized View Group at mv2.world

Complete the following steps to create the hr_repg materialized view group at the `mv2.world` materialized view site. This materialized view group is based on the `hr_repg` materialized view group at the `mv1.world` materialized view site.

1. Create materialized view logs at the master materialized view site.

   If you want one of your master materialized view sites to support another materialized view site, then you need to create materialized view logs for each materialized view that is replicated to another materialized view site. Recall from Figure 2–1 on page 2-2 that `mv1.world` serves as the target master internalized view site for the `mv2.world` materialized view site. The required materialized view logs must be created at `mv1.world`.

   ```
   */

   CONNECT hr/hr@mv1.world

   CREATE MATERIALIZED VIEW LOG ON hr.countries_mv1;
   CREATE MATERIALIZED VIEW LOG ON hr.departments_mv1;
   CREATE MATERIALIZED VIEW LOG ON hr.employees_mv1;
   CREATE MATERIALIZED VIEW LOG ON hr.jobs_mv1;
   CREATE MATERIALIZED VIEW LOG ON hr.job_history_mv1;
   CREATE MATERIALIZED VIEW LOG ON hr.locations_mv1;
   CREATE MATERIALIZED VIEW LOG ON hr.regions_mv1;

   /*
   ```

   > **See Also:** The `CREATE MATERIALIZED VIEW LOG` statement in the *Oracle9i SQL Reference* for detailed information about this SQL statement

2. If they do not already exist, then create the replicated schema its database link by completing the following steps. Before building your materialized view group, you must make sure that the replicated schema exists at the remote materialized view site and that the necessary database links have been created.

   a. For this example, if the hr schema does not exist, then create the schema. If the hr schema already exists at the materialized view site, then go to Step b.

      ```
      */

      CONNECT system/manager@mv2.world
      ```

```
CREATE TABLESPACE demo_mv2
 DATAFILE 'demo_mv2.dbf' SIZE 10M AUTOEXTEND ON
 EXTENT MANAGEMENT LOCAL AUTOALLOCATE;

CREATE TEMPORARY TABLESPACE temp_mv2
 TEMPFILE 'temp_mv2.dbf' SIZE 5M AUTOEXTEND ON;

CREATE USER hr IDENTIFIED BY hr;

ALTER USER hr DEFAULT TABLESPACE demo_mv2
              QUOTA UNLIMITED ON demo_mv2;

ALTER USER hr TEMPORARY TABLESPACE temp_mv2;

GRANT
  CREATE SESSION,
  CREATE TABLE,
  CREATE PROCEDURE,
  CREATE SEQUENCE,
  CREATE TRIGGER,
  CREATE VIEW,
  CREATE SYNONYM,
  ALTER SESSION,
  CREATE MATERIALIZED VIEW,
  ALTER ANY MATERIALIZED VIEW,
  CREATE DATABASE LINK
 TO hr;

/*
```

b. If it does not already exist, then create the database link for the replicated schema.

Before building your materialized view group, you must make sure that the necessary database links exist for the replicated schema. The owner of the materialized views needs a database link pointing to the `proxy_refresher` that was created when the master materialized view site was set up.

```
*/

CONNECT hr/hr@mv2.world

CREATE DATABASE LINK mv1.world
   CONNECT TO proxy_refresher IDENTIFIED BY proxy_refresher;
```

```
/*
```

See Also: Step 6 on page 2-25 for more information about creating proxy master materialized view site users

3. Create the materialized view group.

   The following procedures must be executed by the materialized view administrator at the remote materialized view site.

```
*/

CONNECT mviewadmin/mviewadmin@mv2.world

/*
```

   The replication group that you specify in the `gname` parameter must match the name of the replication group that you are replicating at the target master materialized view site.

```
*/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPGROUP (
      gname => 'hr_repg',
      master => 'mv1.world',
      propagation_mode => 'ASYNCHRONOUS');
END;
/

/*
```

4. Create the refresh group.

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

```
*/
```

```
BEGIN
   DBMS_REFRESH.MAKE (
      name => 'mviewadmin.hr_refg',
      list => '',
      next_date => SYSDATE,
      interval => 'SYSDATE + 1/24',
      implicit_destroy => FALSE,
      rollback_seg => '',
      push_deferred_rpc => TRUE,
      refresh_after_errors => FALSE);
END;
/

/*
```

5. Add objects to the materialized view group by completing the following steps.

   a. Create the materialized views based on the master materialized views.

      Whenever you create a materialized view that is based on another materialized view, always specify the schema name of the materialized view owner in the query for the materialized view. In the examples below, `hr` is specified as the owner of the materialized view in each query.

```
*/

CREATE MATERIALIZED VIEW hr.countries_mv2
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.countries_mv1@mv1.world;

CREATE MATERIALIZED VIEW hr.departments_mv2
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.departments_mv1@mv1.world;

CREATE MATERIALIZED VIEW hr.employees_mv2
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.employees_mv1@mv1.world;

CREATE MATERIALIZED VIEW hr.jobs_mv2
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.jobs_mv1@mv1.world;

CREATE MATERIALIZED VIEW hr.job_history_mv2
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.job_history_mv1@mv1.world;
```

```
CREATE MATERIALIZED VIEW hr.locations_mv2
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.locations_mv1@mv1.world;

CREATE MATERIALIZED VIEW hr.regions_mv2
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.regions_mv1@mv1.world;

/*
```

b. Add the materialized views to the materialized view group.

```
*/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'countries_mv2',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'departments_mv2',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'employees_mv2',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'jobs_mv2',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'job_history_mv2',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'locations_mv2',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'regions_mv2',
      type => 'SNAPSHOT',
      min_communication => TRUE);
END;
/

/*
```

6. Add objects to refresh group.

   All of the materialized view group objects that you add to the refresh group are refreshed at the same time to preserve referential integrity between related materialized views.

```
*/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.countries_mv2',
      lax => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.departments_mv2',
      lax => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.employees_mv2',
      lax => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.jobs_mv2',
      lax => TRUE);
END;
/
```

```
BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.job_history_mv2',
      lax => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.locations_mv2',
      lax => TRUE);
END;
/

BEGIN
   DBMS_REFRESH.ADD (
      name => 'mviewadmin.hr_refg',
      list => 'hr.regions_mv2',
      lax => TRUE);
END;
/

/************************* END OF SCRIPT **********************************/
```

6

Configure Conflict Resolution

This chapter illustrates how to define conflict resolution methods for your replication environment. This chapter contains these topics:

- Preparing for Conflict Resolution
- Creating Conflict Resolution Methods for Update Conflicts
- Creating Conflict Resolution Methods for Uniqueness Conflicts
- Creating Conflict Avoidance Methods for Delete Conflicts
- Using Dynamic Ownership Conflict Avoidance
- Auditing Successful Conflict Resolution

Preparing 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 replication 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 replication 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, then you have the option of building and using your own conflict resolution methods.

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.

After 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 have precedence over another?

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.

> **See Also:** *Oracle9i Replication* for conceptual information about conflict resolution methods and detailed information about data convergence for each method

Creating Conflict Resolution Methods for Update Conflicts

The most common data conflict occurs when the same row at two or more different sites are 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 replication 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 receives two or more updates.

Creating Conflict Resolution Methods and Quiescing: The instructions in the following sections specify that you must quiesce your master group to add conflict resolution methods. However, if your master site is running Oracle release 8.1.7 or higher in a single master environment, then you may not need to quiesce the master group to add conflict resolution methods. See the "What's New in Replication?" section at the beginning of *Oracle9i Replication* for information about when quiesce is not required.

Overwrite and Discard Conflict Resolution Methods

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 materialized view 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 20-20 and *Oracle9i Replication* for more information about overwrite and discard

---

Note: This section uses objects not found in the other scripts within this book, because the configuration `orc1.world`, `orc2.world`, `orc3.world`, and `mview1.world` contains three master sites and one materialized view site and is not appropriate for overwrite and discard.

Complete the following steps to create an overwrite or discard conflict resolution method.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 6-6 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect as the replication administrator. The procedures in the following steps must be executed by the replication administrator.

```
*/

CONNECT repadmin/repadmin@saturn.universe

/*
```

2. Before you define overwrite or discard conflict resolution methods, quiesce the master group that contains the table to which you want to apply the conflict resolution method. In a single master replication environment, quiescing the master group may not be required. See "Creating Conflict Resolution Methods and Quiescing" on page 6-3 for more information.

```
*/

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

/*
```

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

```
*/
```

```
BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      sname => 'titan',
      oname => 'planet',
      column_group => 'planet_cg1',
      list_of_column_names => 'order,circumference,moons');
END;
/

/*
```

4. Use the `DBMS_REPCAT.ADD_UPDATE_RESOLUTION` procedure 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;
/

/*
```

5. 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;
/

/*
```

6. Resume master activity after replication support has been regenerated.

```
*/

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

/************************* END OF SCRIPT **********************************/
```

Minimum and Maximum Conflict Resolution Methods

When Oracle Replication 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, then 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.

Complete the following steps to create an maximum or minimum conflict resolution method.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 6-8 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect as the replication administrator. The procedures in the following steps must be executed by the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

2. Before you define maximum or minimum conflict resolution methods, quiesce the master group that contains the table to which you want to apply the conflict resolution method. In a single master replication environment, quiescing the master group may not be required. See "Creating Conflict Resolution Methods and Quiescing" on page 6-3 for more information.

```
*/

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/*
```

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

```
*/

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      sname => 'hr',
      oname => 'jobs',
      column_group => 'job_minsal_cg',
      list_of_column_names => 'min_salary');
END;
/

/*
```

4. Use the `DBMS_REPCAT.ADD_UPDATE_RESOLUTION` procedure 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 => 'hr',
      oname => 'jobs',
      column_group => 'job_minsal_cg',
      sequence_no => 1,
      method => 'MINIMUM',
      parameter_column_name => 'min_salary');
END;
/

/*
```

5. Regenerate replication support for the table that received the conflict resolution method.

```
*/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'jobs',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

/*
```

6. Resume replication activity by using the `RESUME_MASTER_ACTIVITY` procedure.

```
*/

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/************************* END OF SCRIPT **********************************/
```

Timestamp Conflict Resolution Methods

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.

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

- Synchronized time settings between computers
- Timestamp field and trigger to automatically record timestamp

Complete the following steps to create a timestamp conflict resolution method.

> **Note:** If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 6-13 into a text editor and then edit the text to create a script for your environment.

```
/************************ BEGINNING OF SCRIPT *****************************
```

1. Connect as the replication administrator. The procedures in the following steps must be executed by the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

2. Before defining timestamp conflict resolution methods, quiesce the master group that contains the table to which you want to apply the conflict resolution method. In a single master replication environment, quiescing the master group may not be required. See "Creating Conflict Resolution Methods and Quiescing" on page 6-3 for more information.

```
*/
```

```
BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/*
```

3. 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. You must use the `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 => 'hr',
      oname => 'employees',
      type => 'TABLE',
      ddl_text => 'ALTER TABLE hr.employees ADD (timestamp DATE)');
END;
/

/*
```

4. Regenerate replication support for altered table.

```
*/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'employees',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

/*
```

5. Create a trigger that records the timestamp 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 => 'hr_repg',
      type => 'TRIGGER',
      oname => 'insert_time',
      sname => 'hr',
      ddl_text => 'CREATE TRIGGER hr.insert_time
                      BEFORE
                         INSERT OR UPDATE ON hr.employees FOR EACH ROW
                      BEGIN
                         IF DBMS_REPUTIL.FROM_REMOTE = FALSE THEN
                            :NEW.TIMESTAMP := SYSDATE;
                         END IF;
                      END;');
END;
/

/*
```

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

```
*/

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      sname => 'hr',
      oname => 'employees',
      column_group => 'employees_timestamp_cg',
      list_of_column_names => 'manager_id,hire_date,salary,timestamp');
END;
/

/*
```

7. Use the DBMS_REPCAT.ADD_UPDATE_RESOLUTION procedure 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 => 'hr',
      oname => 'employees',
      column_group => 'employees_timestamp_cg',
      sequence_no => 1,
      method => 'LATEST TIMESTAMP',
      parameter_column_name => 'timestamp');
END;
/

/*
```

8. Regenerate replication support for the table that received the conflict resolution method.

```
*/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'employees',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

/*
```

9. Resume replication activity by using the RESUME_MASTER_ACTIVITY procedure.

```
*/

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/
```

```
/************************* END OF SCRIPT **********************************/
```

Additive and Average Conflict Resolution Methods

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.

Complete the following steps to create an additive or average conflict resolution method.

> **Note:** If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 6-15 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect as the replication administrator. The procedures in the following steps must be executed by the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

2. Before you define additive and average conflict resolution methods, quiesce the master group that contains the table to which you want to apply the conflict resolution method. In a single master replication environment, quiescing the master group may not be required. See "Creating Conflict Resolution Methods and Quiescing" on page 6-3 for more information.

```
*/

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      gname => 'scott_repg');
END;
/
```

```
/*
```

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

```
*/

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      sname => 'scott',
      oname => 'bonus',
      column_group => 'bonus_sal_cg',
      list_of_column_names => 'sal');
END;
/

/*
```

4. Use the `DBMS_REPCAT.ADD_UPDATE_RESOLUTION` procedure 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_sal_cg',
      sequence_no => 1,
      method => 'ADDITIVE',
      parameter_column_name => 'sal');
END;
/

/*
```

5. 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;
/

/*
```

6. Resume replication activity by using the RESUME_MASTER_ACTIVITY procedure.

```
*/

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      gname => 'scott_repg');
END;
/

/************************* END OF SCRIPT **********************************/
```

Priority Groups Conflict Resolution Methods

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

Complete the following steps to create a priority groups conflict resolution method.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 6-19 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect as the replication administrator. The procedures in the following steps must be executed by the replication administrator.

   ```
   */

   CONNECT repadmin/repadmin@orc1.world

   /*
   ```

2. Before you define a priority groups conflict resolution method, quiesce the master group that contains the table to which you want to apply the conflict resolution method. In a single master replication environment, quiescing the master group may not be required. See "Creating Conflict Resolution Methods and Quiescing" on page 6-3 for more information.

   ```
   */

   BEGIN
      DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
         gname => 'hr_repg');
   END;
   /

   /*
   ```

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

   ```
   */
   ```

```
BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      sname => 'hr',
      oname => 'employees',
      column_group => 'employees_priority_cg',
      list_of_column_names => 'manager_id,hire_date,salary,job_id');
END;
/

/*
```

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

```
*/

BEGIN
   DBMS_REPCAT.DEFINE_PRIORITY_GROUP (
      gname => 'hr_repg',
      pgroup => 'job_pg',
      datatype => 'VARCHAR2');
END;
/

/*
```

5. The `DBMS_REPCAT.ADD_PRIORITY_`*`datatype`* procedure is available in several different versions. There is a version for each available datatype (`NUMBER`, `VARCHAR2`, and so on). Execute this procedure as often as necessary until you have defined a priority value for all possible table values.

   See Also: "ADD_PRIORITY_datatype Procedure" on page 20-17 for more information

```
*/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      gname => 'hr_repg',
      pgroup => 'job_pg',
      value => 'ad_pres',
      priority => 100);
END;
/
```

```
BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      gname => 'hr_repg',
      pgroup => 'job_pg',
      value => 'sa_man',
      priority => 80);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      gname => 'hr_repg',
      pgroup => 'job_pg',
      value => 'sa_rep',
      priority => 60);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      gname => 'hr_repg',
      pgroup => 'job_pg',
      value => 'pu_clerk',
      priority => 40);
END;
/

BEGIN
   DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
      gname => 'hr_repg',
      pgroup => 'job_pg',
      value => 'st_clerk',
      priority => 20);
END;
/

/*
```

6. Add the PRIORITY GROUP resolution method to your replicated table. The following example shows that it is the second conflict resolution method for the specified column group (sequence_no parameter).

```
*/

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      sname => 'hr',
      oname => 'employees',
      column_group => 'employees_priority_cg',
      sequence_no => 2,
      method => 'PRIORITY GROUP',
      parameter_column_name => 'job_id',
      priority_group => 'job_pg');
END;
/

/*
```

7. Regenerate replication support for the table that received the conflict resolution method.

```
*/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'employees',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

/*
```

8. Resume replication activity by using the RESUME_MASTER_ACTIVITY procedure.

```
    */

    BEGIN
       DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
          gname => 'hr_repg');
    END;
    /
/************************* END OF SCRIPT **********************************/
```

Site Priority Conflict Resolution Methods

Site priority is a specialized form of a priority group. 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, then the value from `orc2.world` is used.

Complete the following steps to create a site priority conflict resolution method.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 6-25 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect as the replication administrator. The procedures in the following steps must be executed by the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

2. Before you define a site priority conflict resolution method, quiesce the master group that contains the table to which you want to apply the conflict resolution method. In a single master replication environment, quiescing the master group may not be required. See "Creating Conflict Resolution Methods and Quiescing" on page 6-3 for more information.

```
*/

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/*
```

3. Add a `site` column to your table to store the site value. 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 => 'hr',
      oname => 'regions',
      type => 'TABLE',
      ddl_text => 'ALTER TABLE hr.regions ADD (site VARCHAR2(20))');
END;
/

/*
```

4. Regenerate replication support for the affected object.

```
*/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'regions',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

/*
```

5. Create a trigger that records the global name of the site when a row is either inserted or updated. This recorded value is used in the resolution of conflicts based on the site priority 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 => 'hr_repg',
      type => 'TRIGGER',
      oname => 'insert_site',
      sname => 'hr',
      ddl_text => 'CREATE TRIGGER hr.insert_site
                     BEFORE
                       INSERT OR UPDATE ON hr.regions FOR EACH ROW
                     BEGIN
                       IF DBMS_REPUTIL.FROM_REMOTE = FALSE THEN
                         SELECT global_name INTO :NEW.SITE FROM GLOBAL_NAME;
                       END IF;
                     END;');
END;
/

/*
```

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

```
*/

BEGIN
   DBMS_REPCAT.MAKE_COLUMN_GROUP (
      sname => 'hr',
      oname => 'regions',
      column_group => 'regions_sitepriority_cg',
      list_of_column_names => 'region_id,region_name,site');
END;
/

/*
```

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

```
*/
```

```
BEGIN
   DBMS_REPCAT.DEFINE_SITE_PRIORITY (
      gname => 'hr_repg',
      name => 'regions_sitepriority_pg');
END;
/

/*
```

8. Define the priority value for each of the sites in your replication environment using the `DBMS_REPCAT.ADD_SITE_PRIORITY_SITE` procedure. Execute this procedure 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 => 'hr_repg',
      name => 'regions_sitepriority_pg',
      site => 'orc1.world',
      priority => 100);
END;
/

BEGIN
   DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
      gname => 'hr_repg',
      name => 'regions_sitepriority_pg',
      site => 'orc2.world',
      priority => 50);
END;
/

BEGIN
   DBMS_REPCAT.ADD_SITE_PRIORITY_SITE (
      gname => 'hr_repg',
      name => 'regions_sitepriority_pg',
      site => 'orc3.world',
      priority => 25);
END;
/

/*
```

9. After assigning your site priority values, add the SITE PRIORITY resolution method to your replicated table. The following example shows that it is the third conflict resolution method for the specified column group (sequence_no parameter).

```
*/

BEGIN
   DBMS_REPCAT.ADD_UPDATE_RESOLUTION (
      sname => 'hr',
      oname => 'regions',
      column_group => 'regions_sitepriority_cg',
      sequence_no => 1,
      method => 'SITE PRIORITY',
      parameter_column_name => 'site',
      priority_group => 'regions_sitepriority_pg');
END;
/

/*
```

10. Regenerate replication support for the table that received the conflict resolution method.

```
*/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'regions',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

/*
```

11. Resume replication activity by using the RESUME_MASTER_ACTIVITY procedure.

```
*/
```

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

/************************ END OF SCRIPT *********************************/
```

Creating Conflict Resolution Methods for Uniqueness Conflicts

In a replication environment, you may have situations where you encounter a conflict on a unique constraint, often resulting from an insert. If your business rules allow you to delete the duplicate row, then you can define a resolution method 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 the uniqueness conflict resolution method 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.

A uniqueness conflict resolution method detects and resolves conflicts encountered on columns with a `UNIQUE` constraint. The example in this section uses the `employees` table in the `hr` sample schema, which has the unique constraint `emp_email_uk` on the `email` column.

Note: To add unique conflict resolution method for a column, the name of the unique index on the column must match the name of the unique or primary key constraint.

Complete the following steps to create a uniqueness conflict resolution method.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 6-31 into a text editor and then edit the text to create a script for your environment.

```
/************************ BEGINNING OF SCRIPT ****************************
```

1. Connect as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

2. Before you define a uniqueness conflict resolution method, 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 => 'hr_repg');
END;
/

/*
```

3. Create a table that stores the messages received from your notification facility. In this example, the table name is `conf_report`.

```
*/
```

```
BEGIN
   DBMS_REPCAT.EXECUTE_DDL (
      gname => 'hr_repg',
      ddl_text => 'CREATE TABLE hr.conf_report (
                  line NUMBER(2),
                  txt VARCHAR2(80),
                  timestamp DATE,
                  table_name VARCHAR2(30),
                  table_owner VARCHAR2(30),
                  conflict_type VARCHAR2(7))');
END;
/

/*
```

4. Connect as the owner of the table you created in Step 3.

```
*/

CONNECT hr/hr@orc1.world

/*
```

5. Create a package that sends a notification to the `conf_report` table when a conflict is detected. In this example, the package name is `notify`.

    See Also: Appendix B, "User-Defined Conflict Resolution Methods" describes the conflict resolution notification package that is created in this script

```
*/

CREATE OR REPLACE PACKAGE notify AS
   FUNCTION emp_unique_violation (email 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 hr.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(email 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 EMPLOYEES 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) := 'EMAIL: ' || email;
      conf_report(5) := NULL;
      report_conflict(conf_report,5,conf_time,'employees','hr','UNIQUE');
      discard_new_values := FALSE;
      RETURN FALSE;
   END emp_unique_violation;
END notify;
/

/*
```

6. Connect as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

7. Replicate the package you created in Step 5 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 => 'hr_repg',
      type =>  'PACKAGE',
      oname => 'notify',
      sname => 'hr');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type =>  'PACKAGE BODY',
      oname => 'notify',
      sname => 'hr');
END;
/

/*
```

8. Add the notification facility as one of your conflict resolution methods, even though it only notifies of a conflict. The following example demonstrates adding the notification facility as a `USER FUNCTION`.

```
*/
```

```
BEGIN
   DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
      sname => 'hr',
      oname => 'employees',
      constraint_name => 'emp_email_uk',
      sequence_no => 1,
      method => 'USER FUNCTION',
      comment => 'Notify DBA',
      parameter_column_name => 'email',
      function_name => 'hr.notify.emp_unique_violation');
END;
/


/*
```

9. Add the actual conflict resolution method to your table. The following example demonstrates adding the APPEND SITE NAME uniqueness conflict resolution method to your replicated table.

```
*/

BEGIN
   DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
      sname => 'hr',
      oname => 'employees',
      constraint_name => 'emp_email_uk',
      sequence_no => 2,
      method => 'APPEND SITE NAME',
      parameter_column_name => 'email');
END;
/


/*
```

10. Regenerate replication support for the table that received the conflict resolution methods.

```
*/
```

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'employees',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

/*
```

11. Resume replication activity by using the `RESUME_MASTER_ACTIVITY` procedure.

```
*/

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/************************* END OF SCRIPT **********************************/
```

Creating 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 developing the 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 hr.locations WHERE remove_date IS NULL;
```

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

Complete the following steps to create a conflict avoidance method for delete conflicts.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 6-36 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect as the replication administrator at the master definition site.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

2. 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 => 'hr_repg');
END;
/

/*
```

3. Add a column to the 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, the new column can 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 => 'hr',
      oname => 'locations',
      type => 'TABLE',
      ddl_text => 'ALTER TABLE hr.locations ADD (remove_date DATE)');
END;
/

/*
```

4. Regenerate replication support for the altered table.

```
*/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'locations',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

/*
```

5. Create a package that 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 => 'hr_repg',
      type => 'PACKAGE',
      oname => 'purge',
      sname => 'hr',
      ddl_text => 'CREATE OR REPLACE PACKAGE hr.purge AS
                      PROCEDURE remove_locations(purge_date DATE);
                   END;');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'PACKAGE BODY',
      oname => 'purge',
      sname => 'hr',
      ddl_text => 'CREATE OR REPLACE PACKAGE BODY hr.purge AS
                      PROCEDURE remove_locations(purge_date IN DATE) IS
                      BEGIN
                         DBMS_REPUTIL.REPLICATION_OFF;
                         LOCK TABLE hr.locations IN EXCLUSIVE MODE;
                         DELETE hr.locations 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;
/

/*
```

6. Generate replication support for each package and package body. After generating 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_locations`.

```
*/
```

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'purge',
      type => 'PACKAGE',
      min_communication => TRUE);
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'purge',
      type => 'PACKAGE BODY',
      min_communication => TRUE);
END;
/

/*
```

7. In a separate terminal window, manually push any administrative requests at all other master sites. You may need to execute the DO_DEFERRED_REPCAT_ADMIN procedure in the DBMS_REPCAT package several times, because some administrative operations have multiple steps. The following is an example:

```
BEGIN
   DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN (
      gname => 'hr_repg',
      all_sites => FALSE);
END;
/

*/

PAUSE Press <RETURN> to continue when you have verified that there are no
pending administrative requests in the DBA_REPCATLOG data dictionary view.

/*
```

8. Resume replication activity by using the RESUME_MASTER_ACTIVITY procedure.

```
*/

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/************************* END OF SCRIPT **********************************/
```

Using Dynamic Ownership Conflict Avoidance

This section describes a more advanced method of designing your applications to avoid conflicts. This method, known as **token passing,** is similar to the workflow method described below. Although this section describes how to use this method to control the ownership of an entire row, you can use a modified form of this method to control ownership of the individual column groups within a row.

Both workflow and token passing allow dynamic ownership of data. With dynamic ownership, only one site at a time is allowed to update a row, but ownership of the row can be passed from site to site. Both workflow and token passing use the value of one or more "identifier" columns to determine who is currently allowed to update the row.

Workflow

With workflow partitioning, you can think of data ownership as being "pushed" from site to site. Only the current owner of the row is allowed to push the ownership of the row to another site, by changing the value of the "identifier" columns.

Take the simple example of separate sites for ordering, shipping, and billing. Here, the identifier columns are used to indicate the status of an order. The status determines which site can update the row. After a user at the ordering site has entered the order, the user updates the status of this row to `ship`. Users at the ordering site are no longer allowed to modify this row — ownership has been pushed to the shipping site.

After shipping the order, the user at the shipping site updates the status of this row to `bill`, thus pushing ownership to the billing site, and so on.

To successfully avoid conflicts, applications implementing dynamic data ownership must ensure that the following conditions are met:

- Only the owner of the row can update the row.
- The row is never owned by more than one site.
- Ordering conflicts can be successfully resolved at all sites.

With workflow partitioning, only the current owner of the row can push the ownership of the row to the next site by updating the "identifier" columns. No site is given ownership unless another site has given up ownership; thus ensuring there is never more than one owner.

Because the flow of work is ordered, ordering conflicts can be resolved by applying the change from the site that occurs latest in the flow of work. Any ordering conflicts can be resolved using a form of the priority conflict resolution method, where the priority value increases with each step in the work flow process. The priority conflict resolution method successfully converges for more than one master site as long as the priority value is always increasing.

Token Passing

Token passing uses a more generalized approach to meeting these criteria. To implement token passing, instead of the "identifier" columns, your replicated tables must have owner and epoch columns. The owner column stores the global database name of the site currently believed to own the row.

Once you have designed a token passing mechanism, you can use it to implement a variety of forms of dynamic partitioning of data ownership, including workflow.

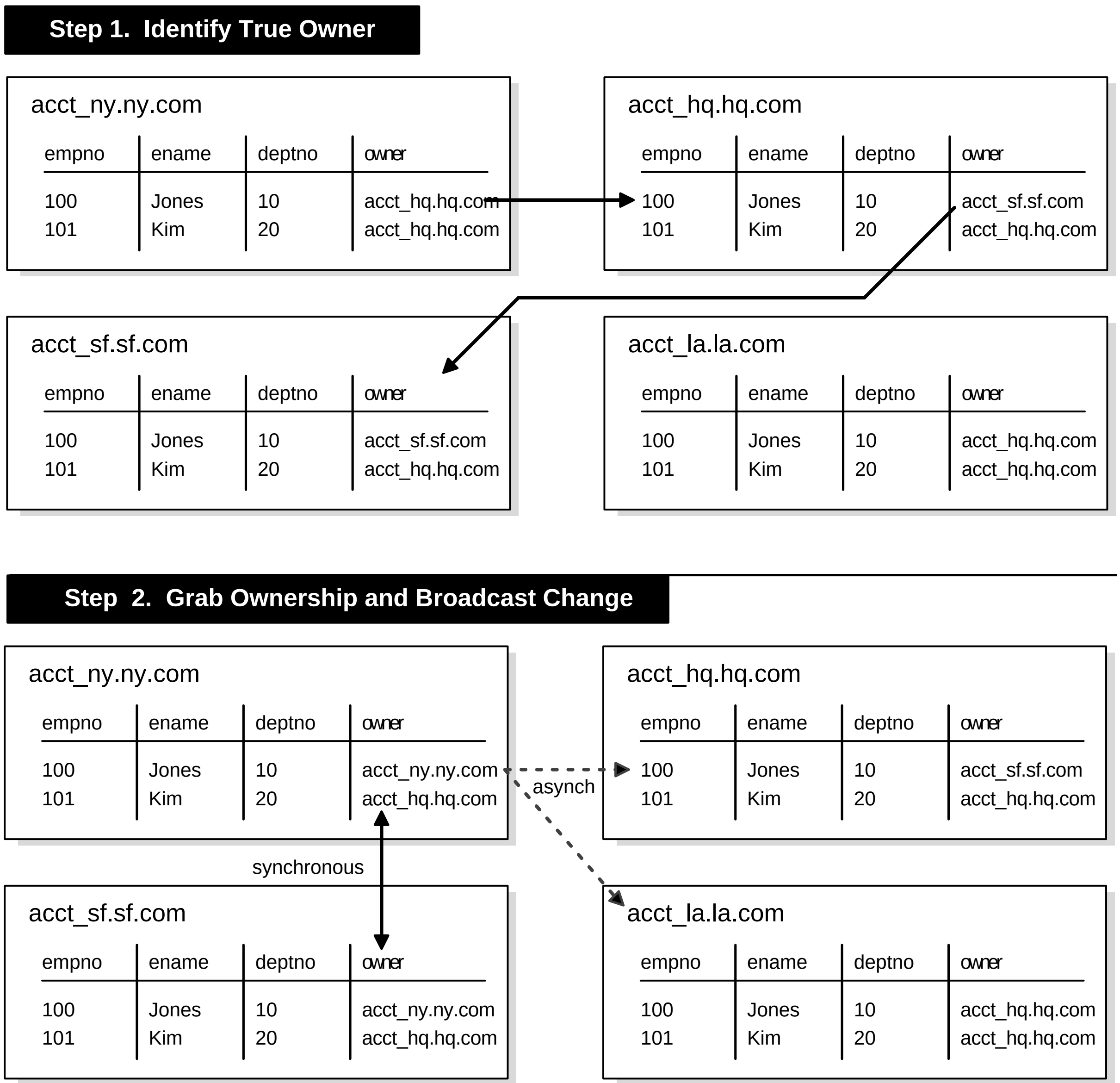
You should design your application to implement token passing for you automatically. You should not allow the owner or epoch columns to be updated outside this application.

Whenever you attempt to update a row, your application should:

1. Locate the current owner of the row.
2. Lock the row to prevent updates while ownership is changing.
3. Establish ownership of the row.
4. Perform the update. Oracle releases the lock when you commit your transaction.

For example, Figure 6–1 illustrates how ownership of employee `100` passes from the `acct_sf` database to the `acct_ny` database.

Figure 6–1 Grabbing the Token



Locating the Owner of a Row

To obtain ownership, the `acct_ny` database uses a simple recursive algorithm to locate the owner of the row. The sample code for this algorithm is shown below:

```
-- Sample code for locating the token owner.
-- This is for a table TABLE_NAME with primary key PK.
-- Initial call should initialize loc_epoch to 0 and loc_owner
-- to the local global name.
get_owner(PK IN primary_key_type, loc_epoch IN OUT NUMBER,
          loc_owner IN OUT VARCHAR2)
{
  -- use dynamic SQL (dbms_sql) to perform a select similar to
  -- the following:
  SELECT owner, epoch into rmt_owner, rmt_epoch
     FROM TABLE_NAME@loc_owner
     WHERE primary_key = PK FOR UPDATE;
  IF rmt_owner = loc_owner AND rmt_epoch >= loc_epoch THEN
   loc_owner := rmt_owner;
   loc_epoch := rmt_epoch;
   RETURN;
  ELSIF rmt_epoch >= loc_epoch THEN
   get_owner(PK, rmt_epoch, rmt_owner);
   loc_owner := rmt_owner;
   loc_epoch := rmt_epoch;
   RETURN;
  ELSE
   raise_application_error(-20000, 'No owner for row');
  END IF;
}
```

Obtaining Ownership

After locating the owner of the row, the `acct_ny` site gets ownership from the `acct_sf` site by completing the following steps:

1. Lock the row at the `sf` site to prevent any changes from occurring while ownership is being exchanged.
2. Synchronously update the owner information at both the `sf` and `ny` sites. This operation ensures that only one site considers itself to be the owner at all times. The update at the `sf` site should not be replicated using `DBMS_REPUTIL.REPLICATION_OFF`. The replicated change of ownership at the `ny` site in Step 4 will ultimately be propagated to all other sites in the replication environment, including the `sf` site, where it will have no effect.

3. Update the row information at the new owner site, `ny`, with the information from the current owner site, `sf`. This data is guaranteed to be the most recent. This time, the change at the `ny` site should not be replicated. Any queued changes to this data at the `sf` site are propagated to all other sites in the usual manner. When the `sf` change is propagated to `ny`, it is ignored because of the values of the epoch numbers, as described in the next bullet point.

4. Update the epoch number at the new owner site to be one greater than the value at the previous site. Perform this update at the new owner only, and then asynchronously propagate this update to the other master sites. Incrementing the epoch number at the new owner site prevents ordering conflicts.

   When the `sf` changes (that were in the deferred queue in Step 2 above) are ultimately propagated to the `ny` site, the `ny` site ignores them because they have a lower epoch number than the epoch number at the `ny` site for the same data.

   As another example, suppose the `hq` site received the `sf` changes after receiving the `ny` changes, the `hq` site would ignore the `sf` changes because the changes applied from the `ny` site would have the greater epoch number.

Applying the Change

You should design your application to implement this method of token passing for you automatically whenever you perform an update. You should not allow the owner or epoch columns to be updated outside this application. The lock that you grab when you change ownership is released when you apply your actual update. The changed information, along with the updated owner and epoch information, are asynchronously propagated to the other sites in the usual manner.

Auditing 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 ALL_REPRESOLUTION_STATISTICS data dictionary view. This view is updated only if you have enabled conflict resolution statistics gathering for the table involved in the conflict.

> **See Also:** The ALL_REPRESOLUTION_STATISTICS view on page 25-37 for more information

Collecting 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 employees table in the hr schema:

```
BEGIN
    DBMS_REPCAT.REGISTER_STATISTICS (
      sname => 'hr',
      oname => 'employees');
END;
/
```

> **See Also:** "REGISTER_STATISTICS Procedure" on page 20-94 for more information

Viewing Conflict Resolution Statistics

After calling REGISTER_STATISTICS for a table, each conflict that is successfully resolved for that table is logged in the ALL_REPRESOLUTION_STATISTICS data dictionary view. Information about unresolved conflicts is always logged in the DEFERROR view, whether the object is registered or not.

> **See Also:** The ALL_REPRESOLUTION_STATISTICS view on page 25-37 and the DEFERROR view on page 27-4 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 employees table in the hr schema:

```
BEGIN
    DBMS_REPCAT.CANCEL_STATISTICS (
      sname => 'hr',
      oname => 'employees');
END;
/
```

See Also: The "CANCEL_STATISTICS Procedure" on page 20-40 for more information

Clearing Statistics Information

If you registered a table to log information about the successful resolution of update, delete, and uniqueness conflicts, then you can remove this information from the DBA_REPRESOLUTION_STATISTICS data dictionary 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 employees table between January 1 and March 31:

```
BEGIN
    DBMS_REPCAT.PURGE_STATISTICS (
      sname  => 'hr',
      oname  => 'employees',
      start_date => '01-JAN-2001',
      end_date => '31-MAR-2001');
END;
/
```

See Also: The "PURGE_STATISTICS Procedure" on page 20-89 for more information

Part II

Managing and Monitoring Your Replication Environment

Part II contains instructions on using the replication management API to manage your replication environment, as well as instructions on using the data dictionary to monitor your replication environment.

Part II contains the following chapters:

7

Managing a Master Replication Environment

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 chapter discusses managing the master sites of your replication environment. Specifically, this section describes altering and reconfiguring your master sites.

This chapter contains these topics:

Changing the Master Definition Site

Many replication administrative tasks can be performed only from the master definition site. Use the RELOCATE_MASTERDEF procedure in the DBMS_REPCAT package 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: The Old Master Definition Site Is Not Available" on page 7-3).

See Also: "RELOCATE_MASTERDEF Procedure" on page 20-95

Option 1: All Master Sites Are Available

Perform the actions in this section to change the master definition site if all master sites are available. Meet the following requirements to complete these actions:

Executed As: Replication Administrator

Executed At: Any Master Site

Replication Status: Running Normally (Not Quiesced)

Complete the following steps:

1. Connect to a master site as the replication administrator.

```
CONNECT repadmin/repadmin@orc1.world
```

2. Relocate the master definition site.

```
BEGIN
   DBMS_REPCAT.RELOCATE_MASTERDEF (
      gname => 'hr_repg',
      old_masterdef => 'orc1.world',
      new_masterdef => 'orc2.world',
      notify_masters => TRUE,
      include_old_masterdef => TRUE);
END;
/
```

Option 2: The Old Master Definition Site Is Not Available

Perform the actions in this section to change the master definition site if the old master definition site is *not* available. Meet the following requirements to complete these actions:

Executed As: Replication Administrator

Executed At: Any Master Site

Replication Status: Normal

Complete the following steps:

1. Connect to a master site as the replication administrator.

```
CONNECT repadmin/repadmin@orc3.world
```

2. Relocate the master definition site.

```
BEGIN
   DBMS_REPCAT.RELOCATE_MASTERDEF (
      gname => 'hr_repg',
      old_masterdef => 'orc1.world',
      new_masterdef => 'orc2.world',
      notify_masters => TRUE,
      include_old_masterdef => FALSE);
END;
/
```

Adding New Master Sites

As your replication environment expands, you may need to add new master sites to a master group. You can either add new master sites to a master group that is running normally or to a master group that is quiesced. If the master group is not quiesced, then users can perform data manipulation language (DML) operations on the data while the new master sites are being added. However, more administrative actions are required when adding new master sites if the master group is not quiesced.

Note: When adding a master site to a master group that contains tables with circular dependencies or a table that contains a self-referential constraint, you must precreate the table definitions and manually load the data at the new master site. The following is an example of a circular dependency: Table A has a foreign key constraint on table B, and table B has a foreign key constraint on table A.

Follow the instructions in the appropriate section to add new master sites to a master group:

- Adding New Master Sites Without Quiescing the Master Group
- Adding New Master Sites to a Quiesced Master Group

Adding New Master Sites Without Quiescing the Master Group

This section contains procedures for adding new master sites to an existing master group that is not quiesced. These new sites may or may not already be replication sites (master sites and/or materialized view sites) in other replication groups.

You can use one of the following methods when you are adding a new master site without quiescing the master group:

- Use full database export/import or change-based recovery to add a new master site that does not currently have any replication groups. See "Using Full Database Export/Import or Change-Based Recovery" on page 7-9 for instructions.
- Use object-level export/import to add a new master site that already has other replication groups or to add a new master site that does not currently have any replication groups. See "Using Object-Level Export/Import" on page 7-19 for instructions.

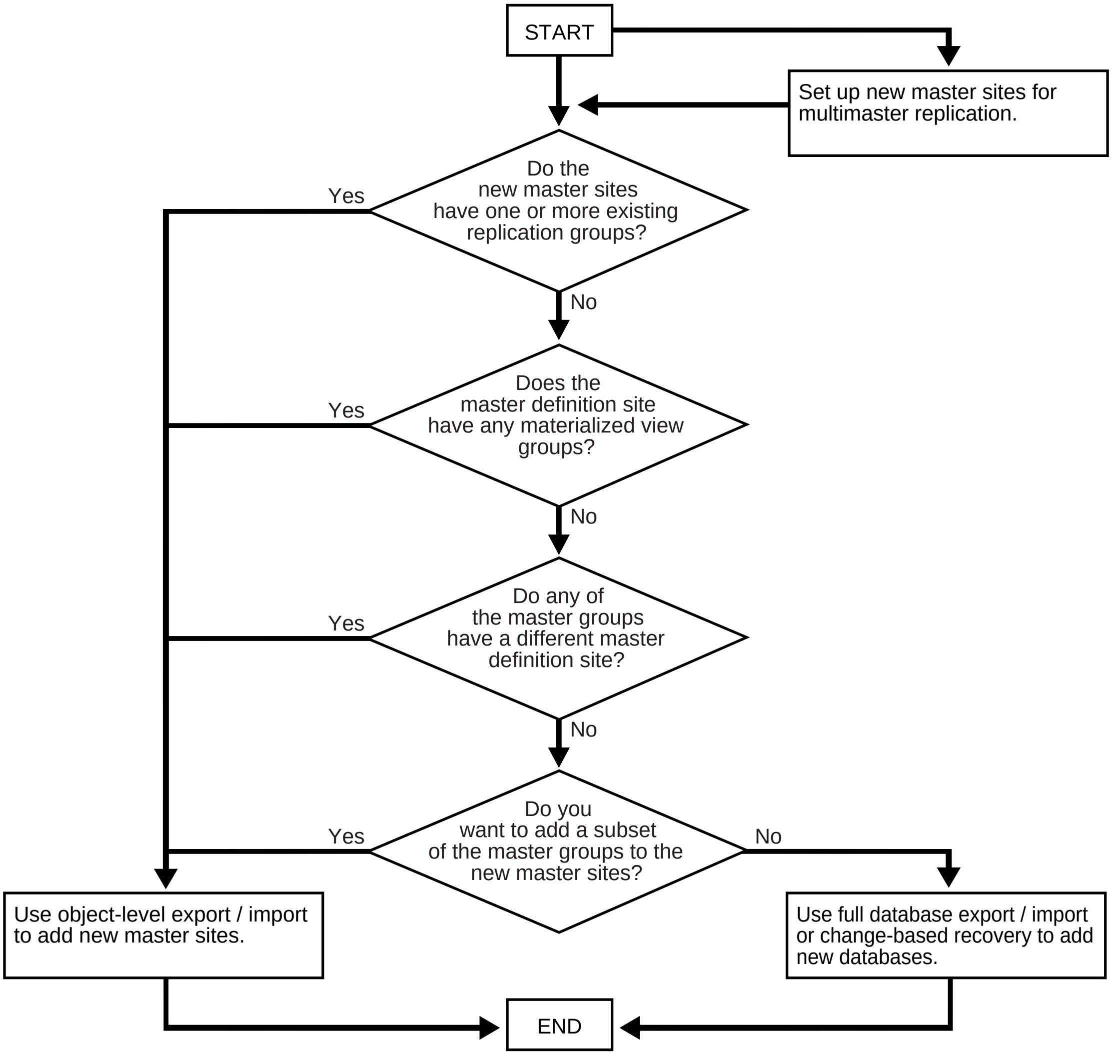
Use full database export/import and change-based recovery to add all of the replication groups at the master definition site to the new master sites. When you use this method, the following conditions apply:

- The new master sites cannot have any existing replication groups.
- The master definition site cannot have any materialized view groups.
- The master definition site must be the same for all of the master groups. If one or more of these master groups have a different master definition site, then do not use full database export/import or change-based recovery. Use object-level export/import instead.
- The new master site must include all of the replication groups in the master definition site when the extension process is complete. That is, you cannot add a subset of the master groups at the master definition site to the new master site. All of the groups must be added.

Note: To use change-based recovery, the existing master site and the new master site must be running under the same operating system, although the release of the operating system can differ. This condition does not apply to full database export/import.

If your environment does not meet all of these conditions, then you must use object-level export/import to add the new master sites. Figure 7–1 summarizes these conditions.

Figure 7–1 Determining Which Method to Use When Adding Master Sites



Use object-level export/import to add a master group to master sites that already have other replication groups or to add a master group to master sites that do not currently have any replication groups. This method can add one or more master groups to new master sites at a time, and you can choose a subset of the master groups at the master definition site to add to the new master sites during the operation.

If you use object-level export/import and there are integrity constraints that span more than one master group, then you must temporarily disable these integrity constraints on the table being added to a new master site, if the other tables to which these constraints refer already exist at the new master site. Initially, there are two rows in the `DEFSCHEDULE` data dictionary view that refer to the new master sites. When propagation is caught up, there is one row in this view, and when propagation from all the master sites to the new master site is caught up, you can re-enable the integrity constraints you disabled.

Again, the two methods for adding new master sites without quiescing the master groups are the following:

- Full database export/import or change-based recovery
- Object-level export/import

When you use either method, propagation of deferred transactions to the new master site is partially or completely disabled while the new master sites are being added. Therefore, make sure each existing master site has enough free space to store the largest unpropagated deferred transaction queue that you may encounter.

In addition, the following restrictions apply to both methods:

- All affected master groups must be using asynchronous replication. Synchronous replication is not allowed.
- All scheduled links must use parallel propagation with parallelism set to 1 or higher.
- Either the database links of all affected master groups must have no connection qualifier or they must all have the same connection qualifier.
- After you begin the process of adding new master sites to one or more master groups, you must wait until these new master sites are added before you begin to add another set of master sites to any of the affected master groups. If there is information about an affected master group in the `DBA_NEW_REPSITES` data dictionary view at the master definition site, then the process is started and is not yet complete for that master group.

- After you begin the process of adding new master sites to one or more master groups, you cannot relocate the master definition site for these master groups until the new master sites are added. If there is information about an affected master group in the `DBA_NEW_REPSITES` data dictionary view, then the process is started and is not yet complete for that master group.
- Only one add master site request at a time is allowed at a master site. For example, if `hq1.world` is the master definition site for `mgroup1` and `hq2.world` is the master definition site for `mgroup2`, then you cannot add `hq1.world` to `mgroup2` and `hq2.world` to `mgroup1` at the same time.
- All master sites must be at 9.0.0 or higher compatibility level. You control the compatibility level with the `COMPATIBLE` initialization parameter. If any master sites are lower than 9.0.0 compatibility level, then the master group must be quiesced to extend it with new master sites. In this case, follow the instructions in "Adding New Master Sites to a Quiesced Master Group" on page 7-28.
- If you are using object-level or full database export/import, then make sure there is enough space in your rollback segments or undo tablespace for the export.

Also, before adding new master sites with either method, make sure you properly set up your new master sites for multimaster replication.

Note: If progress appears to stop during one of the procedures described in the following sections, then check your trace files and the alert log for messages.

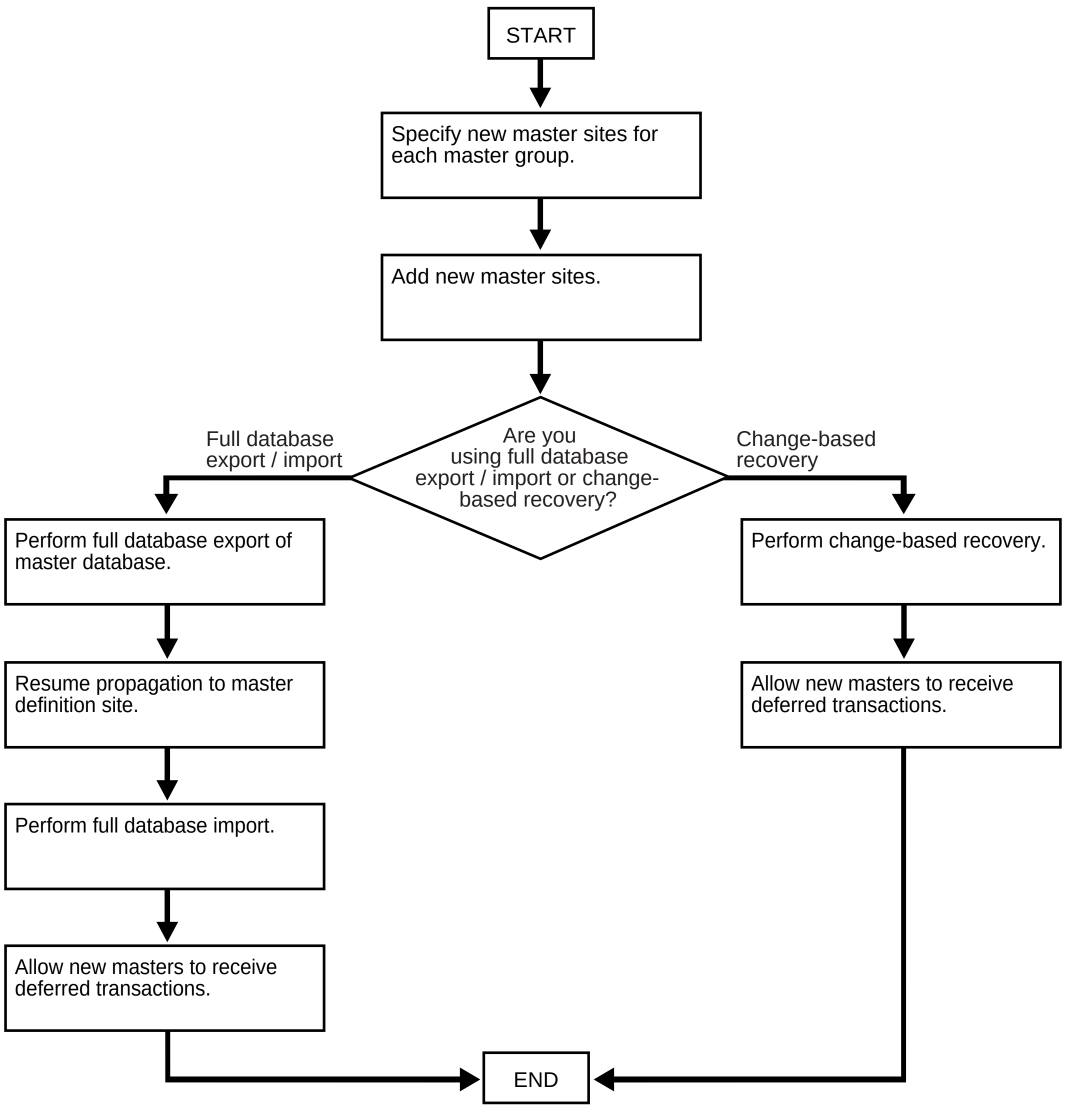
See Also:

- "Setting Up Master Sites" on page 2-4 for information about setting up your new master sites for multimaster replication
- *Oracle9i Database Administrator's Guide* for more information about trace files and the alert log
- *Oracle9i Database Administrator's Guide* for information about managing undo space

Using Full Database Export/Import or Change-Based Recovery

Figure 7–2 shows the major steps for using full database export/import or change-based recovery to add new master sites to a master group without quiescing. The following example script adds the new master sites `orc4.world` and `orc5.world` to the `hr_repg` and `oe_repg` master groups. In this example, `orc4.world` is added using full database export/import and `orc5.world` is added using change-based recovery.

Figure 7–2 Using Full Database Export/Import or Change-Based Recovery



Meet the following requirements to complete these actions:

Executed As: Replication Administrator

Executed At:

- Step 1 at Each New Master Site
- Steps 2 - 5b at Master Definition Site
- Step 5c requires a file transfer between sites.
- Steps 5d - 8 at Each New Master Site

Replication Status: Running Normally (Not Quiesced)

Complete the following steps to use full database export/import or change-based recovery to add sites to a master group.

> **Note:** If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 7-19 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. If you are using full database export/import, then create the databases that you want to add to the master group.

   This step is not required if you are using change-based recovery.

   > **See Also:** *Oracle9i Database Administrator's Guide* for information about creating a database

```
*/

PAUSE Press <RETURN> when the databases for the new master sites are
created.

/*
```

2. Connect as the replication administrator to the master definition site.

```
*/

CONNECT repadmin/repadmin@orc1.world
```

```
/*
```

3. Specify new master sites for each master group.

   Before you begin, create the required scheduled links between existing master sites and each new master site if they do not already exist.

   See Also:

   - *Oracle9i Replication* for information about scheduled links
   - "Creating Scheduled Links Between the Master Sites" on page 2-15 for examples

```
*/

BEGIN
   DBMS_REPCAT.SPECIFY_NEW_MASTERS (
      gname => 'hr_repg',
      master_list => 'orc4.world,orc5.world');
END;
/

BEGIN
   DBMS_REPCAT.SPECIFY_NEW_MASTERS (
      gname => 'oe_repg',
      master_list => 'orc4.world,orc5.world');
END;
/

/*
```

   You can begin to track the extension process by querying the following data dictionary views in another SQL*Plus session:

   - DBA_REPSITES_NEW
   - DBA_REPEXTENSIONS

4. Add the new master sites.

   Before running the following procedure, ensure that there are an adequate number of background jobs running at each new master site. If you are using full database export/import, then make sure there is enough space in your rollback segments or undo tablespace for the export before you run this procedure.

See Also:

- *Oracle9i Replication* for information about setting the JOB_QUEUE_PROCESSES initialization parameter properly for a replication environment
- *Oracle9i Database Administrator's Guide* for information about managing undo space

```
*/

VARIABLE masterdef_flashback_scn NUMBER;
VARIABLE extension_id VARCHAR2(32);
BEGIN
   DBMS_REPCAT.ADD_NEW_MASTERS (
      export_required => true,
      available_master_list => NULL,
      masterdef_flashback_scn => :masterdef_flashback_scn,
      extension_id => :extension_id,
      break_trans_to_masterdef => false,
      break_trans_to_new_masters => false,
      percentage_for_catchup_mdef => 80,
      cycle_seconds_mdef => 60,
      percentage_for_catchup_new => 80,
      cycle_seconds_new => 60);
END;
/

/*
```

The values for masterdef_flashback_scn and extension_id are saved into variables to be used later in the process. To see these values, you can query the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.

If you need to undo the changes made to a particular master site by the SPECIFY_NEW_MASTERS and ADD_NEW_MASTERS procedures, then use the UNDO_ADD_NEW_MASTERS_REQUEST procedure.

For the export_required parameter, true is specified because orc4.world is being added using full database export/import. Although orc5.world is using change-based recovery, the true setting is correct because at least one new master site is added using export/import.

After successfully executing this procedure, monitor its progress by querying the DBA_REPCATLOG data dictionary view in another SQL*Plus session. Do not proceed to Step 5 until there is no remaining information in this view about adding the new master sites. Assuming no extraneous information exists in DBA_REPCATLOG from other operations, you can enter the following statement:

```
SELECT COUNT(*) FROM DBA_REPCATLOG;
```

All of the processing is complete when this statement returns zero (0).

```
*/

PAUSE Press <RETURN> to continue when DBA_REPCATLOG is empty.

/*
```

5. Perform the following substeps for the master sites being added using full database export/import. For master sites being added using change-based recovery, these substeps are not required and you can proceed to Step 6 on page 7-16.

   a. Perform full database export of master definition database. Use the system change number (SCN) returned by the masterdef_flashback_scn parameter in Step 4 for the FLASHBACK_SCN export parameter.

      You can query the DBA_REPEXTENSIONS data dictionary view for the FLASHBACK_SCN value:

      ```
      SELECT FLASHBACK_SCN FROM DBA_REPEXTENSIONS;
      ```

      In this example, orc4.world is using full database export/import. Therefore, perform the full database export of the master definition database so that it can be imported into orc4.world during a later step. However, the orc5.world database is using change-based recovery. Therefore, the export would not be required if you were adding only orc5.world.

      The following is an example of an export statement:

      ```
      exp system/manager FILE=fulldb_orc1.dmp FULL=y DIRECT=n
      GRANTS=y ROWS=y COMPRESS=y STATISTICS=compute LOG=exp_orc1.log
      FLASHBACK_SCN=124723
      ```

Consider the following when you run the Export utility:

– Only users with the `DBA` role or the `EXP_FULL_DATABASE` role can export in full database mode.

– Make sure the `UNDO_RETENTION` initialization parameter is set correctly before performing the export.

– Do not use the `CONSISTENT` export parameter.

See Also:

- *Oracle9i Database Utilities* for information about performing a full database export
- *Oracle9i Database Administrator's Guide* for information about managing undo space and setting this parameter

```
*/

PAUSE Press <RETURN> to continue when the export is complete.

/*
```

b. Resume propagation to the master definition site.

Running the following procedure indicates that export is effectively finished and propagation can be enabled for both extended and unaffected master groups at the master sites.

```
*/

BEGIN
   DBMS_REPCAT.RESUME_PROPAGATION_TO_MDEF (
      extension_id => :extension_id);
END;
/

/*
```

You can find the `extension_id` by querying the `DBA_REPSITES_NEW` data dictionary view.

c. Transfer the export dump file to the new master sites.

   Using FTP or some other method, transfer the export dump file to the other new master sites that are being added with full database export/import. You will need this export dump file at each new site to perform the import described in the next step.

```
*/

PAUSE Press <RETURN> to continue after transferring the dump file.

/*
```

d. Set the `JOB_QUEUE_PROCESSES` initialization parameter to zero for each new master site.

```
*/

PAUSE Press <RETURN> to continue after JOB_QUEUE_PROCESSES is set to
zero at each new master site.

/*
```

6. Perform import or change-based recovery at each new master site.

   - If you are using full database export/import, then complete the full database import of the database you exported in Step 5a at each new master site that is being added with full database export/import.

     The following is an example of an import statement:

```
imp system/manager FILE=fulldb_orc1.dmp FULL=y BUFFER=30720
IGNORE=y GRANTS=y ROWS=y DESTROY=y COMMIT=y LOG=import.log
```

     Only users with the DBA role or the `IMP_FULL_DATABASE` role can import in full database mode.

     See Also: *Oracle9i Database Utilities* for information about performing a full database import

```
*/

PAUSE Press <RETURN> to continue when the import is complete.

/*
```

- If you are using change-based recovery, then perform change-based recovery using the system change number (SCN) returned by the `masterdef_flashback_scn` parameter in Step 4. You can query the `DBA_REPEXTENSIONS` data dictionary view for the `masterdef_flashback_scn` value.

  You can perform a change-based recovery in one of the following ways:

  * Using the SQL*Plus `RECOVER` command. See the *Oracle9i User-Managed Backup and Recovery Guide* for instructions.
  * Using the Recovery Manager (RMAN) `DUPLICATE` command. See the *Oracle9i Recovery Manager User's Guide* for instructions.

  Connect to the site where you will perform the change-based recovery:

```
*/

CONNECT repadmin/repadmin@orc5.world

PAUSE Press <RETURN> to continue when the change-based recovery is
complete. You can use a separate terminal window to perform the
change-based recovery.

/*
```

7. Configure the new sites for multimaster replication by completing the following steps:

a. Ensure that the database structures, such as the datafiles, exist for the replicated schemas at each new master site. In this example, the replicated schemas are `hr` and `oe`.

b. Set the global name for each new master site. The global name for each new master site must match the global names specified in the `SPECIFY_NEW_MASTERS` procedure that you ran in Step 3. You can query the `DBLINK` column in the `DBA_REPSITES_NEW` data dictionary view to see the global name for each new master site.

You can set the global name using the `ALTER DATABASE` statement, as in the following example:

```
ALTER DATABASE RENAME GLOBAL_NAME TO ORC4.WORLD;
```

c. Create the appropriate scheduled links between the new master sites and the existing master sites, including the master definition site.

See Also: "Creating Scheduled Links Between the Master Sites" on page 2-15 for information

```
*/

PAUSE Press <RETURN> when you have completed the these steps.

/*
```

8. Allow new masters to receive deferred transactions.

The following procedure enables the propagation of deferred transactions from other prepared new master sites and existing master sites to the invocation master site. This procedure also enables the propagation of deferred transactions from the invocation master site to the other new master sites and existing master sites.

Caution: Do not invoke this procedure until instantiation (export/import or change-based recovery) of the new master site is complete.

Do not allow any data manipulation language (DML) statements directly on the objects in the extended master group in the new master site until execution of this procedure returns successfully, because these DML statements may not be replicated.

```
*/

CONNECT repadmin/repadmin@orc4.world

BEGIN
   DBMS_REPCAT.PREPARE_INSTANTIATED_MASTER (
      extension_id  => :extension_id);
END;
/
```

```
CONNECT repadmin/repadmin@orc5.world

BEGIN
   DBMS_REPCAT.PREPARE_INSTANTIATED_MASTER (
      extension_id  => :extension_id);
END;
/

/*
```

> **Note:** You can find the `extension_id` by querying the `DBA_REPSITES_NEW` data dictionary view.

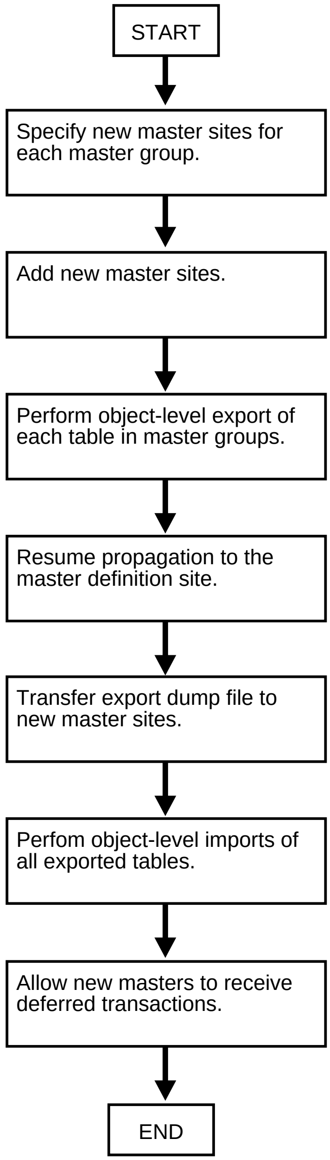
```
/************************* END OF SCRIPT **********************************/
```

Using Object-Level Export/Import

Figure 7–3 shows the major steps for using object-level export/import to add new master sites to a master group without quiescing. The following example procedure adds the new master sites `orc4.world` and `orc5.world` to the `hr_repg` and `oe_repg` master groups. An object-level export/import involves exporting and importing the tables in a master group. When you export and import the tables, other dependent database objects, such as indexes, are exported and imported as well.

If you have an integrity constraint that spans two master groups, then you have a child table in one master group (the child master group) and a parent table in a different master group (the parent master group). In this case, Oracle Corporation recommends that you add new master sites to both master groups at the same time. However, if you cannot do this, then you must quiesce the child master group before adding new master sites to it. Here, the child table includes a foreign key, which makes it dependent on the values in the parent table. If you do not quiesce the child master group, then conflicts may result when you add master sites to it. You can still add master sites to the parent master group without quiescing it.

Figure 7–3 Using Object-Level Export/Import



Meet the following requirements to complete these actions:

Executed As: Replication Administrator

Executed At:

- Steps 1 - 6 at Master Definition Site
- Step 7 requires a file transfer between sites.
- Steps 8 - 9 at Each New Master Site

Replication Status: Running Normally (Not Quiesced)

Also, make sure the replicated schemas exist at the new master sites before you begin. In this example, the replicated schemas are `hr` and `oe`.

Complete the following steps to use object-level export/import to add sites to a master group.

> **Note:** If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 7-27 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect to the master definition site as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

2. Create the required scheduled links between existing master sites and each new master site if they do not already exist.

```
*/

PAUSE Press <RETURN> to continue the required scheduled links have been
created.

/*
```

See Also:

- *Oracle9i Replication* for information about scheduled links
- "Creating Scheduled Links Between the Master Sites" on page 2-15 for examples

3. Specify new master sites for each master group.

```
*/

BEGIN
   DBMS_REPCAT.SPECIFY_NEW_MASTERS (
      gname => 'hr_repg',
      master_list => 'orc4.world,orc5.world');
END;
/

BEGIN
   DBMS_REPCAT.SPECIFY_NEW_MASTERS (
      gname => 'oe_repg',
      master_list => 'orc4.world,orc5.world');
END;
/

/*
```

You can begin to track the extension process by querying the following data dictionary views in another SQL*Plus session:

- `DBA_REPSITES_NEW`
- `DBA_REPEXTENSIONS`

4. Add the new master sites.

Before running the following procedure, ensure that there are an adequate number of background jobs running at each new master site. Also, make sure there is enough space in your rollback segments or undo tablespace for the export before you run this procedure.

See Also:

- *Oracle9i Replication* for information about setting the JOB_QUEUE_PROCESSES initialization parameter properly for a replication environment
- *Oracle9i Database Administrator's Guide* for information about managing undo space

```
*/

VARIABLE masterdef_flashback_scn NUMBER;
VARIABLE extension_id VARCHAR2(32);
BEGIN
   DBMS_REPCAT.ADD_NEW_MASTERS (
      export_required => true,
      available_master_list => 'orc4.world,orc5.world',
      masterdef_flashback_scn => :masterdef_flashback_scn,
      extension_id => :extension_id,
      break_trans_to_masterdef => false,
      break_trans_to_new_masters => false,
      percentage_for_catchup_mdef => 80,
      cycle_seconds_mdef => 60,
      percentage_for_catchup_new => 80,
      cycle_seconds_new => 60);
END;
/

/*
```

The sites specified for the available_master_list parameter must be same as the sites specified in the SPECIFY_NEW_MASTERS procedure in Step 3.

The values for masterdef_flashback_scn and extension_id are saved into variables to be used later in the process. To see these values, you can also query the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.

If you need to undo the changes made to a particular master site by the SPECIFY_NEW_MASTERS and ADD_NEW_MASTERS procedures, then use the UNDO_ADD_NEW_MASTERS_REQUEST procedure.

After successfully executing this procedure, monitor its progress by querying the `DBA_REPCATLOG` data dictionary view in another SQL*Plus session. Do not proceed to Step 5 until there is no remaining information in this view about adding the new master sites. Assuming there is no extraneous information in `DBA_REPCATLOG` from other operations, you can enter the following statement:

```
SELECT COUNT(*) FROM DBA_REPCATLOG;
```

All of the processing is complete when this statement returns zero (0).

```
*/

PAUSE Press <RETURN> to continue when DBA_REPCATLOG is empty.

/*
```

5. Perform object-level export of tables at master definition database.

   At the master definition database, perform an object-level export for each master table in the master groups that will be created at the new master sites. An object-level export includes exports performed in table mode, user mode, or tablespace mode.

   Use the system change number (SCN) returned by the `masterdef_flashback_scn` parameter in Step 4 for the `FLASHBACK_SCN` export parameter. You can query the `DBA_REPEXTENSIONS` data dictionary view for the `FLASHBACK_SCN` value:

   ```
   SELECT FLASHBACK_SCN FROM DBA_REPEXTENSIONS;
   ```

   The following is an example of an object-level export that exports the entire `hr` schema in user mode:

   ```
   exp system/manager FILE=hr_schema.dmp OWNER=hr DIRECT=n GRANTS=y
   ROWS=y COMPRESS=y INDEXES=y CONSTRAINTS=y STATISTICS=compute
   FLASHBACK_SCN=3456871
   ```

   The following is an example of an object-level export that exports the tables in the `oe` schema in table mode:

   ```
   exp system/manager FILE=oe_tables.dmp TABLES=(oe.customers, inventories,
   orders, order_items, product_descriptions, product_information, warehouses)
   DIRECT=n GRANTS=y ROWS=y COMPRESS=y INDEXES=y CONSTRAINTS=y
   STATISTICS=compute FLASHBACK_SCN=3456871
   ```

   When you export tables, their indexes are exported automatically.

Consider the following when you run the Export utility:

- Make sure the `UNDO_RETENTION` initialization parameter is set correctly before performing the export.
- Do not use the `CONSISTENT` export parameter.

See Also:

- *Oracle9i Database Utilities* for information about performing an object-level database export
- *Oracle9i Database Administrator's Guide* for information about managing undo space and setting the `UNDO_RETENTION` initialization parameter

```
*/

PAUSE Press <RETURN> to continue when the export is complete.

/*
```

6. Resume propagation to the master definition site.

   Running the following procedure indicates that export is effectively finished and propagation can be enabled for both extended and unaffected master groups at the master sites.

```
*/

BEGIN
   DBMS_REPCAT.RESUME_PROPAGATION_TO_MDEF (
      extension_id => :extension_id);
END;
/

/*
```

   You can find the `extension_id` by querying the `DBA_REPSITES_NEW` data dictionary view.

7. Transfer the export dump files to the new master sites.

   Using FTP or some other method, transfer the export dump files to the other new master sites that are being added with object-level export/import. You will need these export dump files at each new site to perform the import described in the next step.

```
*/

PAUSE Press <RETURN> to continue when the export dump files have been
transfered to the new master sites that are being added with object-level
export/import.

/*
```

8. Perform object-level imports at each new master site of each object you exported in Step 5.

The following is an example of an object-level import that imports the entire `hr` schema:

```
imp system/manager FILE=hr_schema.dmp FROMUSER=hr BUFFER=30720 IGNORE=y
GRANTS=y ROWS=y DESTROY=y COMMIT=y
```

The following is an example of an object-level import that imports the tables in the `oe` schema:

```
imp system/manager FILE=oe_tables.dmp FROMUSER=oe BUFFER=30720
TABLES=(oe.customers, inventories, orders, order_items,
product_descriptions, product_information, warehouses) IGNORE=y GRANTS=y
ROWS=y DESTROY=y COMMIT=y
```

Other objects, such as the indexes based on the tables, are imported automatically.

> **See Also:** *Oracle9i Database Utilities* for information about performing object-level imports

Connect to the site where you will perform the object-level imports and then perform the imports at each site:

```
*/

CONNECT repadmin/repadmin@orc4.world

PAUSE Press <RETURN> to continue when the imports are complete at this site.
You can use a separate terminal window to perform the object-level imports.

CONNECT repadmin/repadmin@orc5.world

PAUSE Press <RETURN> to continue when the imports are complete at this site.
You can use a separate terminal window to perform the object-level imports.
```

```
/*
```

9. Allow new masters to receive deferred transactions.

The following procedure enables the propagation of deferred transactions from other prepared new master sites and existing master sites to the invocation master site. This procedure also enables the propagation of deferred transactions from the invocation master site to the other new master sites and existing master sites.

Caution: Do not invoke this procedure until object-level export/import for the new master site is complete.

Do not allow any data manipulation language (DML) statements directly on the objects in the extended master group in the new master site until execution of this procedure returns successfully, because these DML statements may not be replicated.

```
*/

CONNECT repadmin/repadmin@orc4.world

BEGIN
   DBMS_REPCAT.PREPARE_INSTANTIATED_MASTER (
      extension_id  => :extension_id);
END;
/

CONNECT repadmin/repadmin@orc5.world

BEGIN
   DBMS_REPCAT.PREPARE_INSTANTIATED_MASTER (
      extension_id  => :extension_id);
END;
/

/*
```

Note: You can find the `extension_id` by querying the `DBA_REPSITES_NEW` data dictionary view.

```
/************************ END OF SCRIPT *********************************/
```

Adding New Master Sites to a Quiesced Master Group

You can add new master sites to a quiesced master group in one of the following ways:

- Adding New Master Sites Using the ADD_MASTER_DATABASE Procedure
- Adding New Master Sites with Offline Instantiation Using Export/Import

Typically, you should only use the ADD_MASTER_DATABASE procedure if you have a relatively small master group or if you plan to precreate the replication tables and load the data into them at the new master sites. If this is not the case, the ADD_MASTER_DATABASE procedure may not be a good option because the entire master group is copied over the network. For larger master groups, either precreate the objects in the master group at the new master sites or use offline instantiation.

Adding New Master Sites Using the ADD_MASTER_DATABASE Procedure

You can use the ADD_MASTER_DATABASE procedure to add additional master sites to an existing master group that is quiesced. Executing this procedure replicates existing master objects to the new site. If any master site is lower than 9.0.0 compatibility level, then you must use the following procedure. That is, the master group must be quiesced to extend it with new master sites. You control the compatibility level of a database with the COMPATIBLE initialization parameter.

Meet the following requirements to complete these actions:

Executed As: Replication Administrator

Executed At: Master Definition Site

Replication Status: Quiesced

Complete the following steps to use the ADD_MASTER_DATABASE procedure to add sites to a master group.

> **Note:** If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 7-30 into a text editor and then edit the text to create a script for your environment.

```
/************************ BEGINNING OF SCRIPT *****************************
```

1. Set up the new master site.

Make sure the appropriate schema and database links have been created before adding 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.

See Also:

- "Setting Up Master Sites" section on page 2-4
- "Creating Scheduled Links Between the Master Sites" on page 2-15

```
*/

PAUSE Press <RETURN> to the new master site has been set up.

/*
```

2. Connect to the master definition site as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

3. If the replication status is normal, then change the status to quiesced.

```
*/

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/*
```

4. Use the `ADD_MASTER_DATABASE` procedure to add the new master sites.

   This example assumes that the replicated objects do not exist at the new master site. Therefore, the `copy_rows` parameter is set to `true` to copy the rows in the replicated objects at the master definition site to the new master site, and the `use_existing_objects` parameter is set to `false` so that Oracle Replication creates the replicated objects at the new site. If the replicated objects already exist at the new site but do not contain any data, then set `use_existing_objects` to `true`.

```
*/

BEGIN
   DBMS_REPCAT.ADD_MASTER_DATABASE (
      gname => 'hr_repg',
      master => 'orc4.world',
      use_existing_objects => FALSE,
      copy_rows => TRUE,
      propagation_mode => 'ASYNCHRONOUS');
END;
/

/*
```

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 COUNT(*) FROM DBA_REPCATLOG WHERE GNAME = 'HR_REPG';
```

All of the processing is complete when this statement returns zero (0).

```
*/

PAUSE Press <RETURN> to continue when DBA_REPCATLOG is empty.

/*
```

5. Resume replication activity by using the `RESUME_MASTER_ACTIVITY` procedure.

```
*/

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/************************* END OF SCRIPT **********************************/
```

Adding New Master Sites with Offline Instantiation Using Export/Import

Expanding established replication environments can cause network traffic when you add a new master site to your replication environment using the `ADD_MASTER_DATABASE` procedure. This is caused by propagating the entire contents of the table or materialized view through the network to the new replicated site.

To minimize such network traffic, you can expand your replication 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.

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

Meet the following requirements to complete these actions:

Executed As: Replication Administrator

Executed At: Master Definition Site and New Master Site

Replication Status: Quiesced and Partial

Complete the following steps to use offline instantiation to add sites to a master group.

> **Note:** If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 7-36 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Set up the new master site.

   Make sure the appropriate schema and database links have been created before performing 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.

See Also:

- "Setting Up Master Sites" section on page 2-4
- "Creating Scheduled Links Between the Master Sites" on page 2-15

```
*/

PAUSE Press <RETURN> to the new master site has been set up.

/*
```

2. Connect to the master definition site as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

3. Suspend master activity.

   You need to suspend master activity for the existing master sites before exporting your master data and beginning the offline instantiation process.

```
*/

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/*
```

4. Verify that there are no pending transactions in a separate SQL*Plus session.

   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.

   Check the 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 reexecute the deferred transaction. The following is an example:

```
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` procedure several times, because some administrative operations have multiple steps. The following is an example:

```
BEGIN
   DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN (
      gname => 'hr_repg',
      all_sites => TRUE);
END;
/

*/

PAUSE Press <RETURN> to continue when you have verified that there are no
pending requests.

/*
```

5. Begin offline instantiation procedure.

```
*/

BEGIN
   DBMS_OFFLINE_OG.BEGIN_INSTANTIATION (
      gname => 'hr_repg',
      new_site => 'orc4.world');
END;
/
```

```
/*
```

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 = 'HR_REPG';

*/

PAUSE Press <RETURN> to continue when DBA_REPCATLOG is empty.

/*
```

6. In a separate terminal window, connect as hr/hr 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 Also: *Oracle9i Database Utilities* for additional information

   The following is an example of an export command for the hr schema:

```
exp hr/hr@orc1.world

*/

PAUSE Press <RETURN> to continue when the export is complete.

/*
```

7. Resume partial replication activity.

   Because it may take 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 RESUME_SUBSET_OF_MASTERS procedure in the DBMS_OFFLINE_OG package after the export is complete. In the example below, replication activity is resumed at all master sites except the new master site -- orc4.world.

```
*/
```

```
BEGIN
   DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS (
      gname => 'hr_repg',
      new_site => 'orc4.world');
END;
/

/*
```

8. Connect to the new master site as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc4.world

/*
```

9. Prepare new master site.

   After transferring 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. Make sure you execute the following procedure at the new master site.

```
*/

BEGIN
   DBMS_OFFLINE_OG.BEGIN_LOAD (
      gname => 'hr_repg',
      new_site => 'orc4.world');
END;
/

/*
```

10. In a separate terminal window, import data from export file.

    After importing the export file that you generated earlier, you have transferred the data from your master definition site to your new master site.

    The following is an example of an import command for the `hr` schema:

```
imp hr/hr@orc4.world FULL=y IGNORE=y

*/

PAUSE Press <RETURN> to continue when the import is complete.
```

```
/*
```

11. Complete load process at new master site.

    After importing 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.

```
*/

BEGIN
   DBMS_OFFLINE_OG.END_LOAD (
      gname => 'hr_repg',
      new_site => 'orc4.world');
END;
/

/*
```

12. Connect to the master definition site as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

13. Complete instantiation process.

    After completing the steps at the new master site, you are ready to complete the offline instantiation process. Executing the `END_INSTANTIATION` procedure in the `DBMS_OFFLINE_OG` package completes the process and resumes normal replication activity at all master sites. Make sure you execute the following procedure at the master definition site.

```
*/

BEGIN
   DBMS_OFFLINE_OG.END_INSTANTIATION (
      gname => 'hr_repg',
      new_site => 'orc4.world');
END;
/

/************************* END OF SCRIPT **********************************/
```

Removing a Master Site from a Master Group

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.

Meet the following requirements to complete these actions:

Executed As: Replication Administrator

Executed At: Master Definition Site

Replication Status: Quiesced

Complete the following steps to remove a master site.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 7-38 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect to the master definition site as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

2. If the replication status is normal for the master group, then change the status to quiesced.

```
*/

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/*
```

3. Remove the master site using the REMOVE_MASTER_DATABASES procedure.

```
*/

BEGIN
   DBMS_REPCAT.REMOVE_MASTER_DATABASES (
      gname => 'hr_repg',
      master_list => 'orc4.world');
END;
/

/*
```

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 = 'HR_REPG';

*/

PAUSE Press <RETURN> to continue when DBA_REPCATLOG is empty for the master
group.

/*
```

4. Resume master activity for the master group.

```
*/

BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/************************* END OF SCRIPT **********************************/
```

Removing an Unavailable Master Site

The sites being removed from a master group do not have to be accessible. When a master site will not be available for an extended period of time due to a system or network failure, you might decide to drop the master site from the master group.

However, because the site is unavailable, you most likely cannot suspend replication activity for the master group. You can use the `REMOVE_MASTER_DATABASES` procedure in the `DBMS_REPCAT` package to remove master sites from a master group, even if the master group is not quiesced.

If this is the case, you are responsible for:

- Cleaning the deferred transaction queue
- Removing any data inconsistencies

Specifically, the next time that you suspend replication activity for a master group, you must complete the following steps as soon as possible after the unavailable master sites are removed:

1. Suspend replication activity for the master group. See "SUSPEND_MASTER_ACTIVITY Procedure" on page 20-109 for information.
2. Delete all deferred transactions from each master site where the destination for the transaction is a removed master site. See "DELETE_TRAN Procedure" on page 14-8 for information.
3. Remove all deferred transactions from removed master sites. See "DELETE_TRAN Procedure" on page 14-8 for information.
4. Reexecute or delete all error transactions at each remaining master site. See "Managing the Error Queue" on page 9-18 for information about reexecuting error transactions, and see "DELETE_TRAN Procedure" on page 14-8 for information about removing error transactions.
5. Ensure that no deferred or error transactions exist at each remaining master. If you cannot remove one or more deferred transactions from a remaining master, execute the `DBMS_DEFER_SYS.DELETE_TRAN` procedure at the master site.
6. Ensure that all replicated data is consistent. See Chapter 18, "DBMS_RECTIFIER_DIFF" for information about determining and correcting differences.
7. Resume replication activity for the master group. See "RESUME_MASTER_ACTIVITY Procedure" on page 20-100 for information.

Note: After dropping an unavailable master site from a master group, you should also remove the master group from the dropped site to finish the cleanup.

Updating the Comments Fields in Data Dictionary Views

Several procedures in the DBMS_REPCAT package enable you to 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 20-44.
DBA_REPOBJECT	COMMENT_ON_REPOBJECT(sname IN VARCHAR2, oname IN VARCHAR2, type IN VARCHAR2, comment IN VARCHAR2)	"COMMENT_ON_REPOBJECT Procedure" on page 20-45.
DBA_REPSITES	COMMENT_ON_REPSITES(gname IN VARCHAR2, master IN VARCHAR, comment IN VARCHAR2)	"COMMENT_ON_REPSITES Procedure" on page 20-46.
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 20-41.
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 20-43.
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 20-43.

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 20-48.
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 20-48.
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 20-48.

Using Procedural Replication

Procedural replication can offer performance advantages for large batch-oriented operations operating on large numbers of rows that can be run serially within a replication environment.

A good example of an appropriate application is a purge operation, also referred to as an archive operation, that you run infrequently (for example, once in each quarter) during off hours to remove old data, or data that was "logically" deleted from the online database. An example using procedural replication to purge deleted rows is described in the "Avoiding Delete Conflicts" section in Chapter 5, "Conflict Resolution Concepts and Architecture", of *Oracle9i Replication*.

Restrictions on Procedural Replication

All parameters for a replicated procedure must be `IN` parameters; `OUT` and `IN/OUT` modes are not supported. The following datatypes are supported for these parameters:

- `VARCHAR2`
- `NVARCHAR2`
- `NUMBER`
- `DATE`
- `RAW`
- `ROWID`
- `CHAR`
- `NCHAR`
- Binary LOB (`BLOB`)
- Character LOB (`CLOB`)
- National character LOB (`NCLOB`)
- User-defined datatypes

Oracle cannot detect update conflicts produced by replicated procedures. Replicated procedures must detect and resolve conflicts themselves. Because of the difficulties involved in writing your own conflict resolution routines, it is best to simply avoid the possibility of conflicts altogether.

Adhering to the following guidelines helps you ensure that your tables remain consistent at all sites when you plan to use procedural replication:

- You must disable row-level replication within the body of the deferred procedure. See "Updating the Comments Fields in Data Dictionary Views" on page 7-40.
- Only one replicated procedure should be run at a time, as described in "Serializing Transactions" on page 7-45.
- Deferred transactions should be propagated serially. For more information about guidelines for scheduled links, see *Oracle9i Replication*.
- The replicated procedure must be packaged and the package cannot contain any functions. Standalone deferred procedures and standalone or packaged deferred functions are not currently supported.
- The deferred procedures must reference only locally owned data.
- The procedures should not use locally generated fields, values, or environmentally dependent SQL functions. For example, the procedure should not call `SYSDATE`.
- Your data ownership should be statically partitioned. That is, ownership of a row should not change between sites.
- If you have multiple master groups at a master site, and one or more master groups are quiesced, then you cannot perform procedural replication on any master group at the master site. This restriction is enforced because a procedure in one master group may update objects in another master group. You can only perform procedural replication when all of the master groups on a master site are replicating data normally (that is, when none of the master groups is quiesced).

  For example, if you have a procedure named `sal_raise` in master group A on master site `db1`, then you cannot run the `sal_raise` procedure if master group B on master site `db1` is quiesced, even if master group A is replicating normally.
- When using procedural replication, a procedure call is only propagated to master replication sites. The procedure call is not propagated to materialized view sites. However, procedural replication can be initiated at a materialized view site. In this case, the procedure call is propagated to all of the master sites in the replication environment, but the procedure call is not propagated to any other materialized view sites. Other materialized view sites must pull changes made at the master site by performing a materialized view refresh.

For example, suppose a replication environment includes two master sites named msite1 and msite2 and two materialized view sites named mview1 and mview2. If procedural replication is initiated at mview1, then the procedure is run at mview1 and the procedure call is propagated to the two master sites, msite1 and msite2, where the procedure is also run. However, the procedure call is not propagated to mview2. Therefore, during the next refresh, mview2 pulls down all of the changes made by the procedure at its master site.

User-Defined Types and Procedural Replication

When using procedural replication, the user-defined types and the objects referenced in the procedure must meet the following conditions:

- For an object type, all replication sites must agree about the order of attributes in the object type. You establish the attribute order when you create the object type. Consider the following object type:

  ```
  CREATE TYPE cust_address_typ AS OBJECT
       (street_address     VARCHAR2(40),
        postal_code        VARCHAR2(10),
        city               VARCHAR2(30),
        state_province     VARCHAR2(10),
        country_id         CHAR(2));
  /
  ```

  At all replication sites, street_address must be the first attribute, postal_code must be the second attribute, city must be the third attribute, and so on.

- For an Oracle object, all replication sites must have the same object identifier (OID), schema owner, and type name for each replicated object type.

You can meet these conditions by always using distributed schema management to create or modify any replicated object, including object types, tables with column objects, and object tables. If you do not use distributed schema management to create and modify object types, then replication errors may result.

> **See Also:** *Oracle9i Replication* for more information about type agreement at replication sites

Serializing Transactions

Serial execution ensures that your data remains consistent. The replication facility propagates and executes replicated transactions one at a time. For example, assume that you have two procedures, A and B, that perform updates on local data. Now assume that you perform the following actions, in order:

1. Execute A and B locally.
2. Queue requests to execute other replicas of A and B on other nodes.
3. Commit.

The replicas of A and B on the other nodes are executed completely serially, in the same order that they were committed at the originating site. If A and B execute concurrently at the originating site, however, then they may produce different results locally than they do remotely. Executing A and B serially at the originating site ensures that all sites have identical results. Propagating the transaction serially ensures that A and B are executing in serial order at the target site in all cases.

Alternatively, you could write the procedures carefully, to ensure serialization. For example, you could use `SELECT... FOR UPDATE` for queries to ensure serialization at the originating site and at the target site if you are using parallel propagation.

Generating Support for Replicated Procedures

You must disable row-level replication support at the start of your procedure, and then re-enable support at the end. This operation ensures that any updates that occur as a result of executing the procedure are not propagated to other sites. Row-level replication is enabled and disabled by calling the following procedures, respectively:

- `DBMS_REPUTIL.REPLICATION_ON`
- `DBMS_REPUTIL.REPLICATION_OFF`

See Also:

- "Disabling Replication" on page 9-5
- "REPLICATION_ON Procedure" on page 24-3
- "REPLICATION_OFF Procedure" on page 24-3

When you generate replication support for your replicated package, Oracle creates a wrapper package in the schema of the replication propagator.

Note: Unregistering the current propagator drops all existing generated wrappers in the propagator's schema. Replication support for wrapped stored procedures must be regenerated after you register a new propagator.

The wrapper package has the same name as the original package, but its name is prefixed with the string you supply when you generate replication support for the procedure. If you do not supply a prefix, then Oracle uses the default prefix, `defer_`. The wrapper procedure has the same parameters as the original, along with two additional parameters: `call_local` and `call_remote`. These two `CHAR` parameters determine where the procedure is executed. When `call_local` is `'Y'`, the procedure is executed locally. When `call_remote` is `'Y'`, the procedure will ultimately be executed at all other master sites in the replication environment.

The remote procedures are called directly if you are propagating changes synchronously, or calls to these procedures are added to the deferred transaction queue if you are propagating changes asynchronously. By default, `call_local` is `'N'`, and `call_remote` is `'Y'`.

Oracle generates replication support for a package in two phases. The first phase creates the package specification at all sites. Phase two generates the package body at all sites. These two phases are necessary to support synchronous replication.

For example, suppose you create the package `emp_mgmt` containing the procedure `new_dept`, which takes one argument, `email`. To replicate this package to all master sites in your system, you can use the Replication Management tool to add the package to a master group and then generate replication support for the object. After completing these steps, an application can call procedure in the replicated package as follows:

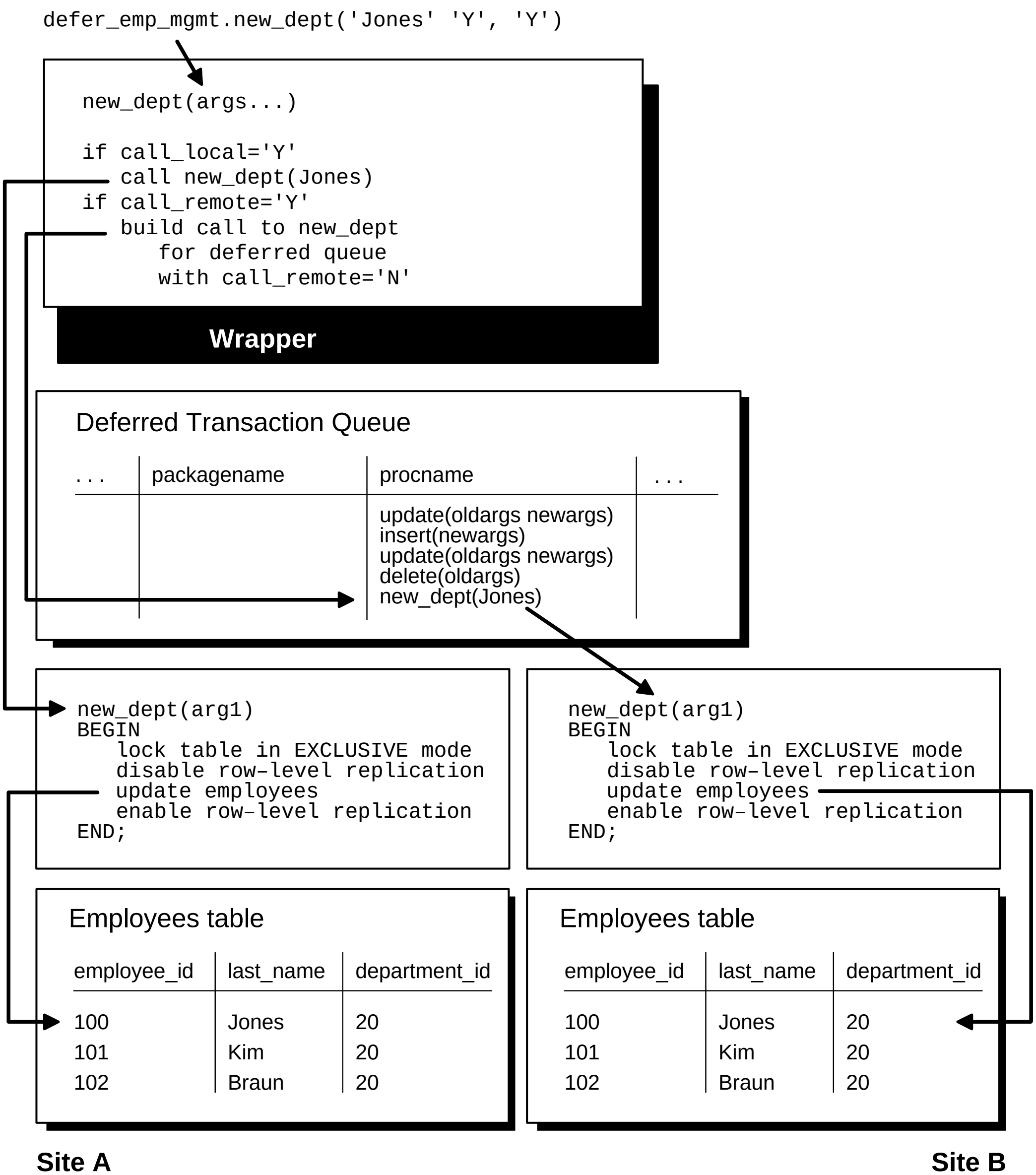
```
BEGIN
defer_emp_mgmt.new_dept( email        => 'jones',
                         call_local   => 'Y',
                         call_remote  => 'Y');
END;
/
```

See Also: The Replication Management tool's online help for more information about managing master groups and replicated objects using the Replication Management tool

As shown in Figure 7–4, the logic of the wrapper procedure ensures that the procedure is called at the local site and subsequently at all remote sites. The logic of the wrapper procedure also ensures that when the replicated procedure is called at the remote sites, `call_remote` is `false`, ensuring that the procedure is not further propagated.

If you are operating in a mixed replication environment with static partitioning of data ownership (that is, if you are not preventing row-level replication), then Oracle Replication preserves the order of operations at the remote node, because both row-level and procedural replication use the same asynchronous queue.

Figure 7–4 Asynchronous Procedural Replication



8

Managing a Materialized View Replication Environment

Materialized view replication provides the flexibility to build data sets to meet the needs of your users and front-end applications, while still meeting the requirements of your security configuration. This chapter describes how to manage materialized view sites with the replication management API.

This chapter contains these topics:

Refreshing Materialized Views

Refreshing a materialized view synchronizes the data in the materialized view's master(s) and the data in the materialized view. You can either refresh all of the materialized views in a refresh group at once, or you can refresh materialized views individually. If you have applications that depend on more than one materialized view at a materialized view site, then Oracle Corporation recommends using refresh groups so that the data is transactionally consistent in all of the materialized views used by the application.

The following example refreshes the `hr_refg` refresh group:

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

See Also: "REFRESH Procedure" on page 19-10 for more information about the `DBMS_REFRESH.REFRESH` procedure

The following example refreshes the `hr.departments_mv` materialized view:

```
BEGIN
   DBMS_MVIEW.REFRESH (
     list   =>  'hr.departments_mv',
     method =>  '?');
END;
/
```

See Also: "REFRESH Procedure" on page 15-12 for more information about the `DBMS_MVIEW.REFRESH` procedure

Changing a Materialized View Group's Master Site

To change the master site of a materialized view group at a level 1 materialized view site to another master site, call the `SWITCH_MVIEW_MASTER` procedure in the `DBMS_REPCAT` package, as shown in the following example:

```
BEGIN
   DBMS_REPCAT.SWITCH_MVIEW_MASTER (
      gname => 'hr_repg',
      master => 'orc3.world');
END;
/
```

In this example, the master site for the `hr_repg` replication group is changed to the `orc3.world` master site. You must call this procedure at the materialized view site whose master site you want to change. The new database must be a master site in the replication environment. When you call this procedure, Oracle uses the new master to perform a full refresh of each materialized view in the local materialized view group. Make sure you have set up the materialized view site to use the new master site before you run the `SWITCH_MVIEW_MASTER` procedure.

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

Note: You cannot switch the master of materialized views that are based on other materialized views (level 2 and greater materialized views). Such a materialized view must be dropped and recreated if you want to base it on a different master.

See Also:

- "SWITCH_MVIEW_MASTER Procedure" on page 20-110 for more information
- "Setting Up Materialized View Sites" on page 2-19

Dropping Materialized View Groups and Objects

You may need to drop replication activity at a materialized view 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 materialized view site.

This section contains the following sections:

- Dropping a Materialized View Groups Created with a Deployment Template
- Dropping a Materialized View Group or Objects Created Manually
- Cleaning Up a Master Site or Master Materialized View Site

Dropping a Materialized View Groups Created with a Deployment Template

If a materialized view group was created with a deployment template, then, before you drop the materialized view group at the remote materialized view site, you need to execute the `DROP_SITE_INSTANTIATION` procedure at the target master site of the materialized view group. In addition to removing the metadata relating to the materialized view 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 materialized view group to drop the materialized view site, while the private version allows the replication administrator to drop a materialized view site on behalf of the materialized view group owner.

Using the Public Version of DROP_SITE_INSTANTIATION

Meet the following requirements to complete these actions:

Executed As:

- Materialized View Group Owner at Master Site
- Materialized View Administrator at Materialized View Site

Executed At:

- Master Site for Target Materialized View Site
- Materialized View Site

Replication Status: Normal

Complete the following steps to drop a materialized view group created with a deployment template.

> **Note:** If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 8-6 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect to the master site as the materialized view group owner.

```
*/
```

```
CONNECT hr/hr@orc3.world

/*
```

2. Drop the instantiated materialized view site from the master site.

```
*/

BEGIN
   DBMS_REPCAT_INSTANTIATE.DROP_SITE_INSTANTIATION(
      refresh_template_name =>  'hr_refg_dt',
      site_name => 'orc3.world');
END;
/

/*
```

3. Connect to the remote materialized view site as the materialized view administrator.

```
*/

CONNECT mviewadmin/mviewadmin@sf.world

/*
```

If you are not able to connect to the remote materialized view site, then the target materialized view group cannot refresh, but the existing data still remains at the materialized view site.

4. Drop the materialized view group.

```
*/

BEGIN
   DBMS_REPCAT.DROP_MVIEW_REPGROUP (
      gname => 'hr_repg',
      drop_contents => TRUE);
END;
/

/*
```

If you want to physically remove the contents of the materialized view group from the materialized view database, then be sure that you specify `TRUE` for the `drop_contents` parameter.

5. Remove the refresh group.

```
*/

BEGIN
   DBMS_REFRESH.DESTROY (
        name => 'hr_refg');
END;
/

/************************* END OF SCRIPT **********************************/
```

Using the Private Version of DROP_SITE_INSTANTIATION

The following steps are to be performed by the replication administrator on behalf of the materialized view group owner. Meet the following requirements to complete these actions:

Executed As:

- Replication Administrator at Master Site
- Materialized View Administrator at Materialized View Site

Executed At:

- Master Site for Target Materialized View Site
- Materialized View Site

Replication Status: Normal

Complete the following steps to drop a materialized view group created with a deployment template.

> **Note:** If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 8-8 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect to the master site as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world
```

```
/*
```

2. Drop the instantiated materialized view site from the master site.

```
*/

BEGIN
   DBMS_REPCAT_RGT.DROP_SITE_INSTANTIATION (
      refresh_template_name =>  'hr_refg_dt',
      user_name => 'hr',
      site_name => 'orc3.world');
END;
/

/*
```

3. Connect to the remote materialized view site as the materialized view administrator.

```
*/

CONNECT mviewadmin/mviewadmin@sf.world

/*
```

If you are unable to connect to the remote materialized view site, then the target materialized view group cannot refresh, but the existing data still remains at the materialized view site.

4. Drop the materialized view group.

```
*/

BEGIN
   DBMS_REPCAT.DROP_MVIEW_REPGROUP (
      gname => 'hr_repg',
      drop_contents => TRUE
      gowner => 'hr');
END;
/

/*
```

If you want to physically remove the contents of the materialized view group from the materialized view database, then be sure that you specify `TRUE` for the `drop_contents` parameter.

5. Remove the refresh group.

```
*/

BEGIN
   DBMS_REFRESH.DESTROY (
        name => 'hr_refg');
END;
/

/************************ END OF SCRIPT **********************************/
```

Dropping a Materialized View Group or Objects Created Manually

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

Ideally, these procedures should be executed while the materialized view is connected to its target master site or master materialized view site. A connection ensures that any related metadata at the master site or master materialized view site is removed. If a connection to the master site or master materialized view site is not possible, then be sure to complete the procedure described in "Cleaning Up a Master Site or Master Materialized View Site" on page 8-10 to manually remove the related metadata.

Dropping a Materialized View Group Created Manually

When it becomes necessary to remove a materialized view group from a materialized view site, use the DROP_MVIEW_REPGROUP procedure to drop a materialized view group. When you execute this procedure and are connected to the target master site or master materialized view site, the metadata for the target materialized view group at the master site or master materialized view site is removed. If you cannot connect, then see "Cleaning Up a Master Site or Master Materialized View Site" on page 8-10 for more information.

Meet the following requirements to complete these actions:

Executed As: Materialized View Administrator

Executed At: Remote Materialized View Site

Replication Status: N/A

Complete the following steps to drop a materialized view group at a materialized view site:

1. Connect to the materialized view site as the materialized view administrator.

```
CONNECT mviewadmin/mviewadmin@mv1.world
```

2. Drop the materialized view group.

```
BEGIN
   DBMS_REPCAT.DROP_MVIEW_REPGROUP (
      gname => 'hr_repg',
      drop_contents => TRUE);
END;
/
```

If you want to physically remove the contents of the materialized view group from the materialized view database, then be sure that you specify TRUE for the drop_contents parameter.

Dropping Objects at a Materialized View Site

When it becomes necessary to remove an individual materialized view from a materialized view site, use the DROP_MVIEW_REPOBJECT procedure API to drop a materialized view. When you execute this procedure and are connected to the target master site or master materialized view site, the metadata for the target materialized view at the master site or master materialized view site is removed. If you cannot connect, then see "Cleaning Up a Master Site or Master Materialized View Site" on page 8-10 for more information.

Meet the following requirements to complete these actions:

Executed As: Materialized View Administrator

Executed At: Remote Materialized View Site

Replication Status: N / A

Complete the following steps to drop an individual materialized view at a materialized view site.

1. Connect to the materialized view site as the materialized view administrator.

```
CONNECT mviewadmin/mviewadmin@mv1.world
```

2. Drop the materialized view.

```
BEGIN
   DBMS_REPCAT.DROP_MVIEW_REPOBJECT (
      sname => 'hr',
      oname => 'employees_mv1',
      type => 'SNAPSHOT',
      drop_objects => TRUE);
END;
/
```

   If you want to physically remove the contents of the materialized view from the materialized view database, then be sure that you specify `TRUE` for the `drop_contents` parameter.

Cleaning Up a Master Site or Master Materialized View Site

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

Cleaning Up After Dropping a Materialized View Group

If you have executed the steps described in "Dropping a Materialized View Group Created Manually" on page 8-8 and were not connected to the master site or master materialized view site, then you are encouraged to complete the following steps to clean up the target master site or master materialized view site.

Meet the following requirements to complete these actions:

Executed As: Replication Administrator

Executed At: Master Site or Master Materialized View Site for Target Materialized View Site

Replication Status: Normal

Complete the following steps to clean up a master site or master materialized view site after dropping a materialized view group:

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 8-13 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect to the master site or master materialized view site as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

2. Unregister the materialized view groups.

```
*/

BEGIN
   DBMS_REPCAT.UNREGISTER_MVIEW_REPGROUP (
      gname => 'hr_repg',
      mviewsite => 'mv1.world');
END;
/

/*
```

3. Purge the materialized view logs of the entries that were marked for the target materialized views. Execute the `PURGE_MVIEW_FROM_LOG` procedure for each materialized view that was in the materialized view groups you unregistered in Step 2.

Note: If for some reason unregistering the materialized view group fails, then you should still complete this step.

```
*/
```

```
BEGIN
   DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
      mviewowner => 'hr',
      mviewname => 'countries_mv1',
      mviewsite => 'mv1.world');
END;
/

BEGIN
   DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
      mviewowner => 'hr',
      mviewname => 'departments_mv1',
      mviewsite => 'mv1.world');
END;
/

BEGIN
   DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
      mviewowner => 'hr',
      mviewname => 'employees_mv1',
      mviewsite => 'mv1.world');
END;
/

BEGIN
   DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
      mviewowner => 'hr',
      mviewname => 'jobs_mv1',
      mviewsite => 'mv1.world');
END;
/

BEGIN
   DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
      mviewowner => 'hr',
      mviewname => 'job_history_mv1',
      mviewsite => 'mv1.world');
END;
/

BEGIN
   DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
      mviewowner => 'hr',
      mviewname => 'locations_mv1',
      mviewsite => 'mv1.world');
END;
/
```

```
BEGIN
   DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
      mviewowner => 'hr',
      mviewname => 'regions_mv1',
      mviewsite => 'mv1.world');
END;
/
```

```
/************************* END OF SCRIPT **********************************/
```

Cleaning Up Individual Materialized View Support

If you have executed the steps described in "Dropping Objects at a Materialized View Site" on page 8-9 and were not connected to the master site or master materialized view site, then you are encouraged to complete the following steps to clean up the target master site or master materialized view site.

Meet the following requirements to complete these actions:

Executed As: Replication Administrator

Executed At: Master Site or Master Materialized View Site for Target Materialized View Site

Replication Status: Normal

Complete the following steps to clean up a master site or master materialized view site after dropping an individual materialized view.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 8-14 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect to the master site or master materialized view site as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

2. Unregister the materialized view.

```
*/

BEGIN
   DBMS_MVIEW.UNREGISTER_MVIEW (
      mviewowner => 'hr',
      mviewname => 'employees_mv1',
      mviewsite => 'mv1.world');
END;
/

/*
```

3. Purge the associated materialized view log of the entries that were marked for the target materialized views.

> **Note:** If for some reason unregistering the materialized view fails, then you should still complete this step.

```
*/

BEGIN
   DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
      mviewowner => 'hr',
      mviewname => 'employees_mv1',
      mviewsite => 'mv1.world');
END;
/

/************************* END OF SCRIPT **********************************/
```

Managing Materialized View Logs

The following sections explain how to manage materialized view logs:

- Altering Materialized View Logs
- Managing Materialized View Log Space
- Reorganizing Master Tables that Have Materialized View Logs
- Dropping a Materialized View Log

Altering Materialized View Logs

After creating a materialized view log, you can alter its storage parameters and support for corresponding materialized views. The following sections explain more about altering materialized view logs. Only the following users can alter a materialized view log:

- The owner of the master table or master materialized view
- A user with the `SELECT` privilege for the master table or master materialized view and `ALTER` privilege on the `MLOG$_`*master_name*, where *master_name* is the name of the master for the materialized view log. For example, if the master table is `employees`, then the materialized view log table name is `MLOG$_employees`.

Altering Materialized View Log Storage Parameters

To alter a materialized view log's storage parameters, use the `ALTER MATERIALIZED VIEW LOG` statement. For example, the following statement alters a materialized view log on the `employees` table in the `hr` schema:

```
ALTER MATERIALIZED VIEW LOG ON hr.employees
  PCTFREE 25
  PCTUSED 40;
```

Altering a Materialized View Log to Add Columns

To add new columns to a materialized view log, use the SQL statement `ALTER MATERIALIZED VIEW LOG`. For example, the following statement alters a materialized view log on the `customers` table in the `sales` schema:

```
ALTER MATERIALIZED VIEW LOG ON hr.employees
  ADD (department_id);
```

> **See Also:** *Oracle9i Replication* for more information about adding columns to a materialized view log

Managing Materialized View Log Space

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

For example, suppose two materialized views were created against the `customers` table in a master site. Oracle refreshes the `customers` materialized view at the `spdb1` database. However, the server that manages the master table and associated materialized view log does not purge the materialized view log rows used during the refresh of this materialized view until the `customers` materialized view at the `spdb2` database also refreshes using these rows.

Because Oracle must wait for all dependent materialized views to refresh before purging rows from a materialized view log, unwanted situations can occur that cause a materialized view log to grow indefinitely when multiple materialized views are based on the same master table or master materialized view. For example, such situations can occur when more than one materialized view is based on a master table or master materialized view and one of the following conditions is true:

- One materialized view is not configured for automatic refreshes and has not been manually refreshed for a long time.
- One materialized view 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 materialized views based on the master table or master materialized view.
- A network or site failure has prevented a master table or master materialized view from becoming aware that a materialized view has been dropped.

Note: If you purge or `TRUNCATE` a materialized view log before a materialized view has refreshed the changes that were deleted, then the materialized view must perform a complete refresh.

Purging Rows from a Materialized View Log

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

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

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

```
BEGIN
   DBMS_MVIEW.PURGE_LOG (
      master => 'hr.employees',
      num    => 1,
      flag   => 'DELETE');
END;
/
```

See Also: "PURGE_LOG Procedure" on page 15-9

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

Truncating a Materialized View Log

If a materialized view log grows and allocates many extents, then purging the log of rows does not reduce the amount of space allocated for the log. In such cases, you should truncate the materialized view log. Only the owner of a materialized view log or a user with the DELETE ANY TABLE system privilege can truncate a materialized view log.

To reduce the space allocated for a materialized view log by truncating it, complete the following steps:

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

   ```
   LOCK TABLE hr.employees IN EXCLUSIVE MODE;
   ```

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

   ```
   CREATE TABLE hr.templog AS SELECT * FROM hr.MLOG$_employees;
   ```

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

   ```
   TRUNCATE hr.MLOG$_employees;
   ```

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

   ```
   INSERT INTO hr.MLOG$_employees SELECT * FROM hr.templog;

   DROP TABLE hr.templog;
   ```

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

   ```
   ROLLBACK;
   ```

Note: Any changes made to the master table or master materialized view 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.

Reorganizing Master Tables that Have Materialized View Logs

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

> **Note:** These sections do not discuss online redefinition of tables. Online redefinition is not allowed on master tables with materialized view logs, master materialized views, or materialized views. Online redefinition is allowed only on master tables that do not have materialized view logs. See the *Oracle9i Database Administrator's Guide* for more information about online redefinition of tables.

Reorganization Notification

When you reorganize a table, any ROWID information of the materialized view log must be invalidated. Oracle detects a table reorganization automatically only if the table is truncated as part of the reorganization.

If the table is not truncated, then Oracle must be notified of the table reorganization. To support table reorganizations, two procedures in the DBMS_MVIEW package, BEGIN_TABLE_REORGANIZATION and 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 materialized view log. The inputs are the owner and name of the master to be reorganized. There is no output.

> **See Also:** "Method 2 for Reorganizing Table employees" on page 8-21

Truncating Masters

When a table is truncated, its materialized view log is also truncated. However, for primary key materialized views, you can preserve the materialized view log, allowing fast refreshes to continue. Although the information stored in a materialized view log is preserved, the materialized view log becomes invalid with respect to rowids when the master is truncated. The rowid information in the materialized view log will seem to be newly created and cannot be used by rowid materialized views for fast refresh.

The PRESERVE MATERIALIZED VIEW LOG option is the default. Therefore, if you specify the PRESERVE MATERIALIZED VIEW LOG option or no option, then the information in the master's materialized view log is preserved, but current rowid materialized views can use the log for a fast refresh only *after* a complete refresh has been performed.

Note: To ensure that any previously fast refreshable materialized view is still refreshable, follow the guidelines in "Methods of Reorganizing a Database Table" on page 8-20.

If the PURGE MATERIALIZED VIEW LOG option is specified, then the materialized view log is purged along with the master.

Examples Either of the following two statements preserves materialized view log information when the master table named orders is truncated:

```
TRUNCATE TABLE hr.employees PRESERVE MATERIALIZED VIEW LOG;
TRUNCATE TABLE hr.employees;
```

The following statement truncates the materialized view log along with the master table:

```
TRUNCATE TABLE hr.employees PURGE MATERIALIZED VIEW 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. Direct Loader can cause reordering of the columns, which could invalidate the log information used in subquery and LOB materialized views.

Method 1 for Reorganizing Table employees Complete the following steps:

1. Call `DBMS_MVIEW.BEGIN_TABLE_REORGANIZATION` for table `employees`.
2. Rename table `employees` to `employees_old`.
3. Create table `employees` as `SELECT * FROM employees_old`.
4. Call `DBMS_MVIEW.END_TABLE_REORGANIZATION` for new table `employees`.

> **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 `BEGIN_TABLE_REORGANIZATION` and `END_TABLE_REORGANIZATION`.

Method 2 for Reorganizing Table employees Complete the following steps:

1. Call `DBMS_MVIEW.BEGIN_TABLE_REORGANIZATION` for table `employees`.
2. Export table `employees`.
3. Truncate table `employees` with `PRESERVE MATERIALIZED VIEW LOG` option.
4. Import table `employees` using conventional path.
5. Call `DBMS_MVIEW.END_TABLE_REORGANIZATION` for new table `employees`.

> **Caution:** When you truncate masters as part of a reorganization, you must use the `PRESERVE MATERIALIZED VIEW LOG` clause of the truncate table DDL.

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

Method 3 for Reorganizing Table employees Complete the following steps:

1. Call `DBMS_MVIEW.BEGIN_TABLE_REORGANIZATION` for table `employees`.
2. Export table `employees`.
3. Rename table `employees` to `employees_old`.
4. Import table `employees` using conventional path.
5. Call `DBMS_MVIEW.END_TABLE_REORGANIZATION` for new table `employees`.

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 `BEGIN_TABLE_REORGANIZATION` and `END_TABLE_REORGANIZATION`.

Method 4 for Reorganizing Table employees Complete the following steps:

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

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 `BEGIN_TABLE_REORGANIZATION` and `END_TABLE_REORGANIZATION`.

See Also:

- "BEGIN_TABLE_REORGANIZATION Procedure" on page 15-3
- "END_TABLE_REORGANIZATION Procedure" on page 15-4

Dropping a Materialized View Log

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

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

Here, a master can be a master table or a master materialized view. To delete a materialized view log, execute the DROP MATERIALIZED VIEW LOG statement in SQL*Plus. For example, the following statement deletes the materialized view log for a table named customers in the sales schema:

```
DROP MATERIALIZED VIEW LOG ON hr.employees;
```

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

Performing an Offline Instantiation of a Materialized View Site Using Export/Import

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

To minimize such network traffic, you can expand your replication 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.

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 materialized view group at a materialized view site using the offline instantiation process. In some cases, it is even more useful for materialized views considering that the target computer could very well be a laptop using a modem connection.

The following script performs an offline instantiation for a new materialized view group. Meet the following requirements to complete these actions:

Executed As:

- Replication Administrator at Master Site
- Materialized View Administrator at New Materialized View Site

Executed At:

- Master Site for Target Materialized View Site
- New Materialized View Site

Replication Status: Normal

Complete the following steps to perform an offline instantiation of a materialized view site using export/import.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 8-34 into a text editor and then edit the text to create a script for your environment.

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. In a separate terminal window, set up the new materialized view site.

   Make sure that the appropriate schema and database links have been created before you perform the offline instantiation of your materialized view.

   See Also: "Setting Up Materialized View Sites" on page 2-19

   ```
   */

   PAUSE Press <RETURN> to continue the new materialized view site is set up.

   /*
   ```

2. Connect to the master site as the replication administrator.

   ```
   */

   CONNECT repadmin/repadmin@orc1.world

   /*
   ```

3. Create the necessary materialized view logs.

   If materialized view logs do not already exist for the target master tables, then create them at the target master site.

   ```
   */

   CREATE MATERIALIZED VIEW LOG ON hr.countries;
   CREATE MATERIALIZED VIEW LOG ON hr.departments;
   CREATE MATERIALIZED VIEW LOG ON hr.employees;
   CREATE MATERIALIZED VIEW LOG ON hr.jobs;
   ```

```
CREATE MATERIALIZED VIEW LOG ON hr.job_history;
CREATE MATERIALIZED VIEW LOG ON hr.locations;
CREATE MATERIALIZED VIEW LOG ON hr.regions;

/*
```

4. Create temporary materialized views at the master site. These materialized views contain the data that you transfer to your new materialized view site using the export file.

> **Note:** If you added any of the conflict resolution routines described in Chapter 6, "Configure Conflict Resolution", then you may have additional columns in your tables. Be certain to include these additional columns in the `SELECT` statements below. Updatable materialized views require that you explicitly select all columns in the master table. So, do not use `SELECT *` statements.

```
*/

CREATE MATERIALIZED VIEW hr.countries_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      country_id, country_name, region_id
      FROM hr.countries;

CREATE MATERIALIZED VIEW hr.departments_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      department_id, department_name, manager_id, location_id
      FROM hr.departments;

CREATE MATERIALIZED VIEW hr.employees_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      employee_id, first_name, last_name, email, phone_number,
      hire_date, job_id, salary, commission_pct, manager_id,
      department_id
      FROM hr.employees;

CREATE MATERIALIZED VIEW hr.jobs_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      job_id, job_title, min_salary, max_salary
      FROM hr.jobs;
```

```
CREATE MATERIALIZED VIEW hr.job_history_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      employee_id, start_date, end_date, job_id, department_id
      FROM hr.job_history;

CREATE MATERIALIZED VIEW hr.locations_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      location_id, street_address, postal_code, city,
      state_province, country_id
      FROM hr.locations;

CREATE MATERIALIZED VIEW hr.regions_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      region_id, region_name
      FROM hr.regions;

/*
```

5. In a separate terminal window, connect as the owner of the materialized views to export the temporary materialized views you created in Step 4.

   Use the Oracle Export utility to generate the export file that you will transfer to the new materialized view site. The export file will contain the base tables of your temporary materialized views.

   Note: The following example is for Oracle8*i* and higher databases only. Base tables in database versions earlier than Oracle8*i* are preceded by the `SNAP$` prefix (that is, `SNAP$_employees_mv`).

   See Also: *Oracle9i Database Utilities* for additional information about exporting

   The following is an example of an export command for the `hr` schema.

```
exp hr/hr@orc1.world TABLES='countries_mv','departments_mv','employees_mv',
'jobs_mv','job_history_mv','locations_mv','regions_mv'

*/

PAUSE Press <RETURN> to continue when the export is complete.

/*
```

6. Connect to the new materialized view site as SYSTEM user.

```
*/

CONNECT system/manager@mview.world

/*
```

7. Create necessary schema and database link at the materialized view site, if they do not exist.

   Before you perform the offline instantiation of your materialized views, create the schema that will contain the materialized views at the new materialized view site and the database link from the materialized view site to the master site. The materialized views must be in the same schema that contains the master objects at the master site.

```
*/

CREATE TABLESPACE demo_mview
 DATAFILE 'demo_mview.dbf' SIZE 10M AUTOEXTEND ON
 EXTENT MANAGEMENT LOCAL AUTOALLOCATE;

CREATE TEMPORARY TABLESPACE temp_mview
 TEMPFILE 'temp_mview.dbf' SIZE 5M AUTOEXTEND ON;

CREATE USER hr IDENTIFIED BY hr;

ALTER USER hr DEFAULT TABLESPACE demo_mview
              QUOTA UNLIMITED ON demo_mview;

ALTER USER hr TEMPORARY TABLESPACE temp_mview;

GRANT
  CREATE SESSION,
  CREATE TABLE,
  CREATE PROCEDURE,
  CREATE SEQUENCE,
  CREATE TRIGGER,
  CREATE VIEW,
  CREATE SYNONYM,
  ALTER SESSION,
  CREATE MATERIALIZED VIEW,
```

```
  ALTER ANY MATERIALIZED VIEW,
  CREATE DATABASE LINK,
TO hr;

CONNECT hr/hr@mview.world

CREATE DATABASE LINK orc1.world CONNECT TO hr IDENTIFIED by hr;

/*
```

8. Connect to the new materialized view site as the materialized view administrator.

```
*/

CONNECT mviewadmin/mviewadmin@mview.world

/*
```

9. Create an empty materialized view group.

Run the DBMS_REPCAT.CREATE_MVIEW_REPGROUP procedure at the new materialized view site to create an empty materialized view group to which you will add your materialized views.

```
*/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPGROUP (
      gname => 'hr_repg',
      master => 'orc1.world',
      propagation_mode => 'ASYNCHRONOUS');
END;
/

/*
```

10. Prepare the materialized view site for offline instantiation.

The DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD procedure creates the necessary support mechanisms for the new materialized views. This step also adds the new materialized views to the materialized view group that you created in the previous step. Be sure to execute the BEGIN_LOAD procedure for each materialized view that you will be importing.

```
*/
```

```
BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      master_site => 'orc1.world',
      snapshot_oname => 'countries_mv');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      master_site => 'orc1.world',
      snapshot_oname => 'departments_mv');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      master_site => 'orc1.world',
      snapshot_oname => 'employees_mv');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      master_site => 'orc1.world',
      snapshot_oname => 'jobs_mv');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      master_site => 'orc1.world',
      snapshot_oname => 'job_history_mv');
END;
/
```

```
BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      master_site => 'orc1.world',
      snapshot_oname => 'locations_mv');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      master_site => 'orc1.world',
      snapshot_oname => 'regions_mv');
END;
/

/*
```

11. In a separate terminal window, connect as the owner of the materialized views to import at the new materialized view site.

    Use the Oracle Import utility to import the file that you exported in Step 5. Make sure that you import your data as the same user who exported the data. This user `hr` in the following example:

```
imp hr/hr@mview.world FULL=y IGNORE=y

*/

PAUSE Press <RETURN> to continue when the import is complete.

/*
```

12. Complete the offline instantiation.

    Execute the `DBMS_OFFLINE_SNAPSHOT.END_LOAD` procedure to finish the offline instantiation of the imported materialized views.

```
*/
```

```
BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      snapshot_oname => 'countries_mv');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      snapshot_oname => 'departments_mv');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      snapshot_oname => 'employees_mv');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      snapshot_oname => 'jobs_mv');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      snapshot_oname => 'job_history_mv');
END;
/
```

```
BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      snapshot_oname => 'locations_mv');
END;
/

BEGIN
   DBMS_OFFLINE_SNAPSHOT.END_LOAD (
      gname => 'hr_repg',
      sname => 'hr',
      snapshot_oname => 'regions_mv');
END;
/

/*
```

13. Connect as the owner of the materialized views at the materialized view site.

```
*/

CONNECT hr/hr@mview.world

/*
```

14. Refresh materialized views to register them at master site.

In addition to retrieving the latest changes from the master tables, refreshing the materialized views at the new materialized view site registers the offline instantiated materialized views at the target master site.

```
*/

BEGIN
   DBMS_MVIEW.REFRESH ('countries_mv');
END;
/

BEGIN
   DBMS_MVIEW.REFRESH ('departments_mv');
END;
/
```

```
BEGIN
   DBMS_MVIEW.REFRESH ('employees_mv');
END;
/

BEGIN
   DBMS_MVIEW.REFRESH ('jobs_mv');
END;
/

BEGIN
   DBMS_MVIEW.REFRESH ('job_history_mv');
END;
/

BEGIN
   DBMS_MVIEW.REFRESH ('locations_mv');
END;
/

BEGIN
   DBMS_MVIEW.REFRESH ('regions_mv');
END;
/

/*
```

15. Connect to the master site as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

16. Delete the temporary materialized views you created in Step 4 at the master site.

```
*/

DROP MATERIALIZED VIEW hr.countries_mv;
DROP MATERIALIZED VIEW hr.departments_mv;
DROP MATERIALIZED VIEW hr.employees_mv;
DROP MATERIALIZED VIEW hr.jobs_mv;
```

```
    DROP MATERIALIZED VIEW hr.job_history_mv;
    DROP MATERIALIZED VIEW hr.locations_mv;
    DROP MATERIALIZED VIEW hr.regions_mv;

    /*

/************************* END OF SCRIPT **********************************/
```

Using a Group Owner for a Materialized View Group

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

Complete the following steps to use a group owner.

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 8-40 into a text editor and then edit the text to create a script for your environment.

See Also: *Oracle9i Replication* for a complete description of using group owners and the advantages of using multiple data sets

```
/************************* BEGINNING OF SCRIPT ******************************
```

1. Connect to the materialized view site as the materialized view administrator.

```
*/

CONNECT mviewadmin/mviewadmin@mv1.world

/*
```

2. Create materialized view group with group owner (`gowner`) `bob` using the `CREATE_MVIEW_REPGROUP` procedure.

   The replication group that you specify in the `gname` parameter must match the name of the replication group that you are replicating at the target master site or master materialized view site. The `gowner` parameter enables you to specify an additional identifier that lets you create multiple materialized view groups based on the same replication group at the same materialized view site.

   In this example, materialized view groups are created for the group owners `bob` and `jane`, and these two materialized view groups are based on the same replication group.

```
*/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPGROUP (
      gname => 'hr_repg',
      master => 'orc1.world',
      propagation_mode => 'ASYNCHRONOUS',
      gowner => 'bob');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPGROUP (
      gname => 'hr_repg',
      master => 'orc1.world',
      propagation_mode => 'ASYNCHRONOUS',
      gowner => 'jane');
END;
/

/*
```

3. Create the materialized views owned by `bob`.

   The `gowner` value used when creating your materialized view objects must match the `gowner` value specified when you created the materialized view group in the previous procedures. After creating the materialized view groups, you can create materialized views based on the same master in the `hr_repg` materialized view group owned by `bob` and `jane`.

Caution: Each object must have a unique name. When using a `gowner` to create multiple materialized view 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 MATERIALIZED VIEW hr.countries_bob`). Such a naming method ensures that you do not create any objects with conflicting names.

Whenever you create a materialized view, always specify the schema name of the table owner in the query for the materialized view. In the examples below, `hr` is specified as the owner of the table in each query.

```
*/

CREATE MATERIALIZED VIEW hr.countries_bob
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.countries@orc1.world;

CREATE MATERIALIZED VIEW hr.departments_bob
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.departments@orc1.world;

CREATE MATERIALIZED VIEW hr.employees_bob
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.employees@orc1.world;

CREATE MATERIALIZED VIEW hr.jobs_bob
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.jobs@orc1.world;

CREATE MATERIALIZED VIEW hr.job_history_bob
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.job_history@orc1.world;

CREATE MATERIALIZED VIEW hr.locations_bob
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.locations@orc1.world;

CREATE MATERIALIZED VIEW hr.regions_bob
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.regions@orc1.world;
```

```
/*
```

4. Create the materialized views owned by `jane`.

```
*/

CREATE MATERIALIZED VIEW hr.departments_jane
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.departments@orc1.world;

CREATE MATERIALIZED VIEW hr.employees_jane
  REFRESH FAST WITH PRIMARY KEY FOR UPDATE
  AS SELECT * FROM hr.employees@orc1.world;

/*
```

5. Add the materialized views owned by `bob` to the materialized view group.

```
*/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'countries_bob',
      type => 'SNAPSHOT',
      min_communication => TRUE,
      gowner => 'bob');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'departments_bob',
      type => 'SNAPSHOT',
      min_communication => TRUE,
      gowner => 'bob');
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'employees_bob',
      type => 'SNAPSHOT',
      min_communication => TRUE,
      gowner => 'bob');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'jobs_bob',
      type => 'SNAPSHOT',
      min_communication => TRUE,
      gowner => 'bob');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'job_history_bob',
      type => 'SNAPSHOT',
      min_communication => TRUE,
      gowner => 'bob');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'locations_bob',
      type => 'SNAPSHOT',
      min_communication => TRUE,
      gowner => 'bob');
END;
/
```

```
BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'regions_bob',
      type => 'SNAPSHOT',
      min_communication => TRUE,
      gowner => 'bob');
END;
/

/*
```

6. Add the materialized views owned by `jane` to the materialized view group.

```
*/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'departments_jane',
      type => 'SNAPSHOT',
      min_communication => TRUE,
      gowner => 'jane');
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
      gname => 'hr_repg',
      sname => 'hr',
      oname => 'employees_jane',
      type => 'SNAPSHOT',
      min_communication => TRUE,
      gowner => 'jane');
END;
/
```

7. Add your materialized views to a refresh group.

See Also: Chapter 5, "Create Materialized View Group" (Step 6) for more information about adding materialized views to a refresh group

```
/************************* END OF SCRIPT **********************************/
```

9

Managing Replication Objects and Queues

This chapter illustrates how to manage the replication objects and queues in your replication environment using the replication management API. This chapter contains these topics:

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 replication environment to fail.

Altering a Replicated Object in a Quiesced Master Group

Use the `ALTER_MASTER_REPOBJECT` procedure in the `DBMS_REPCAT` package to alter the characteristics of your replicated objects in a quiesced master group. From the example below, notice that you simply include the necessary DDL within the procedure call (see the `ddl_text` parameter).

If any master site is lower than 9.0.0 compatibility level, then you must use the following procedure. That is, the master group must be quiesced to modify a replicated object. You control the compatibility level of a database with the `COMPATIBLE` initialization parameter.

Meet the following requirements to complete these actions:

Executed As: Replication Administrator

Executed At: Master Definition Site

Replication Status: Quiesced

Complete the following steps to alter a replicated object in a quiesced master group.

Note:

- If your master site is running Oracle release 8.1.7 or higher in a single master environment and you are making a safe change to a replicated object, then you may not need to quiesce the master group. See the "ALTER_MASTER_REPOBJECT Procedure" on page 20-29 for information about when quiesce is not required.
- If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the "END OF SCRIPT" line on page 9-4 into a text editor and then edit the text to create a script for your environment.

```
/************************ BEGINNING OF SCRIPT ******************************
```

1. Connect to the master definition site as the replication administrator.

```
*/

CONNECT repadmin/repadmin@orc1.world

/*
```

2. If necessary, then quiesce the master group. See the "ALTER_MASTER_REPOBJECT Procedure" on page 20-29 for information about when quiesce is not required.

```
*/

BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/

/*
```

3. In a separate SQL*Plus session, check the status of the master group you are quiescing, and do not proceed until the group's status is `QUIESCED`.

   To check the status, run the following query:

```
SELECT GNAME, STATUS FROM DBA_REPGROUP;

*/

PAUSE Press <RETURN> to continue when the master group's status is QUIESCED.

/*
```

4. Alter the replicated object.

```
*/

BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      sname => 'hr',
      oname => 'employees',
      type => 'TABLE',
      ddl_text => 'ALTER TABLE hr.employees ADD (timestamp DATE)';
END;
/
```

```
/*
```

5. Regenerate replication support for the altered object.

```
*/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'hr',
      oname => 'employees',
      type => 'TABLE',
      min_communication => TRUE);
END;
/

/*
```

6. In a separate SQL*Plus session, check if `DBA_REPCATLOG` is empty, and do not proceed until it is empty.

   Execute the following `SELECT` statement in another SQL*Plus session to monitor the `DBA_REPCATLOG` view:

```
SELECT * FROM DBA_REPCATLOG WHERE GNAME = 'HR_REPG';

*/

PAUSE Press <RETURN> to continue when DBA_REPCATLOG is empty.

/*
```

7. Resume replication activity.

```
*/

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

/************************* END OF SCRIPT **********************************/
```

Modifying Tables without Replicating the Modifications

You may have a situation in which you need to modify a replicated object, but you do not want this modification replicated to the other sites in the replication environment. For example, you may want to disable replication in the following situations:

- When you are using procedural replication to propagate a change, always disable row-level replication at the start of your procedure.
- You may need to disable replication in triggers defined on replicated tables to avoid replicating trigger actions multiple times. See "Ensuring That Replicated Triggers Fire Only Once" on page 9-7.
- Sometimes when you manually resolve a conflict, you may not want to replicate this modification to the other copies of the table.

  You may need to do this, for example, if you need to correct the state of a record at one site so that a conflicting replicated update will succeed when you reexecute the error transaction. Or, you may use an unreplicated modification to undo the effects of a transaction at its origin site because the transaction could not be applied at the destination site. In this example, you can use the Replication Management tool to delete the conflicting transaction from the destination site.

To modify tables without replicating the modifications, use the `REPLICATION_ON` and `REPLICATION_OFF` procedures in the `DBMS_REPUTIL` package. These procedures take no arguments and are used as flags by the generated replication triggers.

> **Note:** To enable and disable replication, you must have the `EXECUTE` privilege on the `DBMS_REPUTIL` package.

Disabling Replication

The `DBMS_REPUTIL.REPLICATION_OFF` procedure sets the state of an internal replication variable for the current session to `false`. Because all replicated triggers check the state of this variable before queuing any transactions, modifications made to the replicated tables that use row-level replication do not result in any queued deferred transactions.

Caution: Turning replication on or off affects only the current session. That is, other users currently connected to the same server are not restricted from placing committed changes in the deferred transaction queue.

If you are using procedural replication, then call `REPLICATION_OFF` at the start of your procedure, as shown in the following example. This ensures that the replication facility does not attempt to use row-level replication to propagate the changes that you make.

```
CREATE OR REPLACE PACKAGE update AS
  PROCEDURE update_emp(adjustment IN NUMBER);
END;
/

CREATE OR REPLACE PACKAGE BODY update AS
  PROCEDURE update_emp(adjustment IN NUMBER) IS
  BEGIN
   --turn off row-level replication for set update
   DBMS_REPUTIL.REPLICATION_OFF;
   UPDATE emp . . .;
   --re-enable replication
   DBMS_REPUTIL.REPLICATION_ON;
  EXCEPTION WHEN OTHERS THEN
   . . .
   DBMS_REPUTIL.REPLICATION_ON;
  END;
END;
/
```

Reenabling the Replication Facility

After resolving any conflicts, or at the end of your replicated procedure, be certain to call `DBMS_REPUTIL.REPLICATION_ON` to resume normal replication of changes to your replicated tables or materialized views. This procedure takes no arguments. Calling `REPLICATION_ON` sets the internal replication variable to `true`.

Ensuring That Replicated Triggers Fire Only Once

If you have defined a replicated trigger on a replicated table, then you may need to ensure that the trigger fires only once for each change that you make. Typically, you only want the trigger to fire when the change is first made, and you do not want the remote trigger to fire when the change is replicated to the remote site.

You should check the value of the `DBMS_REPUTIL.FROM_REMOTE` package variable at the start of your trigger. The trigger should update the table only if the value of this variable is `false`.

Alternatively, you can disable replication at the start of the trigger and re-enable it at the end of the trigger when modifying rows other than the one that caused the trigger to fire. Using this method, only the original change is replicated to the remote sites. Then the replicated trigger fires at each remote site. Any updates performed by the replicated trigger are not pushed to any other sites.

Using this approach, conflict resolution is not invoked. Therefore, you must ensure that the changes resulting from the trigger do not affect the consistency of the data.

Converting a LONG Column to a LOB Column in a Replicated Table

LOB columns can be replicated, but `LONG` columns cannot be replicated. You can convert the datatype of a `LONG` column to a `CLOB` column and the datatype of a `LONG_RAW` column to a `BLOB` column.

Converting a `LONG` column to a LOB column can result in increased network bandwidth requirements because the data in such a column is replicated after conversion. Make sure you have adequate network bandwidth before completing the procedure in this section.

> **See Also:** *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for more information about applications and `LONG` to LOB conversion

Complete the following steps to convert a `LONG` column to a LOB column in a replicated table:

1. Make sure the data in the `LONG` column is consistent at all replication sites.

   If a table containing a `LONG` column is configured as a master table, then Oracle does not replicate changes to the data in the `LONG` column. Therefore, the data in the `LONG` column may not match at all of your replication sites. You must make sure the data in the `LONG` column matches at all master sites before proceeding.

2. Connect to the master definition site as the replication administrator. For example:

```
CONNECT repadmin/repadmin@orc1.world
```

3. If the replication status is normal, then change the status to quiesced. For example:

```
BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      gname => 'sales_mg');
END;
/
```

4. Run the ALTER_MASTER_REPOBJECT procedure in the DBMS_REPCAT package to convert the LONG column to a LOB column. For example:

```
BEGIN
   DBMS_REPCAT.ALTER_MASTER_REPOBJECT (
      sname => 'staff',
      oname => 'positions',
      type => 'TABLE',
      ddl_text => 'ALTER TABLE positions MODIFY (job_desc CLOB)');
END;
/
```

   A LONG_RAW column can be converted to a BLOB column using a similar ALTER TABLE statement.

5. Regenerate replication support for the altered master table. For example:

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname => 'staff',
      oname => 'positions',
      type => 'TABLE',
      min_communication => TRUE);
END;
/
```

6. Resume replication. For example:

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

7. If materialized views are based on the altered table at any of the master sites, then rebuild these materialized views.

Determining Differences Between Replicated Tables

It is possible for the differences to arise in replicated tables. When administering a replication 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.

Using the DIFFERENCES Procedure

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.

Using the RECTIFY Procedure

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 `employee_id`, `salary`, and `department_id` columns of the `employees` table, then your `CREATE` statement would need to be similar to the following:

```
CREATE TABLE hr.missing_rows_data (
  employee_id     NUMBER(6),
  salary          NUMBER(8,2),
  department_id   NUMBER(4));
```

   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 hr.missing_rows_location (
  present     VARCHAR2(128),
  absent      VARCHAR2(128),
  r_id        ROWID);
```

4. Suspend replication activity for the replication 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.

```
BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/
```

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

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

```
BEGIN
   DBMS_RECTIFIER_DIFF.DIFFERENCES (
      sname1              =>   'hr',
      oname1              =>   'employees',
      reference_site      =>   'ny.world',
      sname2              =>   'hr',
      oname2              =>   'employees',
      comparison_site     =>   'sf.world',
      where_clause        =>   '',
      column_list         =>   'employee_id,salary,department_id',
      missing_rows_sname  =>   'hr',
      missing_rows_oname1 =>   'missing_rows_data',
      missing_rows_oname2 =>   'missing_rows_location',
      missing_rows_site   =>   'ny.world',
      max_missing         =>    500,
      commit_rows         =>    50);
END;
/
```

   Figure 9–1 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 9–1 Determining Differences Between Replicas

employees Table at NY.COM

employee_id	last_name	department_id	salary	commission_pct
100	Jones	20	55,000	.4
101	Kim	20	62,000	.25
102	Braun	20	43,500	.1

employees Table at SF.COM

employee_id	last_name	department_id	salary	commission_pct
100	Jones	20	55,000	.4
101	Kim	20	62,000	.3
102	Braun	20	43,500	.1
103	Rama	20	48,750	.35

Replicas

missing_rows_data Table

employee_id	salary	commission_pct	rowid
101	62,000	.25	000015E8.0000.0002
101	62,000	.3	000015E8.0001.0002
103	48,750	.35	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 `RECTIFY` procedure in the as shown in the `DBMS_RECTIFIER_DIFF` package following example:

```
BEGIN
   DBMS_RECTIFIER_DIFF.RECTIFY (
      sname1              =>   'hr',
      oname1              =>   'employees',
      reference_site      =>   'ny.world',
      sname2              =>   'hr',
      oname2              =>   'employees',
      comparison_site     =>   'sf.world',
      column_list         =>   'employee_id,salary,department_id',
      missing_rows_sname  =>   'hr',
      missing_rows_oname1 =>   'missing_rows_data',
      missing_rows_oname2 =>   'missing_rows_location',
      missing_rows_site   =>   'ny.world',
      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 DELETE operations and then performs all of the INSERT operations. 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, then you must ensure that they are not violated when you call RECTIFY. You may need to update the table directly using the information in the missing rows table. If so, then be sure 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.

```
BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/
```

Managing the Deferred Transactions Queue

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 process for pushing the deferred transaction queue is the same at master sites and materialized view sites.

Pushing the Deferred Transaction Queue

Master sites are configured to push the deferred transaction queue automatically at set intervals. At materialized view sites, if you do not automatically propagate the transactions in your deferred transaction queue during the refresh of your materialized view, then you must complete the following steps to propagate changes made to the updatable materialized view to its master table or master materialized view.

This example illustrates pushing the deferred transaction queue at a materialized view site, but the process is the same at master sites and materialized view sites.

Executed As: Materialized View Administrator

Executed At: Materialized View Site

Complete the following steps:

1. Connect to the materialized view site as the materialized view administrator.

   ```
   CONNECT mviewadmin/mviewadmin@mv1.world
   ```

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

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

3. Execute the `DBMS_DEFER_SYS.PUSH` function for each 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;
/
```

Run the PUSH procedure for each destination that was returned in the SELECT statement you ran in Step 2.

Purging the Deferred Transaction Queue

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

This example illustrates purging the deferred transaction queue at a materialized view site, but the process is the same at master sites and materialized view sites.

Executed As: Materialized View Administrator

Executed At: Materialized View Site

Complete the following steps:

1. Connect to the materialized view site as the materialized view administrator.

```
CONNECT mviewadmin/mviewadmin@mv1.world
```

2. Purge the deferred transaction queue.

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

Note: If you use the `purge_method_quick` parameter, 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 the "Usage Notes" for `DBMS_DEFER_SYS.PURGE` on page 14-15 for details.

Using the AnyData Type to Determine the Value of an Argument in a Deferred Call

If you are using column objects, collections, or `REF`s in a replicated table, then you can use the `GET_AnyData_ARG` function in the `DBMS_DEFER_QUERY` package to determine the value of an argument in a deferred call that involves one of these user-defined types.

The following example illustrates how to use the `GET_AnyData_ARG` function. This example uses the following user-defined types in the `oe` sample schema.

```
CREATE TYPE phone_list_typ AS VARRAY(5) OF VARCHAR2(25);
/

CREATE TYPE warehouse_typ AS OBJECT
    (warehouse_id       NUMBER(3),
     warehouse_name     VARCHAR2(35),
     location_id        NUMBER(4)
    );
/

CREATE TYPE inventory_typ AS OBJECT
    (product_id          NUMBER(6),
     warehouse           warehouse_typ,
     quantity_on_hand    NUMBER(8)
    );
/

CREATE TYPE inventory_list_typ AS TABLE OF inventory_typ;
/
```

The following procedure retrieves the argument value for collection, object, and `REF` instances of calls stored in the deferred transactions queue. This procedure assumes that the call number and transaction id are available.

The user who creates the procedure must have EXECUTE privilege on the DBMS_DEFER_QUERY package and must have CREATE PROCEDURE privilege. This example uses the oe sample schema. Therefore, to run the example, you must grant the oe user these privileges.

```
CONNECT system/manager as sysdba

GRANT EXECUTE ON DBMS_DEFER_QUERY TO oe;

GRANT CREATE PROCEDURE TO oe;

CONNECT oe/oe@orc1.world

CREATE OR REPLACE PROCEDURE get_userdef_arg AS
  call_no      NUMBER := 0;
  txn_id       VARCHAR2(128) := 'xx.xx.xx';
  anydata_val  Sys.AnyData;
  t            SYS.AnyType;
  data_pl      phone_list_typ;     -- varray
  data_ntt     inventory_list_typ; -- nested table type
  data_p       warehouse_typ;      -- object type
  ref1         REF inventory_typ;  -- REF type
  rval         PLS_INTEGER;        -- return value
  tc           PLS_INTEGER;        -- return value
  prec         PLS_INTEGER;        -- precision
  scale        PLS_INTEGER;        -- scale
  len          PLS_INTEGER;        -- length
  csid         PLS_INTEGER;        -- character set id
  csfrm        PLS_INTEGER;        -- character set form
  cnt          PLS_INTEGER;        -- count of varray elements or number of
                                   -- object attributes
  sname        VARCHAR2(35);       -- schema name
  type_name    VARCHAR2(35);       -- type name
  version      VARCHAR2(35);
BEGIN
   FOR i IN 1 .. 5 LOOP
     anydata_val := DBMS_DEFER_QUERY.GET_AnyData_ARG(call_no, i, txn_id);
     -- Get the type information, including type name.
     tc := anydata_val.GetType(t);
     tc := t.GetInfo(prec, scale, len, csid, csfrm, sname, type_name,
                     version, cnt);
     -- Based on the type name, convert the anydata value to the appropriate
     -- user-defined types.
     IF type_name = 'PHONE_LIST_TYP' THEN
       -- The anydata_val contains phone_list_typ varray instance.
```

```
      rval := anydata_val.GetCollection(data_pl);
      -- Do something with data_pl.
    ELSIF type_name = 'INVENTORY_LIST_TYP' THEN
      -- anydata_val contains inventory_list_typ nested table instance.
      rval := anydata_val.GetCollection(data_ntt);
      -- Do something with data_ntt.
    ELSIF type_name = 'WAREHOUSE_TYP' THEN
      -- The anydata_val contains warehouse_typ object instance.
      rval := anydata_val.GetObject(data_p);
      -- Do something with data_p.
    ELSIF type_name = 'INVENTORY_TYP' THEN
      -- The anydata_val contains a reference to inventory_typ object instance.
      rval := anydata_val.GetRef(ref1);
      -- Do something with ref1.
    END IF;
  END LOOP;
END;
/
```

See Also:

- "GET_datatype_ARG Function" on page 13-9
- *Oracle9i SQL Reference* and *Oracle9i Application Developer's Guide - Object-Relational Features* for more information about the `AnyData` datatype

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, then you should resolve the error condition and reexecute the deferred transaction. You have two options when reexecuting a deferred transaction: you can reexecute in the security context of the user who received the deferred transaction, or you can reexecute the deferred transaction with an alternate security context.

> **Caution:** If you have multiple error transactions and you want to make sure they are reexecuted in the correct order, then you can specify `NULL` for the `deferred_tran_id` parameter in the procedures in the following sections. If you do not specify `NULL`, then reexecuting individual transactions in the wrong order can cause conflicts.

Reexecuting Error Transaction as the Receiver

The following procedure reexecutes 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.

Meet the following requirements to complete these actions:

Executed As: Replication Administrator

Executed At: Site Containing Errors

Replication Status: Normal

Complete the following steps:

1. Connect to the master site as the replication administrator.

```
CONNECT repadmin/repadmin@orc2.world
```

2. Reexecute the error transaction.

```
BEGIN
   DBMS_DEFER_SYS.EXECUTE_ERROR (
      deferred_tran_id => '1.12.2904',
      destination => 'orc2.world');
END;
/
```

Reexecuting Error Transaction as Alternate User

The following procedure reexecutes 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.

Meet the following requirements to complete these actions:

Executed As: Connected User

Executed At: Site Containing Errors

Replication Status: Normal

Complete the following steps:

1. Connect to the master site as the alternate user.

```
CONNECT hr/hr@orc2.world
```

2. Reexecute the error transaction.

```
BEGIN
   DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER (
      deferred_tran_id => '1.12.2904',
      destination => 'orc2.world');
END;
/
```

10

Monitoring a Replication Environment

This chapter illustrates how to monitor a replication a replication environment using the data dictionary. This chapter contains these topics:

Note: The Replication Management tool in Oracle Enterprise Manager is also an excellent way to monitor a replication environment. Most of the information obtained by the queries in this chapter can be found in the reports available in the Replication Management tool. See the Replication Management tool online help for more information.

Monitoring Master Replication Environments

This section contains queries that you can run to display information about a master replication environment. The replication environment can be a multimaster environment, a master materialized view environment, or a hybrid environment that includes multiple master sites and materialized views.

Monitoring Master Sites

This section contains queries that you can run to display information about master sites.

Listing General Information About a Master Site

You can find the following general information about a master site by running the query in this section:

- The number of administrative requests
- The number of administrative request errors
- The number of unpropagated deferred transaction-destination pairs. Each deferred transaction may have multiple destinations to which it will be propagated, and each destination is a single deferred transaction-destination pair.

  For example, if there are ten deferred transactions and each one must be propagated to three sites, then there are 30 deferred transaction-pairs returned by this query. After some time, if the first deferred transaction is propagated to two of the three destination sites, then there are still ten deferred transactions, but there are two fewer deferred-transaction pairs, and this query returns 28 unpropagated deferred transaction-pairs. In this case, the first deferred transaction only has one transaction-pair remaining.
- The number of deferred transaction errors (error transactions)
- The number of successfully propagated transactions that are still in the queue. These transactions should be purged from the queue.

Run the following query to list this information for the current master site:

```
COLUMN GLOBAL_NAME HEADING 'Database' FORMAT A25
COLUMN ADMIN_REQUESTS HEADING 'Admin|Reqests' FORMAT 9999
COLUMN STATUS HEADING 'Admin|Errors' FORMAT 9999
COLUMN TRAN HEADING 'Def|Trans|Pairs' FORMAT 9999
COLUMN ERRORS HEADING 'Def|Trans|Errors' FORMAT 9999
COLUMN COMPLETE HEADING 'Propagated|Trans' FORMAT 9999

SELECT G.GLOBAL_NAME, D.ADMIN_REQUESTS, E.STATUS, DT.TRAN, DE.ERRORS, C.COMPLETE
    FROM (SELECT GLOBAL_NAME FROM GLOBAL_NAME) G,
      (SELECT COUNT(ID) ADMIN_REQUESTS FROM DBA_REPCATLOG) D,
     (SELECT COUNT(STATUS) STATUS FROM DBA_REPCATLOG WHERE STATUS = 'ERROR') E,
      (SELECT COUNT(*) TRAN FROM DEFTRANDEST) DT,
      (SELECT COUNT(*) ERRORS FROM DEFERROR) DE,
      (SELECT COUNT(A.DEFERRED_TRAN_ID) COMPLETE FROM DEFTRAN A
    WHERE A.DEFERRED_TRAN_ID NOT IN (
      SELECT B.DEFERRED_TRAN_ID FROM DEFTRANDEST B)) C;
```

Your output looks similar to the following:

```
                                           Def    Def
                          Admin  Admin Trans  Trans Propagated
Database                Reqests Errors Pairs Errors      Trans
------------------------- ------- ------ ----- ------ ----------
SF.WORLD                      5      0    37      0         53
```

Note: This query can be expensive if you have a large number of transactions in the deferred transactions queue.

Monitoring Master Groups

This section contains queries that you can run to display information about the master groups at a replication site.

Listing the Master Sites Participating in a Master Group

Run the following query to list the master sites for each master group at a replication site and indicate which master site is the master definition site for each master group:

```
COLUMN GNAME HEADING 'Master Group' FORMAT A20
COLUMN DBLINK HEADING 'Sites' FORMAT A25
COLUMN MASTERDEF HEADING 'Master|Definition|Site?' FORMAT A10
```

```
SELECT GNAME, DBLINK, MASTERDEF
    FROM DBA_REPSITES
    WHERE MASTER = 'Y'
    AND GNAME NOT IN (SELECT GNAME FROM DBA_REPSITES WHERE SNAPMASTER = 'Y')
    ORDER BY GNAME;
```

The subquery in the SELECT statement ensures that materialized view groups do not appear in the output. Your output looks similar to the following:

```
                                                    Master
                                                    Definition
Master_Group         Sites                          Site?
-------------------- ------------------------- ----------
HR_RG                SF.WORLD                       Y
HR_RG                NY.WORLD                       N
```

This list indicates that sf.world is the master definition site for the hr_rg master group.

Listing General Information About Master Groups

You can use the query in this section to list the following general information about the master groups at a master site:

- The name of each master group
- The number of unpropagated deferred transaction-destination pairs. Each deferred transaction may have multiple destinations to which it will be propagated, and each destination is a single deferred transaction-destination pair.

  For example, if there are ten deferred transactions and each one must be propagated to three sites, then there are 30 deferred transaction-pairs returned by this query. After some time, if the first deferred transaction is propagated to two of the three destination sites, then there are still ten deferred transactions, but there are two fewer deferred-transaction pairs, and this query returns 28 unpropagated deferred transaction-pairs. In this case, the first deferred transaction only has one transaction-pair remaining.
- The number of deferred transaction errors (error transactions) for each master group
- The number of administrative requests for each master group
- The number of administrative request errors for each master group

Run the following query to list this information:

```
COLUMN GNAME HEADING 'Master Group' FORMAT A15
COLUMN deftran HEADING 'Number of|Deferred|Transaction|Pairs' FORMAT 9999
COLUMN deftranerror HEADING 'Number of|Deferred|Transaction|Errors' FORMAT 9999
COLUMN adminreq HEADING 'Number of|Administrative|Requests' FORMAT 9999
COLUMN adminreqerror HEADING 'Number of|Administrative|Request|Errors'
COLUMN adminreqerror FORMAT 9999

SELECT G.GNAME,
       NVL(T.CNT1, 0) deftran,
       NVL(IE.CNT2, 0) deftranerror,
       NVL(A.CNT3, 0) adminreq,
       NVL(B.CNT4, 0) adminreqerror
    FROM
    (SELECT DISTINCT GNAME FROM DBA_REPGROUP WHERE MASTER='Y') G,
    (SELECT DISTINCT RO.GNAME, COUNT(DISTINCT D.DEFERRED_TRAN_ID) CNT1
        FROM  DBA_REPOBJECT RO, DEFCALL D, DEFTRANDEST TD
        WHERE RO.SNAME = D.SCHEMANAME
        AND RO.ONAME = D.PACKAGENAME
        AND RO.TYPE IN ('TABLE', 'PACKAGE', 'SNAPSHOT')
        AND TD.DEFERRED_TRAN_ID = D.DEFERRED_TRAN_ID
        GROUP BY RO.GNAME ) T,
    (SELECT DISTINCT RO.GNAME, COUNT(DISTINCT E.DEFERRED_TRAN_ID) CNT2
        FROM DBA_REPOBJECT RO, DEFCALL D, DEFERROR E
        WHERE RO.SNAME = D.SCHEMANAME
        AND RO.ONAME = D.PACKAGENAME
        AND RO.TYPE IN ('TABLE', 'PACKAGE', 'SNAPSHOT')
        AND E.DEFERRED_TRAN_ID = D.DEFERRED_TRAN_ID
        AND E.CALLNO = D.CALLNO
        GROUP BY RO.GNAME ) IE,
    (SELECT GNAME, COUNT(*) CNT3 FROM DBA_REPCATLOG GROUP BY GNAME) A,
    (SELECT GNAME, COUNT(*) CNT4 FROM DBA_REPCATLOG
        WHERE STATUS = 'ERROR'
        GROUP BY GNAME) B WHERE G.GNAME = IE.GNAME (+)
        AND G.GNAME = T.GNAME (+)
        AND G.GNAME = A.GNAME (+)
        AND G.GNAME = B.GNAME (+) ORDER BY G.GNAME;
```

Your output looks similar to the following:

```
                  Number of   Number of                        Number of
                   Deferred    Deferred      Number of Administrative
                Transaction Transaction Administrative        Request
Master Group          Pairs      Errors       Requests         Errors
--------------- ----------- ----------- -------------- --------------
HR_RG                    54           0              0              0
OE_RG                    33           1              5              0
```

Note: This query can be expensive if you have a large number of transactions waiting to be propagated.

Monitoring Masters

A master can be either a master site or a master materialized view site. This section contains queries that you can run to display information about masters.

Listing Information About Materialized Views Based on a Master

If you have materialized view sites based on a master, then you can use the query in this section to list the following information about the master:

- The number of replication groups at a master. The replication groups can be either master groups or materialized view groups.
- The number of registered materialized view groups based on the replication groups at the master
- The number of registered materialized views based on objects at the master. The objects can be either master tables or master materialized views.
- The number of materialized view logs at the master
- The number of deployment templates at the master

Run the following query to list this information:

```
COLUMN repgroup HEADING 'Number of|Replication|Groups' FORMAT 9999
COLUMN mvgroup HEADING 'Number of|Registered|MV Groups' FORMAT 9999
COLUMN mv HEADING 'Number of|Registered MVs' FORMAT 9999
COLUMN mvlog HEADING 'Number of|MV Logs' FORMAT 9999
COLUMN template HEADING 'Number of|Templates' FORMAT 9999
```

```
SELECT A.REPGROUP repgroup,
       B.MVGROUP mvgroup,
       C.MV mv,
       D.MVLOG mvlog,
       E.TEMPLATE template
    FROM (SELECT COUNT(G.GNAME) REPGROUP
             FROM DBA_REPGROUP G, DBA_REPSITES S
             WHERE G.MASTER = 'Y'
             AND S.MASTER = 'Y'
             AND G.GNAME = S.GNAME
             AND S.MY_DBLINK = 'Y') A,
         (SELECT COUNT(*) MVGROUP
             FROM DBA_REGISTERED_MVIEW_GROUPS) B,
         (SELECT COUNT(*) MV
             FROM DBA_REGISTERED_MVIEWS) C,
         (SELECT COUNT(*) MVLOG
             FROM (SELECT 1 FROM DBA_MVIEW_LOGS
             GROUP BY LOG_OWNER, LOG_TABLE)) D,
         (SELECT COUNT(*) TEMPLATE FROM DBA_REPCAT_REFRESH_TEMPLATES) E;
```

Your output looks similar to the following:

```
  Number of  Number of
Replication Registered      Number of Number of Number of
     Groups  MV Groups Registered MVs   MV Logs Templates
----------- ---------- -------------- --------- ---------
          1          5             27         6         3
```

Listing Information About the Materialized View Logs at a Master

A materialized view log enables you to fast refresh materialized views based on a master. A master can be a master table or a master materialized view. If you have materialized view logs based at a master, then you can use the query in this section to list the following information about them:

- The name of each log table that stores the materialized view log data
- The owner of each materialized view log
- The master on which each materialized view log is based
- Whether a materialized view log is a row id materialized view log
- Whether a materialized view log is a primary key materialized view log
- Whether the materialized view log is an object id materialized view log
- Whether a materialized view log has filter columns

Run the following query to list this information:

```
COLUMN LOG_TABLE HEADING 'Log Table' FORMAT A20
COLUMN LOG_OWNER HEADING 'Log|Owner' FORMAT A5
COLUMN MASTER HEADING 'Master' FORMAT A15
COLUMN ROWIDS HEADING 'Row|ID?' FORMAT A3
COLUMN PRIMARY_KEY HEADING 'Primary|Key?' FORMAT A7
COLUMN OBJECT_ID HEADING 'Object|ID?' FORMAT A6
COLUMN FILTER_COLUMNS HEADING 'Filter|Columns?' FORMAT A8

SELECT DISTINCT LOG_TABLE,
       LOG_OWNER,
       MASTER,
       ROWIDS,
       PRIMARY_KEY,
       OBJECT_ID,
       FILTER_COLUMNS
    FROM DBA_MVIEW_LOGS
    ORDER BY 1;
```

Your output looks similar to the following:

```
                     Log                      Row Primary Object Filter
Log Table            Owner Master             ID? Key?    ID?    Columns?
-------------------- ----- --------------- --- ------- ------ --------
MLOG$_COUNTRIES      HR    COUNTRIES       NO  YES     NO     NO
MLOG$_DEPARTMENTS    HR    DEPARTMENTS     NO  YES     NO     NO
MLOG$_EMPLOYEES      HR    EMPLOYEES       NO  YES     NO     NO
MLOG$_JOBS           HR    JOBS            NO  YES     NO     NO
MLOG$_JOB_HISTORY    HR    JOB_HISTORY     NO  YES     NO     NO
MLOG$_LOCATIONS      HR    LOCATIONS       NO  YES     NO     NO
```

See Also: *Oracle9i Replication* for information about materialized view logs

Listing the Materialized Views That Use a Materialized View Log

More than one materialized view can use a materialized view log. If you have materialized view logs based at a master, then you can use the query in this section to list the following the materialized views that use each log:

- The name of each log table that stores the materialized view log data
- The owner of each materialized view log
- The master on which each materialized view log is based
- The materialized view identification number of each materialized view that uses the materialized view log
- The name of each materialized view that uses the materialized view log

Run the following query to list this information:

```
COLUMN LOG_TABLE HEADING 'Mview|Log Table' FORMAT A20
COLUMN LOG_OWNER HEADING 'Mview|Log Owner' FORMAT A10
COLUMN MASTER HEADING 'Master' FORMAT A20
COLUMN MVIEW_ID HEADING 'Mview|ID' FORMAT 9999
COLUMN NAME HEADING 'Mview Name' FORMAT A20

SELECT L.LOG_TABLE, L.LOG_OWNER, B.MASTER, B.MVIEW_ID, R.NAME
FROM ALL_MVIEW_LOGS L, ALL_BASE_TABLE_MVIEWS B, ALL_REGISTERED_MVIEWS R
WHERE B.MVIEW_ID = R.MVIEW_ID
AND B.OWNER = L.LOG_OWNER
AND B.MASTER = L.MASTER;
```

Your output looks similar to the following:

```
Mview                                                     Mview
Log Table            Owner      Master                       ID Mview Name
-------------------- ---------- -------------------- ----- --------------------
MLOG$_COUNTRIES      HR         COUNTRIES                    24 COUNTRIES_MVIEW1
MLOG$_COUNTRIES      HR         COUNTRIES                    31 COUNTRIES_MVIEW2
MLOG$_DEPARTMENTS    HR         DEPARTMENTS                  19 DEPARTMENTS_MVIEW1
MLOG$_DEPARTMENTS    HR         DEPARTMENTS                  64 DEPARTMENTS_MVIEW2
MLOG$_DEPARTMENTS    HR         DEPARTMENTS                  15 DEPARTMENTS_MVIEW3
```

Listing Information About the Deployment Templates at a Master

Deployment templates enable you to create multiple materialized view environments quickly. They also enable you to use variables to customize each materialized view environment for its individual needs. You can use the query in this section to list the following information about the deployment templates at a master:

- The name of each deployment template
- The owner of each deployment template
- Whether a deployment template is public
- The number of instantiated materialized view sites based on each deployment template
- The comment associated with each deployment template

Run the following query to list this information:

```
COLUMN REFRESH_TEMPLATE_NAME HEADING 'Template|Name' FORMAT A10
COLUMN OWNER HEADING 'Owner' FORMAT A10
COLUMN PUBLIC_TEMPLATE HEADING 'Public?' FORMAT A7
COLUMN INSTANTIATED HEADING 'Number of|Instantiated|Sites' FORMAT 9999
COLUMN TEMPLATE_COMMENT HEADING 'Comment' FORMAT A35

SELECT DISTINCT RT.REFRESH_TEMPLATE_NAME,
       OWNER,
       PUBLIC_TEMPLATE,
       RS.INSTANTIATED,
       RT.TEMPLATE_COMMENT
    FROM DBA_REPCAT_REFRESH_TEMPLATES RT,
    (SELECT Y.REFRESH_TEMPLATE_NAME, COUNT(X.STATUS) INSTANTIATED
        FROM DBA_REPCAT_TEMPLATE_SITES X, DBA_REPCAT_REFRESH_TEMPLATES Y
        WHERE X.REFRESH_TEMPLATE_NAME(+) = Y.REFRESH_TEMPLATE_NAME
        GROUP BY Y.REFRESH_TEMPLATE_NAME) RS
        WHERE RT.REFRESH_TEMPLATE_NAME(+) = RS.REFRESH_TEMPLATE_NAME
        ORDER BY 1;
```

Your output looks similar to the following:

```
                                 Number of
Template                      Instantiated
Name       Owner      Public?        Sites Comment
---------- ---------- ------- ------------ -----------------------------------
HR_REFG_DT HR         N                  2 Human Resources Deployment Template
```

The N in the Public? column means that the deployment template is private. Therefore, it can only be instantiated by authorized users. A Y in this column means that the deployment template is public. Any user can instantiate a public deployment template.

Monitoring Materialized View Sites

This section contains queries that you can run to display information about the materialized view sites.

Listing General Information About a Materialized View Site

You can use the query in this section to list the following general information about the current materialized view site:

- The number of materialized view groups at the site
- The number of materialized views at the site
- The number of refresh groups at the site

Run the following query to list this information:

```
COLUMN MVGROUP HEADING 'Number of|Materialized|View Groups' FORMAT 9999
COLUMN MV HEADING 'Number of|Materialized|Views' FORMAT 9999
COLUMN RGROUP HEADING 'Number of|Refresh Groups' FORMAT 9999

SELECT A.MVGROUP, B.MV, C.RGROUP
    FROM
    (SELECT COUNT(S.GNAME) MVGROUP
        FROM DBA_REPSITES S
        WHERE S.SNAPMASTER = 'Y') A,
     (SELECT COUNT(*) MV
        FROM DBA_MVIEWS) B,
     (SELECT COUNT(*) RGROUP
        FROM DBA_REFRESH) C;
```

Your output looks similar to the following:

```
   Number of    Number of
Materialized Materialized      Number of
 View Groups        Views Refresh Groups
------------ ------------ --------------
           5           25              5
```

Listing General Information About Materialized View Groups

You can use the query in this section to list the following general information about the materialized view groups at the current materialized view site:

- The name of each materialized view group
- The master of each materialized view group
- The method of propagation to a materialized view group's master, either asynchronous or synchronous
- The comment associated with each materialized view group

Run the following query to list this information:

```
COLUMN GNAME HEADING 'Group Name' FORMAT A10
COLUMN DBLINK HEADING 'Master' FORMAT A25
COLUMN Propagation HEADING 'Propagation|Method' FORMAT A12
COLUMN SCHEMA_COMMENT HEADING 'Comment' FORMAT A30

SELECT S.GNAME,
       S.DBLINK,
       DECODE(S.PROP_UPDATES,
              0, 'ASYNCHRONOUS',
              1, 'SYNCHRONOUS') Propagation,
       G.SCHEMA_COMMENT
    FROM DBA_REPSITES S, DBA_REPGROUP G
    WHERE S.GNAME = G.GNAME
    AND S.SNAPMASTER = 'Y';
```

Your output looks similar to the following:

```
                                     Propagation
Group Name Master                    Method       Comment
---------- ------------------------- ------------ ------------------------------
HR_RG      SF.WORLD                  ASYNCHRONOUS Human Resources Group
```

Listing Information About Materialized Views

This section contains queries that you can run to display information about the materialized views at a replication site.

Listing Master Information For Materialized Views

The following query shows the master for each materialized view at a replication site and whether the materialized view can be fast refreshed:

```
COLUMN MVIEW_NAME HEADING 'Materialized|View Name' FORMAT A15
COLUMN OWNER HEADING 'Owner' FORMAT A10
COLUMN MASTER_LINK HEADING 'Master Link' FORMAT A30
COLUMN Fast_Refresh HEADING 'Fast|Refreshable?' FORMAT A16

SELECT MVIEW_NAME,
       OWNER,
       MASTER_LINK,
       DECODE(FAST_REFRESHABLE,
              'NO', 'NO',
              'DML', 'YES',
              'DIRLOAD', 'DIRECT LOAD ONLY',
              'DIRLOAD_DML', 'YES',
              'DIRLOAD_LIMITEDDML', 'LIMITED') Fast_Refresh
    FROM DBA_MVIEWS;
```

Your output looks similar to the following:

```
Materialized                                                Fast
View Name       Owner      Master Link                      Refreshable?
--------------- ---------- ------------------------------ ----------------
DEPARTMENTS_MV  HR         @SF.WORLD                        YES
EMPLOYEES_MV    HR         @SF.WORLD                        YES
JOBS_MV         HR         @SF.WORLD                        YES
JOB_HISTORY_MV  HR         @SF.WORLD                        YES
LOCATIONS_MV    HR         @SF.WORLD                        YES
```

Listing the Properties of Materialized Views

You can use the query in this section to list the following information about the materialized views at the current replication site:

- The name of each materialized view
- The owner of each materialized view
- The refresh method used by each materialized view: COMPLETE, FORCE, FAST, or NEVER
- Whether a materialized view is updatable
- The last date on which each materialized view was refreshed

Run the following query to list this information:

```
COLUMN MVIEW_NAME HEADING 'Materialized|View Name' FORMAT A15
COLUMN OWNER HEADING 'Owner' FORMAT A10
COLUMN REFRESH_METHOD HEADING 'Refresh|Method' FORMAT A10
COLUMN UPDATABLE HEADING 'Updatable?' FORMAT A10
COLUMN LAST_REFRESH_DATE HEADING 'Last|Refresh|Date'
COLUMN LAST_REFRESH_TYPE HEADING 'Last|Refresh|Type' FORMAT A15

SELECT MVIEW_NAME,
       OWNER,
       REFRESH_METHOD,
       UPDATABLE,
       LAST_REFRESH_DATE,
       LAST_REFRESH_TYPE
    FROM DBA_MVIEWS;
```

Your output looks similar to the following:

```
                                                         Last            Last
Materialized                Refresh                      Refresh         Refresh
View Name       Owner       Method     Updatable? Date            Type
--------------- ---------- ---------- ---------- --------------- ---------------
DEPARTMENTS_MV  HR         FORCE      Y          22-JAN-01       FAST
EMPLOYEES_MV    HR         FAST       Y          22-JAN-01       COMPLETE
JOBS_MV         HR         COMPLETE   Y          22-JAN-01       COMPLETE
JOB_HISTORY_MV  HR         FAST       Y          22-JAN-01       FAST
LOCATIONS_MV    HR         FAST       Y          22-JAN-01       FAST
```

Listing Information About the Refresh Groups at a Materialized View Site

Each refresh group at a materialized view site is associated with a refresh job that refreshes the materialized views in the refresh group at a set interval. You can query the DBA_REFRESH data dictionary view to list the following information about the refresh jobs at a materialized view site:

- The name of the refresh group
- The owner of the refresh group
- Whether the refresh job is broken
- The next date and time when the refresh job will run
- The current interval setting for the refresh job. The interval setting specifies the amount of time between the start of a job and the next start of the same job.

The following query displays this information:

```
COLUMN RNAME HEADING 'Refresh|Group|Name' FORMAT A10
COLUMN ROWNER HEADING 'Refresh|Group|Owner' FORMAT A10
COLUMN BROKEN HEADING 'Broken?' FORMAT A7
COLUMN next_refresh HEADING 'Next Refresh'
COLUMN INTERVAL HEADING 'Interval' FORMAT A20

SELECT RNAME,
       ROWNER,
       BROKEN,
       TO_CHAR(NEXT_DATE, 'DD-MON-YYYY HH:MI:SS AM') next_refresh,
       INTERVAL
    FROM DBA_REFRESH
    ORDER BY 1;
```

Your output looks similar to the following:

```
Refresh    Refresh
Group      Group
Name       Owner      Broken? Next Refresh           Interval
---------- ---------- ------- ---------------------- --------------------
HR_REFG    MVIEWADMIN N       01-JAN-4000 12:00:00 AM SYSDATE + 1/24
```

The N in the Broken? column means that the job is not broken. Therefore, the refresh job will run at the next start time. A Y in this column means that the job is broken.

Determining the Job ID for Each Refresh Job at a Materialized View Site

You can use the query in this section to list the following information about the refresh jobs at a materialized view site:

- The job identification number of each refresh job. Each job created by the `DBMS_JOBS` package is assigned a unique identification number.
- The privilege schema, which is the schema whose default privileges apply to the job
- The schema that owns each refresh job. Typically, the materialized view administrator owns a refresh job. A common username for the materialized view administrator is `mviewadmin`.
- The name of the refresh group that the job refreshes
- The status of the refresh job, either normal or broken

The following query displays this information:

```
COLUMN JOB HEADING 'Job ID' FORMAT 999999
COLUMN PRIV_USER HEADING 'Privilege|Schema' FORMAT A10
COLUMN RNAME HEADING 'Refresh|Group|Name' FORMAT A10
COLUMN ROWNER HEADING 'Refresh|Group|Owner' FORMAT A10
COLUMN BROKEN HEADING 'Broken?' FORMAT A7

SELECT J.JOB,
       J.PRIV_USER,
       R.ROWNER,
       R.RNAME,
       J.BROKEN
    FROM DBA_REFRESH R, DBA_JOBS J
    WHERE R.JOB = J.JOB
    ORDER BY 1;
```

Your output looks similar to the following:

```
                   Refresh    Refresh
        Privilege  Group      Group
 Job ID Schema     Owner      Name       Broken?
------- ---------- ---------- ---------- -------
     21 MVIEWADMIN MVIEWADMIN HR_REFG    N
```

The `N` in the `Broken?` column means that the job is not broken. Therefore, the job will run at the next start time. A `Y` in this column means that the job is broken.

Determining Which Materialized Views Are Currently Refreshing

The following query shows the materialized views that are currently refreshing:

```
COLUMN SID HEADING 'Session|Identifier' FORMAT 9999
COLUMN SERIAL# HEADING 'Serial|Number' FORMAT 999999
COLUMN CURRMVOWNER HEADING 'Owner' FORMAT A15
COLUMN CURRMVNAME HEADING 'Materialized|View' FORMAT A25

SELECT * FROM V$MVREFRESH;
```

Your output looks similar to the following:

```
   Session  Serial                 Materialized
Identifier  Number Owner           View
---------- ------- --------------- -------------------------
        19     233 HR              COUNTRIES_MV
         5     647 HR              EMPLOYEES_MV
```

Note: The V$MVREFRESH dynamic performance view does not contain information about updatable materialized views when the materialized views' deferred transactions are being pushed to its master.

Monitoring Administrative Requests

This section contains queries that you can run to display information about the administrative requests at a master site.

Listing General Information About Administrative Requests

You can use the query in this section to list the following general information about the administrative requests at a master site:

- The identification number of each administrative request
- The action requested by each administrative request
- The status of each request
- The master site where the request is being executed

The following query displays this information:

```
COLUMN ID HEADING 'Admin|Request|ID' FORMAT 999999
COLUMN REQUEST HEADING 'Request' FORMAT A25
COLUMN STATUS HEADING 'Status' FORMAT A15
COLUMN MASTER HEADING 'Master|Site' FORMAT A25

SELECT ID, REQUEST, STATUS, MASTER FROM DBA_REPCATLOG;
```

Your output looks similar to the following:

```
  Admin
Request                                           Master
     ID Request                   Status          Site
------- ------------------------- --------------- -------------------------
     44 RESUME_MASTER_ACTIVITY    AWAIT_CALLBACK  NY.WORLD
```

Determining the Cause of Administrative Request Errors

You can determine the cause of an administrative request error by displaying its error message. The following query displays the error message for each administrative request that resulted in an error:

```
COLUMN ID HEADING 'Admin|Request|ID' FORMAT 999999
COLUMN REQUEST HEADING 'Request' FORMAT A30
COLUMN ERRNUM HEADING 'Error|Number' FORMAT 999999
COLUMN MESSAGE HEADING 'Error|Message' FORMAT A32

SELECT ID, REQUEST, ERRNUM, MESSAGE
    FROM DBA_REPCATLOG WHERE STATUS = 'ERROR';
```

Your output looks similar to the following:

```
  Admin
Request                                  Error Error
     ID Request                         Number Message
------- ------------------------------ ------- --------------------------------
     70 CREATE_MASTER_REPOBJECT          -2292 ORA-02292: integrity constrain
                                               t (HR.DEPT_LOC_FK) violated -
                                               child record found
                                               ORA-02266: unique/primary keys
                                                in table referenced by enable
                                               d foreign keys

     71 GENERATE_INTERNAL_PKG_SUPPORT   -23308 ORA-23308: object HR.LOCATIONS
                                                does not exist or is invalid
```

Listing General Information About the Job that Executes Administrative Requests

Each master group is associated with a `do_deferred_repcat_admin` job that executes administrative requests. You can query the `DBA_JOBS` data dictionary view to list the following information about this job at a replication site:

- The job identification number of each `do_deferred_repcat_admin` job. Each job created by the `DBMS_JOBS` package is assigned a unique identification number.
- The privilege schema, which is the schema whose default privileges apply to the job
- The status of each `do_deferred_repcat_admin` job, either normal or broken
- The next date and time when each `do_deferred_repcat_admin` job will run
- The current interval setting for each `do_deferred_repcat_admin` job. The interval setting specifies the amount of time between the start of a job and the next start of the same job.

The following query displays this information:

```
COLUMN JOB HEADING 'Job ID' FORMAT 999999
COLUMN PRIV_USER HEADING 'Privilege|Schema' FORMAT A10
COLUMN BROKEN HEADING 'Broken?' FORMAT A7
COLUMN next_start HEADING 'Next Start'
COLUMN INTERVAL HEADING 'Interval' FORMAT A20

SELECT JOB,
       PRIV_USER,
       BROKEN,
       TO_CHAR(NEXT_DATE,'DD-MON-YYYY HH:MI:SS AM') next_start,
       INTERVAL
    FROM DBA_JOBS
    WHERE WHAT LIKE '%dbms_repcat.do_deferred_repcat_admin%'
    ORDER BY 1;
```

Your output looks similar to the following:

```
        Privilege
 Job ID Schema     Broken? Next Start              Interval
------- ---------- ------- ----------------------- --------------------
      3 REPADMIN   N       02-FEB-2001 04:34:36 PM SYSDATE + (1/144)
```

The `N` in the `Broken?` column means that the job is not broken. Therefore, the job will run at the next start time. A `Y` in this column means that the job is broken.

Checking the Definition of Each do_deferred_repcat_admin Job

You can query the DBA_JOBS data dictionary view to show the definition of each do_deferred_repcat_admin job at a replication site. The following query shows the definitions:

```
COLUMN JOB HEADING 'Job ID' FORMAT 999999
COLUMN WHAT HEADING 'Definitions of Admin Req Jobs' FORMAT A70

SELECT JOB, WHAT
  FROM DBA_JOBS
  WHERE WHAT LIKE '%dbms_repcat.do_deferred_repcat_admin%'
  ORDER BY 1;
```

Your output looks similar to the following:

```
 Job ID Definitions of Admin Req Jobs
------- ----------------------------------------------------------------------
    321 dbms_repcat.do_deferred_repcat_admin('"HR_RG"', FALSE);
    342 dbms_repcat.do_deferred_repcat_admin('"OE_RG"', FALSE);
```

Monitoring the Deferred Transactions Queue

This section contains queries that you can run to display information about the deferred transactions queue at a replication site.

Monitoring Transaction Propagation

This section contains queries that you can run to display information about propagation of transactions in the deferred transactions queue.

Listing the Number of Deferred Transactions for Each Destination Master Site

You can find the number of unpropagated deferred transactions for each destination master site by running the query in this section. This query shows each master site to which the current master site is propagating deferred transactions and the number of deferred transactions to be propagated to each destination site.

Run the following query to see the number of deferred and error transactions:

```
COLUMN DEST HEADING 'Destination' FORMAT A45
COLUMN TRANS HEADING 'Def Trans' FORMAT 9999
```

```
SELECT DBLINK DEST, COUNT(*) TRANS
    FROM DEFTRANDEST D
    GROUP BY DBLINK;
```

Your output looks similar to the following:

```
Destination                                     Def Trans
---------------------------------------------  ---------
NY.WORLD                                               27
SF.WORLD                                               44
```

Note: This query can be expensive if you have a large number of transactions waiting to be propagated.

Listing General Information About the Push Jobs at a Replication Site

Each scheduled link at a replication site is associated with a push job that propagates deferred transactions in the deferred transaction queue to a destination site. You can use the query in this section to list the following information about the push jobs at a replication site:

- The job identification number of each push job. Each job created by the DBMS_JOBS package is assigned a unique identification number.
- The privilege schema, which is the schema whose default privileges apply to the job
- The destination site where the deferred transactions are pushed
- The status of the push job, either normal or broken

The following query displays this information:

```
COLUMN JOB HEADING 'Job ID' FORMAT 999999
COLUMN PRIV_USER HEADING 'Privilege|Schema' FORMAT A10
COLUMN DBLINK HEADING 'Destination' FORMAT A40
COLUMN BROKEN HEADING 'Broken?' FORMAT A7

SELECT J.JOB,
       J.PRIV_USER,
       S.DBLINK,
       J.BROKEN
    FROM DEFSCHEDULE S, DBA_JOBS J
    WHERE S.DBLINK != (SELECT GLOBAL_NAME FROM GLOBAL_NAME)
    AND S.JOB = J.JOB
    ORDER BY 1;
```

Your output looks similar to the following:

```
        Privilege
 Job ID Schema     Destination                              Broken?
------- ---------- ---------------------------------------- -------
      2 REPADMIN   NY.WORLD                                 N
```

The N in the Broken? column means that the job is not broken. Therefore, the job will run at the next start time. A Y in this column means that the job is broken.

Determining the Next Start Time and Interval for the Push Jobs

Each scheduled link at a replication site is associated with a push job that propagates deferred transactions in the deferred transaction queue to a destination site. You can query the DEFSCHEDULE and DBA_JOBS data dictionary views to list the following information about the push jobs at a replication site:

- The job identification number of each push job. Each job created by the DBMS_JOBS package is assigned a unique identification number.
- The destination site where the deferred transactions are pushed
- The next date and time when the push job will run
- The current interval setting for the push job. The interval setting specifies the amount of time between the start of a job and the next start of the same job.

The following query displays this information:

```
COLUMN JOB HEADING 'Job ID' FORMAT 999999
COLUMN DBLINK HEADING 'Destination' FORMAT A22
COLUMN next_start HEADING 'Next Start'
COLUMN INTERVAL HEADING 'Interval' FORMAT A25

SELECT JOB,
       DBLINK,
       TO_CHAR(NEXT_DATE, 'DD-MON-YYYY HH:MI:SS AM') next_start,
       INTERVAL
    FROM DEFSCHEDULE
    WHERE DBLINK != (SELECT GLOBAL_NAME FROM GLOBAL_NAME)
    AND JOB IS NOT NULL
    ORDER BY 1;
```

Your output looks similar to the following:

```
 Job ID Destination            Next Start              Interval
------- ---------------------- ----------------------- -------------------------
      2 NY.WORLD               02-FEB-2001 04:44:39 PM SYSDATE + 10 / (24 * 60)
```

Determining the Total Number of Transactions Queued for Propagation

Run the following query to display the total number of transactions in the deferred transaction queue that are waiting to be propagated:

```
SELECT COUNT(DISTINCT DEFERRED_TRAN_ID) "Transactions Queued"
    FROM DEFTRANDEST;
```

Your output looks similar to the following:

```
Transactions Queued
-------------------
                 37
```

Note: This query can be expensive if you have a large number of transactions waiting to be propagated.

Monitoring Purges of Successfully Propagated Transactions

This section contains queries that you can run to display information about purges of successfully propagated transactions from the deferred transactions queue.

Listing General Information About the Purge Job

During standard setup of a replication site, you configure a purge job to remove successfully propagated transactions from the deferred transactions queue. You can query the `DBA_JOBS` data dictionary view to list the following information about the purge job at a replication site:

- The job identification number of the purge job. Each job created by the `DBMS_JOBS` package is assigned a unique identification number.
- The privilege schema, which is the schema whose default privileges apply to the job
- The status of the job, either normal or broken

- The next date and time when the purge job will run
- The current interval setting for the purge job. The interval setting specifies the amount of time between the start of a job and the next start of the same job.

The following query displays this information:

```
COLUMN JOB HEADING 'Job ID' FORMAT 999999
COLUMN PRIV_USER HEADING 'Privilege|Schema' FORMAT A10
COLUMN BROKEN HEADING 'Broken?' FORMAT A7
COLUMN next_start HEADING 'Next Start'
COLUMN INTERVAL HEADING 'Interval' FORMAT A25

SELECT JOB,
       PRIV_USER,
       BROKEN,
       TO_CHAR(NEXT_DATE, 'DD-MON-YYYY HH:MI:SS AM') next_start,
       INTERVAL
    FROM DBA_JOBS
    WHERE WHAT LIKE '%dbms_defer_sys.purge%'
    ORDER BY 1;
```

Your output looks similar to the following:

```
        Privilege
 Job ID Schema     Broken? Next Start              Interval
------- ---------- ------- ----------------------- -------------------------
      1 REPADMIN   N       02-FEB-2001 05:06:43 PM SYSDATE + 1/24
```

The `N` in the `Broken?` column means that the job is not broken. Therefore, the job will run at the next start time. A `Y` in this column means that the job is broken.

Checking the Definition of the Purge Job

You can query the `DBA_JOBS` data dictionary view to show the definition of the purge job at a replication site. The following query shows the definition:

```
SELECT WHAT "Definition of the Purge Job"
    FROM DBA_JOBS
    WHERE WHAT LIKE '%dbms_defer_sys.purge%' ORDER BY 1;
```

Your output looks similar to the following:

```
Definition of the Purge Job
--------------------------------------------------------------------------------
declare rc binary_integer; begin rc := sys.dbms_defer_sys.purge( delay_seconds=>
0); end;
```

Determining the Amount of Time Since the Last Purge

The following query shows the total amount of time, in minutes, since the successfully propagated transactions were purged from the deferred transactions queue:

```
SELECT ((SYSDATE - LAST_PURGE_TIME) / 60) "Minutes Since Last Purge"
    FROM V$REPLQUEUE;
```

Your output looks similar to the following:

```
Minutes Since Last Purge
------------------------
              13.43333
```

Determining the Total Number of Purged Transactions

The following query shows the total number of successfully propagated transactions that have been purged from the deferred transaction queue since the instance was last started:

```
SELECT TXNS_PURGED "Transactions Purged"
    FROM V$REPLQUEUE;
```

Your output looks similar to the following:

```
Transactions Purged
-------------------
               6541
```

Monitoring the Error Queue

This section contains queries that you can run to display information about the error queue at a replication site. The error queue contains deferred transactions that resulted in an error at the destination site. These error transactions are placed in the error queue at the destination site.

Listing General Information About the Error Transactions at a Replication Site

The following query lists the general information about the error transactions at a replication site:

```
COLUMN DEFERRED_TRAN_ID HEADING 'Deferred|Transaction|ID' FORMAT A11
COLUMN ORIGIN_TRAN_DB HEADING 'Origin|Database' FORMAT A15
COLUMN DESTINATION HEADING 'Destination|Database' FORMAT A15
COLUMN TIME_OF_ERROR HEADING 'Time of|Error' FORMAT A22
COLUMN ERROR_NUMBER HEADING 'Oracle|Error|Number' FORMAT 999999

SELECT DEFERRED_TRAN_ID,
       ORIGIN_TRAN_DB,
       DESTINATION,
       TO_CHAR(START_TIME, 'DD-Mon-YYYY hh24:mi:ss') TIME_OF_ERROR,
       ERROR_NUMBER
    FROM DEFERROR ORDER BY START_TIME;
```

Your output looks similar to the following:

```
Deferred                                                           Oracle
Transaction Origin          Destination     Time of                 Error
ID          Database        Database        Error                  Number
----------- --------------- --------------- ---------------------- -------
1.8.2470    SF.WORLD        NY.WORLD        25-Jan-2001 17:11:17      1403
```

You can use the deferred transaction ID and the destination database to either attempt to rerun the transaction that caused the error or to delete the error.

For example, to attempt to rerun the transaction in the previous example, enter the following:

```
EXECUTE DBMS_DEFER_SYS.EXECUTE_ERROR('1.8.2470', 'NY.WORLD');
```

To delete the error in the previous example, enter the following:

```
EXECUTE DBMS_DEFER_SYS.DELETE_ERROR('1.8.2470', 'NY.WORLD');
```

Typically, you should delete an error only if you have resolved it manually.

Determining the Percentage of Error Transactions

When propagating transactions to a remote master site, some transactions are propagated and applied successfully while other transactions may result in errors at the remote master site. Transactions that result in errors are called error transactions.

Run the following query to display the percentage of error transactions that resulted from propagation to the remote master site `SF.WORLD`:

```
SELECT DECODE(TOTAL_TXN_COUNT, 0, 'No Transactions',
       (TOTAL_ERROR_COUNT/TOTAL_TXN_COUNT)*100) "ERROR PERCENTAGE"
  FROM DEFSCHEDULE
  WHERE DBLINK = 'SF.WORLD';
```

Your output looks similar to the following:

```
Error Percentage
----------------
           3.265
```

Note: If this query returns `'No transactions'`, then no transactions have been propagated to the specified remote site since the statistics were last cleared.

Listing the Number of Error Transactions from Each Origin Master Site

You can find the number of transaction errors resulting from pushes by each origin master site by running the query in this section.

Run the following query to see the number of deferred and error transactions:

```
COLUMN SOURCE HEADING 'Origin' FORMAT A45
COLUMN ERRORS HEADING 'Def Trans Errors' FORMAT 9999

SELECT E.ORIGIN_TRAN_DB SOURCE, COUNT(*) ERRORS
    FROM DEFERROR E
    GROUP BY E.ORIGIN_TRAN_DB;
```

Your output looks similar to the following:

```
Origin                                        Def Trans Errors
--------------------------------------------- ----------------
NY.WORLD                                                     1
SF.WORLD                                                     3
```

Listing the Error Messages for the Error Transactions at a Replication Site

The following query lists the error messages for the error transactions at a replication site:

```
COLUMN DEFERRED_TRAN_ID HEADING 'Deferred|Transaction|ID' FORMAT A11
COLUMN ERROR_MSG HEADING 'Error Messages' FORMAT A68

SELECT DEFERRED_TRAN_ID, ERROR_MSG
    FROM DEFERROR;
```

Your output looks similar to the following:

```
Deferred
Transaction
ID          Error Messages
----------- --------------------------------------------------------------------
1.8.2470    ORA-01403: no data found
```

Determining the Error Operations at a Replication Site

The following query lists the type of operation that was attempted for each call that caused an error at a replication site:

```
COLUMN CALLNO HEADING 'Call|Number' FORMAT 9999
COLUMN DEFERRED_TRAN_ID HEADING 'Deferred|Transaction|ID' FORMAT A11
COLUMN PACKAGENAME HEADING 'Package|Name' FORMAT A20
COLUMN PROCNAME HEADING 'Operation' FORMAT A15
COLUMN ORIGIN_TRAN_DB HEADING 'Origin|Database' FORMAT A15

SELECT /*+ ORDERED */
       C.CALLNO,
       C.DEFERRED_TRAN_ID,
       C.PACKAGENAME,
       C.PROCNAME, E.ORIGIN_TRAN_DB
    FROM DEFERROR E, DEFCALL C
    WHERE C.DEFERRED_TRAN_ID = E.DEFERRED_TRAN_ID
    AND C.CALLNO = E.CALLNO
    ORDER BY E.START_TIME;
```

Your output looks similar to the following:

```
       Deferred
  Call Transaction Package                                  Origin
Number ID          Name                Operation       Database
------ ----------- -------------------- -------------- ---------------
     0 1.8.2470    EMPLOYEES$RP         REP_UPDATE      SF.WORLD
```

Monitoring Initialization Parameters

Certain initialization parameters are important in a replication environment. The following query lists each of these initialization parameters, its current value, and whether the initialization parameter is using its default value:

```
COLUMN NAME HEADING 'Parameter_Name' FORMAT A31
COLUMN VALUE HEADING 'Value' FORMAT A30
COLUMN ISDEFAULT HEADING 'Is Default?' FORMAT A11

SELECT NAME, VALUE, ISDEFAULT
    FROM V$PARAMETER
    WHERE NAME IN (
    'compatible',
    'db_encrypt_login',
    'distributed_transactions',
    'global_names',
    'job_queue_processes',
    'open_links',
    'open_links_per_instance',
    'parallel_automatic_tuning',
    'parallel_max_servers',
    'parallel_min_servers',
    'processes',
    'replication_dependency_tracking',
    'shared_pool_size',
    'utl_file_dir')
    ORDER BY NAME;
```

Your output looks similar to the following:

```
Parameter_Name                  Value                          Is Default?
------------------------------- ------------------------------ -----------
compatible                      9.0.0                          FALSE
distributed_transactions        10                             FALSE
global_names                    TRUE                           FALSE
job_queue_processes             20                             FALSE
```

```
open_links                      4                              TRUE
open_links_per_instance         4                              TRUE
parallel_automatic_tuning       FALSE                          TRUE
parallel_max_servers            50                             FALSE
parallel_min_servers            5                              FALSE
processes                       30                             FALSE
replication_dependency_tracking TRUE                           FALSE
shared_pool_size                75497472                       FALSE
utl_file_dir                    /home/dboutput/                FALSE
```

`FALSE` in the `Is Default?` column means the initialization parameter is not set to its default value. `TRUE` in this column means that the initialization parameter is set to its default value.

> **See Also:** *Oracle9i Replication* for more information about these initialization parameters

Monitoring Performance in a Replication Environment

This section contains queries that you can run to monitor the performance of your replication environment.

Tracking the Average Number of Row Changes in a Replication Transaction

The following query shows the average number of row changes in a replication transaction since instance startup:

```
SELECT DECODE(TXNS_ENQUEUED, 0, 'No Transactions Enqueued',
       (CALLS_ENQUEUED / TXNS_ENQUEUED)) "Average Number of Row Changes"
  FROM V$REPLQUEUE;
```

Your output looks similar to the following:

```
Average Number of Row Changes
-----------------------------
                        56.16
```

> **Note:** If this query returns `'No Transactions Enqueued'`, then no transactions have been enqueued since the start of the instance.

Tracking the Rate of Transactions Entering the Deferred Transactions Queue

The following query shows the average number of transactions per second entering at the deferred transactions queue at the current site since instance startup

```
SELECT (R.TXNS_ENQUEUED / ((SYSDATE - I.STARTUP_TIME)*24*60*60)) "Average TPS"
    FROM V$REPLQUEUE R, V$INSTANCE I;
```

Your output looks similar to the following:

```
Average TPS
-----------
        150
```

Determining the Average Network Traffic Created To Propagate a Transaction

Propagation of deferred transactions creates a certain amount of traffic on your network. Here, the network traffic created by a transaction is the number of bytes being sent and received and the number of network round trips needed to propagate the transaction.

A round trip is one or more consecutively sent messages followed by one or more consecutively received messages. For example, both of the following scenarios constitute only one round trip:

- Site A sends one message to site B and then site B sends one message to site A.
- Site A sends 20 messages to site B and then site B sends one message to site A.

These scenarios illustrate that the number of messages is irrelevant when evaluating the number of round trips, because the number of round trips is the number of back and forth communications between sites.

The following query shows the average network traffic created when propagating a transaction to the `SF.WORLD` remote master site:

```
SELECT
 DECODE(TOTAL_TXN_COUNT, 0, 'No Transactions',
 ((TOTAL_BYTES_SENT + TOTAL_BYTES_RECEIVED) / TOTAL_TXN_COUNT)) "Average Bytes",
 DECODE(TOTAL_TXN_COUNT, 0, 'No Transactions',
 (TOTAL_ROUND_TRIPS / TOTAL_TXN_COUNT)) "Average Round Trips"
 FROM DEFSCHEDULE WHERE DBLINK = 'SF.WORLD';
```

Your output looks similar to the following:

```
Average Bytes Average Round Trips
------------- -------------------
      69621.5                   5
```

> **Note:** If this query returns 'No transactions' in both columns, then no transactions have been propagated to the specified remote site since the statistics were last cleared.

Determining the Average Amount of Time to Apply Transactions at Remote Sites

Average latency is the average number of seconds between the first call of a transaction on the current site and the confirmation that the transaction was applied at the remote site. The first call begins when the user makes the first data manipulation language (DML) change, not when the transaction is committed.

The following query shows the average latency for applying transactions at the remote master site SF.WORLD:

```
SELECT AVG_LATENCY "Average Latency"
    FROM DEFSCHEDULE
    WHERE DBLINK='SF.WORLD';
```

Your output looks similar to the following:

```
Average Latency
---------------
           25.5
```

Determining the Percentage of Time the Parallel Propagation Job Spends Sleeping

When the parallel propagation coordinator is inactive, it is sleeping. You control the amount of time that the propagation coordinator sleeps using the delay_seconds parameter in the DBMS_DEFER_SYS.PUSH procedure.

The following query shows the percentage of time that the parallel propagation coordinator spends sleeping when propagating transactions to the SF.WORLD remote master site:

```
SELECT DECODE(AVG_THROUGHPUT, 0, NULL,
       ((TOTAL_SLEEP_TIME / (TOTAL_TXN_COUNT / AVG_THROUGHPUT)) * 100))
       "Percent Sleep Time"
  FROM DEFSCHEDULE WHERE DBLINK = 'SF.WORLD';
```

Your output looks similar to the following:

```
Percent Sleep Time
------------------
                 2
```

Of course, in this case, the parallel propagation coordinator is active 98% of the time.

> **Note:** If this query returns a NULL, then no transactions have been propagated to the specified remote site since the statistics were last cleared or since the last database startup.

Clearing the Statistics for a Remote Master Site in the DEFSCHEDULE View

To clear the propagation statistics in the DEFSCHEDULE view for a particular remote master site, use the CLEAR_PROP_STATISTICS procedure in the DBMS_DEFER_SYS package. For example, to clear the propagation statistics for the SF.WORLD remote master site, run the following procedure:

```
BEGIN
  DBMS_DEFER_SYS.CLEAR_PROP_STATISTICS (
   dblink => 'SF.WORLD');
END;
/
```

Monitoring Parallel Propagation of Deferred Transactions Using V$REPLPROP

The V$REPLPROP dynamic performance view provides information about current parallel propagation sessions.

> **Note:** The V$REPLPROP dynamic performance view is only relevant if you are using parallel propagation of deferred transactions. If you are using serial propagation, then this view is empty.

Determining the Databases to Which You Are Propagating Deferred Transactions

Run the following query to list the database link of each database to which you are currently propagating deferred transactions using parallel propagation:

```
SELECT DBLINK "Database Link"
    FROM V$REPLPROP
    WHERE NAME LIKE '%Coordinator%';
```

Your output looks similar to the following:

```
Database Link
-----------------
SF.WORLD
NY.WORLD
HK.WORLD
```

Determining the Transactions Currently Being Propagated to a Remote Master

You can list the following information about the transactions that are currently being propagated to a specified remote master site using parallel propagation:

- The transaction identification number of each transaction
- The number of calls in each transaction
- The percentage of processed calls in each transaction. The number in this column becomes larger as the calls in the transaction are processed. When the number reaches 100, all of the calls are processed.

The following query displays this information:

```
SELECT /*+ ORDERED */ P.XID "Tran Being Propagated",
       (MAX(C.CALLNO) + 1) "Number of Calls in Tran",
       (P.SEQUENCE/MAX(C.CALLNO) + 1) * 100 "% Processed Calls"
   FROM V$REPLPROP P, DEFCALL C
   WHERE P.NAME LIKE '%SLAVE%'
   AND P.DBLINK = 'SF.WORLD'
   AND C.DEFERRED_TRAN_ID = P.XID
   GROUP BY P.XID, P.SEQUENCE;
```

Your output looks similar to the following:

```
Tran Being Propagated  Number of Calls in Tran % Processed Calls
---------------------- ----------------------- -----------------
1.11.4264                                43357                78
1.15.4256                                23554                49
```

The transaction identification numbers should change as existing transactions are pushed and new transactions are processed. This query can be particularly useful if the any of the following conditions apply to your replication environment:

- You push a large number of transactions on a regular basis.
- You have some transactions that are very large.
- You are simulating continuous push using asynchronous propagation.

If the first two bullets apply to your replication environment, then you can run this query to check if the slave processes are pushing the transactions. In this type of environment, the slave processes do not exist when they are not pushing transactions.

In replication environments that are simulating continuous push, the slave processes exist whenever there are transactions to push in the deferred transactions queue. When there are no transactions to push, the slave processes may not exist. So, when there are transactions to push, you can use this query to make sure the slave processes exist and are processing the transactions.

> **See Also:** *Oracle9i Replication* for more information about scheduling continuous push in your replication environment

Part III

Replication Management API Packages Reference

Part III includes reference information about the 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 the chapters in this part 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.

PL/SQL Packages

Oracle's replication management API includes the following PL/SQL packages:

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

11

Introduction to the 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. The Replication Management tool in Oracle Enterprise Manager also uses the procedures and functions of each site's replication management API to perform work.

This chapter contains the following topics:

Note: Some of the PL/SQL procedures and functions described in the chapters in this part 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.

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 SQL*Plus or Enterprise Manager SQL Worksheet. For example, the following call to the DBMS_REPCAT.CREATE_MASTER_REPOBJECT procedure creates a new replicated table hr.employees in the hr_repg replication group:

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname => 'hr_repg',
      type => 'TABLE',
      oname => 'employees',
      sname => 'hr',
      use_existing_object => TRUE,
      copy_rows => FALSE);
END;
/
```

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 When Using the Replication Management API

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.

The Replication Management Tool and the Replication Management API

The Replication Management tool uses the replication management API to perform most of its functions. Using the Replication Management tool 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 Management tool's scripting feature. When you start an administrative session with the Replication Management tool, 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 Management tool's help for more information about its scripting feature.

Abbreviations for Datetime and Interval Datatypes

Many of the datetime and interval datatypes have names that are too long to be used with the procedures and functions in the replication management API. Therefore, you must use abbreviations for these datatypes instead of the full names. The following table lists each datatype and its abbreviation. No abbreviation is necessary for the `DATE` and `TIMESTAMP` datatypes.

Datatype	Abbreviation
`TIMESTAMP WITH TIME ZONE`	`TSTZ`
`TIMESTAMP LOCAL TIME ZONE`	`TSLTZ`
`INTERVAL YEAR TO MONTH`	`IYM`
`INTERVAL DAY TO SECOND`	`IDS`

For example, if you want to use the `DBMS_DEFER_QUERY.GET_`*`datatype`*`_ARG` function to determine the value of a `TIMESTAMP LOCAL TIME ZONE` argument in a deferred call, then you substitute `TSLTZ` for *`datatype`*. Therefore, you run the `DBMS_DEFER_QUERY.GET_TSLTZ_ARG` function.

12

DBMS_DEFER

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

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

This chapter discusses the following topics:

- Summary of DBMS_DEFER Subprograms

Summary of DBMS_DEFER Subprograms

Table 12–1 DBMS_DEFER Package Subprograms

Subprogram	Description
CALL Procedure on page 12-3	Builds a deferred call to a remote procedure.
COMMIT_WORK Procedure on page 12-4	Performs a transaction commit after checking for well-formed deferred remote procedure calls.
datatype_ARG Procedure on page 12-5	Provides the data that is to be passed to a deferred remote procedure call.
TRANSACTION Procedure on page 12-8	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 12–2 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 parameters for the procedure, even if some of the parameters have defaults.
nodes	A PL/SQL index-by table of fully qualified database names to which you want to propagate the deferred call. The table is indexed starting at position 1 and continuing until a `NULL` entry is found, or the `no_data_found` exception is raised. The data in the table is case insensitive. This parameter is optional.
group_name	Reserved for internal use.

Exceptions

Table 12–3 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 12–4 COMMIT_WORK Procedure Parameters

Parameter	Description
`commit_work_comment`	Equivalent to the `COMMIT COMMENT` statement in SQL.

Exceptions

Table 12–5 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 procedure call 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('PR', 'MY_PACK', 'MY_PROC', 2);
   DBMS_DEFER.VARCHAR2_ARG('TEST');
   DBMS_DEFER.VARCHAR2_ARG('SALES'); -- required, cannot omit to use default
END;
/
```

Note:

- The `AnyData_ARG` procedure supports the following user-defined types: object types, collections, and `REF`s. See *Oracle9i SQL Reference* and *Oracle9i Application Developer's Guide - Object-Relational Features* for more information about the `AnyData` datatype.
- This procedure uses abbreviations for some datetime and interval datatypes. For example, `TSTZ` is used for the `TIMESTAMP WITH TIME ZONE` datatype. For information about these abbreviations, see "Abbreviations for Datetime and Interval Datatypes" on page 11-4.

Syntax

```
DBMS_DEFER.AnyData_ARG      (arg  IN SYS.AnyData);
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);
DBMS_DEFER.IDS_ARG          (arg  IN DSINTERVAL_UNCONSTRAINED);
DBMS_DEFER.IYM_ARG          (arg  IN YMINTERVAL_UNCONSTRAINED);
DBMS_DEFER.TIMESTAMP_ARG    (arg  IN TIMESTAMP_UNCONSTRAINED);
DBMS_DEFER.TSLTZ_ARG        (arg  IN TIMESTAMP_LTZ_UNCONSTRAINED);
DBMS_DEFER.TSTZ_ARG         (arg  IN TIMESTAMP_TZ_UNCONSTRAINED);
```

Parameters

Table 12–6 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 12–7 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 12–8 TRANSACTION Procedure Parameters

Parameter	Description
nodes	A PL/SQL index-by 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 and continuing until a `NULL` entry is found, or the `no_data_found` exception is raised. The data in the table is case insensitive.

Exceptions

Table 12–9 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.

13

DBMS_DEFER_QUERY

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

This chapter discusses the following topics:

- Summary of DBMS_DEFER_QUERY Subprograms

Summary of DBMS_DEFER_QUERY Subprograms

Table 13–1 DBMS_DEFER_QUERY Package Subprograms

Subprogram	Description
GET_ARG_FORM Function on page 13-3	Determines the form of an argument in a deferred call.
GET_ARG_TYPE Function on page 13-5	Determines the type of an argument in a deferred call.
GET_CALL_ARGS Procedure on page 13-7	Returns the text version of the various arguments for the specified call.
GET_datatype_ARG Function on page 13-9	Determines the value of an argument in a deferred call.
GET_OBJECT_NULL_VECTOR_ARG Function on page 13-12	Returns the type information for a column object.

GET_ARG_FORM Function

This function returns the character set form of a deferred call parameter.

> **See Also:** The Replication Management tool's online help for information about displaying deferred transactions and error transactions in the Replication Management tool

Syntax

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

Parameters

Table 13–2 GET_ARG_FORM Function Parameters

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

Exceptions

Table 13–3 GET_ARG_FORM Function Exceptions

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

Returns

Table 13–4 GET_ARG_FORM Function Returns

Constant Return Value	Return Value	Possible Datatype
DBMS_DEFER_QUERY.ARG_FORM_NONE	0	DATE NUMBER ROWID RAW BLOB User-defined types
DBMS_DEFER_QUERY.ARG_FORM_IMPLICIT	1	CHAR VARCHAR2 CLOB
DBMS_DEFER_QUERY.ARG_FORM_NCHAR	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 Management tool's online help for information about displaying deferred transactions and error transactions in the Replication Management tool

Syntax

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

Parameters

Table 13–5 GET_ARG_TYPE Function Parameters

Parameter	Description
callno	Identification 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 13–6 GET_ARG_TYPE Function Exceptions

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

Returns

Table 13–7 GET_ARG_TYPE Function Returns

Constant Return Value	Return Value	Corresponding Datatype
DBMS_DEFER_QUERY.ARG_TYPE_VARCHAR2	1	VARCHAR2
DBMS_DEFER_QUERY.ARG_TYPE_NUM	2	NUMBER
DBMS_DEFER_QUERY.ARG_TYPE_ROWID	11	ROWID
DBMS_DEFER_QUERY.ARG_TYPE_DATE	12	DATE
DBMS_DEFER_QUERY.ARG_TYPE_RAW	23	RAW
DBMS_DEFER_QUERY.ARG_TYPE_CHAR	96	CHAR
DBMS_DEFER_QUERY.ARG_TYPE_AnyData	109	AnyData
DBMS_DEFER_QUERY.ARG_TYPE_CLOB	112	CLOB
DBMS_DEFER_QUERY.ARG_TYPE_BLOB	113	BLOB
DBMS_DEFER_QUERY.ARG_TYPE_BFIL	114	BFILE
DBMS_DEFER_QUERY.ARG_TYPE_OBJECT_NULL_VECTOR	121	OBJECT_NULL_VECTOR
DBMS_DEFER_QUERY.ARG_TYPE_TIMESTAMP	180	TIMESTAMP
DBMS_DEFER_QUERY.ARG_TYPE_TSTZ	181	TSTZ
DBMS_DEFER_QUERY.ARG_TYPE_IYM	182	IYM
DBMS_DEFER_QUERY.ARG_TYPE_IDS	183	IDS
DBMS_DEFER_QUERY.ARG_TYPE_TSLTZ	231	TSLTZ

Note:

- The AnyData datatype supports the following user-defined types: object types, collections, and REFs. See *Oracle9i SQL Reference* and *Oracle9i Application Developer's Guide - Object-Relational Features* for more information about the AnyData datatype.
- This function uses abbreviations for some datetime and interval datatypes. For example, TSTZ is used for the TIMESTAMP WITH TIME ZONE datatype. For information about these abbreviations, see "Abbreviations for Datetime and Interval Datatypes" on page 11-4.

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.

See Also:

- "GET_datatype_ARG Function" on page 13-9
- *Oracle9i SQL Reference* and *Oracle9i Application Developer's Guide - Object-Relational Features* for more information about the AnyData datatype

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 13–8 GET_CALL_ARGS Procedure Parameters

Parameter	Description
callno	Identification number from the DEFCALL view of the deferred remote procedure call (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 is 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 13–9 GET_CALL_ARGS Procedure Exceptions

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

GET_*datatype*_ARG Function

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

The AnyData type supports the following user-defined types: object types, collections and REFs. Not all types supported by this function can be enqueued by the AnyData_ARG procedure in the DBMS_DEFER package.

The returned text for type arguments includes the following values: type owner, type name, type version, length, precision, scale, character set identifier, character set form, and number of elements for collections or number of attributes for object types. These values are separated by a colon (:).

See Also:

- "datatype_ARG Procedure" on page 12-5
- The Replication Management tool's online help for information about displaying deferred transactions and error transactions in the Replication Management tool
- *Oracle9i SQL Reference* and *Oracle9i Application Developer's Guide - Object-Relational Features* for more information about the AnyData datatype
- This function uses abbreviations for some datetime and interval datatypes. For example, TSTZ is used for the TIMESTAMP WITH TIME ZONE datatype. For information about these abbreviations, see "Abbreviations for Datetime and Interval Datatypes" on page 11-4.

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* is:

```
{ AnyData
| NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| ROWID
| BLOB
| CLOB
| NCLOB
| NCHAR
| NVARCHAR2
| IDS
| IYM
| TIMESTAMP
| TSLTZ
| TSTZ }
```

Parameters

Table 13–10 GET_datatype_ARG Function Parameters

Parameter	Description
callno	Identification 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 the GET_ARG_TYPE function. The default is NULL.

Exceptions

Table 13–11 GET_datatype_ARG Function Exceptions

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

GET_OBJECT_NULL_VECTOR_ARG Function

This function returns the type information for a column object, including the type owner, name, and hashcode.

Syntax

```
DBMS_DEFER_QUERY.GET_OBJECT-NULL_VECTOR_ARG (
   callno               IN   NUMBER,
   arg_no               IN   NUMBER,
   deferred_tran_id     IN   VARCHAR2)
  RETURN SYSTEM.REPCAT$_OBJECT_NULL_VECTOR;
```

Parameters

Table 13–12   GET_OBJECT_NULL_VECTOR_ARG 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 identification.

Exceptions

Table 13–13   GET_OBJECT_NULL_VECTOR_ARG Function Exceptions

Exception	Description
`NO_DATA_FOUND`	Input parameters do not correspond to a parameter of a deferred call.
`ORA-26564`	Parameter is not an `object_null_vector` type.

Returns

Table 13–14 GET_OBJECT_NULL_VECTOR_ARG Function Returns

Return Value	Type Definition
`SYSTEM.REPCAT$_OBJECT_NULL_VECTOR` type	`CREATE TYPE SYSTEM.REPCAT$_OBJECT_NULL_VECTOR AS OBJECT ( type_owner VARCHAR2(30), type_name VARCHAR2(30), type_hashcode RAW(17), null_vector RAW(2000));`

14

DBMS_DEFER_SYS

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

This chapter discusses the following topics:

Summary of DBMS_DEFER_SYS Subprograms

Table 14–1 DBMS_DEFER_SYS Package Subprograms (Page 1 of 2)

Subprogram	Description
ADD_DEFAULT_DEST Procedure on page 14-4	Adds a destination database to the DEFDEFAULTDEST view.
CLEAR_PROP_STATISTICS Procedure on page 14-5	Clears the propagation statistics in the DEFSCHEDULE data dictionary view.
DELETE_DEFAULT_DEST Procedure on page 14-6	Removes a destination database from the DEFDEFAULTDEST view.
DELETE_DEF_DESTINATION Procedure on page 14-6	Removes a destination database from the DEFSCHEDULE view.
DELETE_ERROR Procedure on page 14-7	Deletes a transaction from the DEFERROR view.
DELETE_TRAN Procedure on page 14-8	Deletes a transaction from the DEFTRANDEST view.
DISABLED Function on page 14-9	Determines whether propagation of the deferred transaction queue from the current site to a specified site is enabled.
EXCLUDE_PUSH Function on page 14-10	Acquires an exclusive lock that prevents deferred transaction PUSH.
EXECUTE_ERROR Procedure on page 14-11	Reexecutes 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 14-12	Reexecutes a deferred transaction that did not initially complete successfully in the security context of the user who executes this procedure.
PURGE Function on page 14-13	Purges pushed transactions from the deferred transaction queue at your current master site or materialized view site.
PUSH Function on page 14-16	Forces a deferred remote procedure call queue at your current master site or materialized view site to be pushed to a remote site.

Table 14–1 DBMS_DEFER_SYS Package Subprograms **(Page 2 of 2)**

Subprogram	Description
REGISTER_PROPAGATOR Procedure on page 14-19	Registers the specified user as the propagator for the local database.
SCHEDULE_PURGE Procedure on page 14-20	Schedules a job to purge pushed transactions from the deferred transaction queue at your current master site or materialized view site.
SCHEDULE_PUSH Procedure on page 14-22	Schedules a job to push the deferred transaction queue to a remote site.
SET_DISABLED Procedure on page 14-24	Disables or enables propagation of the deferred transaction queue from the current site to a specified destination site.
UNREGISTER_PROPAGATOR Procedure on page 14-26	Unregisters a user as the propagator from the local database.
UNSCHEDULE_PURGE Procedure on page 14-27	Stops automatic purges of pushed transactions from the deferred transaction queue at a master site or materialized view site.
UNSCHEDULE_PUSH Procedure on page 14-27	Stops automatic pushes of the deferred transaction queue from a master site or materialized view site to a remote site.

ADD_DEFAULT_DEST Procedure

This procedure adds a destination database to the DEFDEFAULTDEST data dictionary view.

Syntax

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

Parameters

Table 14–2 ADD_DEFAULT_DEST Procedure Parameters

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

Exceptions

Table 14–3 ADD_DEFAULT_DEST Procedure Exceptions

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

CLEAR_PROP_STATISTICS Procedure

This procedure clears the propagation statistics in the DEFSCHEDULE data dictionary view. When this procedure is executed successfully, all statistics in this view are returned to zero and statistic gathering starts fresh.

Specifically, this procedure clears statistics from the following columns in the DEFSCHEDULE data dictionary view:

- TOTAL_TXN_COUNT
- AVG_THROUGHPUT
- AVG_LATENCY
- TOTAL_BYTES_SENT
- TOTAL_BYTES_RECEIVED
- TOTAL_ROUND_TRIPS
- TOTAL_ADMIN_COUNT
- TOTAL_ERROR_COUNT
- TOTAL_SLEEP_TIME

Syntax

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

Parameters

Table 14–4 CLEAR_PROP_STATISTICS Procedure Parameters

Parameter	Description
dblink	The fully qualified database name of the node whose statistics you want to clear. The statistics to be cleared are the statistics for propagation of deferred transactions from the current node to the node you specify for dblink.

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 14–5 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 14–6 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 14–7 DELETE_ERROR Procedure Parameters

Parameter	Description
deferred_tran_id	Identification 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 14–8 DELETE_TRAN Procedure Parameters

Parameter	Description
deferred_tran_id	Identification 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 14–9 DISABLED Function Parameters

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

Returns

Table 14–10 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 14–11 DISABLED Function Exceptions

Exception	Description
`NO_DATA_FOUND`	Specified `destination` does not appear in the `DEFSCHEDULE` view.

EXCLUDE_PUSH Function

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 14–12 EXCLUDE_PUSH Function Parameters

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

Returns

Table 14–13 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 reexecutes 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 14–14 EXECUTE_ERROR Procedure Parameters

Parameter	Description
deferred_tran_id	Identification number from the DEFERROR view of the deferred transaction that you want to reexecute. If this is NULL, then all transactions queued for destination are reexecuted.
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 14–15 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 reexecutes 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 14–16   EXECUTE_ERROR_AS_USER Procedure Parameters

Parameter	Description
`deferred_tran_id`	Identification number from the `DEFERROR` view of the deferred transaction that you want to reexecute. If this is `NULL`, then all transactions queued for `destination` are reexecuted.
`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 14–17   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 site or materialized view 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 14–18 PURGE Function Parameters **(Page 1 of 2)**

Parameter	Description

`purge_method`	Controls how to purge the deferred transaction queue: `purge_method_quick` costs less, while `purge_method_precise` offers better precision.
	
	Specify the following for this parameter to use `purge_method_quick`:
	
	`dbms_defer_sys.purge_method_quick`
	
	Specify the following for this parameter to user `purge_method_precise`:
	
	`dbms_defer_sys.purge_method_precise`
	
	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 14-15 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.

Table 14–18 PURGE Function Parameters **(Page 2 of 2)**

Parameter	Description
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. When set to false, Oracle does not record the result value.

Returns

Table 14–19 Purge Function Returns

Value	Description
result_ok	OK, terminated after delay_seconds expired.
result_startup_seconds	Terminated by lock timeout while starting.
result_execution_seconds	Terminated by exceeding execution_seconds.
result_transaction_count	Terminated by exceeding transaction_count.
result_errors	Terminated after errors.
result_split_del_order_limit	Terminated after failing to acquire the enqueue in exclusive mode. If you receive this return code, then retry the purge. If the problem persists, then contact Oracle Support Services.
result_purge_disabled	Queue purging is disabled internally for synchronization when adding new master sites without quiesce.

Exceptions

Table 14–20 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 call (RPC) queue at your current master site or materialized view site to be pushed (propagated) to a remote 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 14–21 PUSH Function Parameters **(Page 1 of 2)**

Parameter	Description
destination	The fully qualified database name of the master site or master materialized view site to which you are forwarding changes.
parallelism	0 specifies serial propagation. *n* > 1 specifies parallel propagation with *n* parallel processes. 1 specifies parallel propagation using only one parallel process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. **Note:** Do not set the parameter unless so directed by Oracle Support Services.
stop_on_error	The default, `false`, indicates that the executor should continue even if errors, such as conflicts, are encountered. If `true`, then stops propagation at the first indication that a transaction encountered an error at the destination site.

Table 14–21 PUSH Function Parameters **(Page 2 of 2)**

Parameter	Description
write_trace	When set to true, Oracle records the result value returned by the function in the server's trace file. When set to false, Oracle does not record the result value.
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.
delivery_order_limit	Stop execution cleanly before pushing a transaction where delivery_order >= delivery_order_limit

Returns

Table 14–22 PUSH Function Returns

Value	Description
`result_ok`	OK, terminated after `delay_seconds` expired.
`result_startup_seconds`	Terminated by lock timeout while starting.
`result_execution_seconds`	Terminated by exceeding `execution_seconds`.
`result_transaction_count`	Terminated by exceeding `transaction_count`.
`result_delivery_order_limit`	Terminated by exceeding `delivery_order_limit`.
`result_errors`	Terminated after errors.
`result_push_disabled`	Push was disabled internally. Typically, this return value means that propagation to the destination was set to disabled internally by Oracle for propagation synchronization when adding a new master site to a master group without quiescing the master group. Oracle will enable propagation automatically at a later time
`result_split_del_order_limit`	Terminated after failing to acquire the enqueue in exclusive mode. If you receive this return code, then retry the push. If the problem persists, then contact Oracle Support Services.

Exceptions

Table 14–23 PUSH Function Exceptions

Exception	Description
`incompleteparallelpush`	Serial propagation requires that parallel propagation shuts down cleanly.
`executiondisabled`	Execution of deferred remote procedure calls (RPCs) is disabled at the destination.
`crt_err_err`	Error while creating entry in `DEFERROR`.
`deferred_rpc_quiesce`	Replication activity for replication group is suspended.
`commfailure`	Communication failure during deferred remote procedure call (RPC).
`missingpropagator`	A propagator does not exist.

REGISTER_PROPAGATOR Procedure

This procedure registers the specified user as the propagator for the local database. It also grants the following privileges to the specified user (so that the user can create wrappers):

- CREATE SESSION
- CREATE PROCEDURE
- CREATE DATABASE LINK
- EXECUTE ANY PROCEDURE

Syntax

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

Parameters

Table 14–24 REGISTER_PROPAGATOR Procedure Parameters

Parameter	Description
username	Name of the user.

Exceptions

Table 14–25 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 site or materialized view site. You should schedule one purge job.

> **See Also:** *Oracle9i Replication* for information about using this procedure to schedule continuous or periodic purge of your deferred transaction queue

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 14–26 SCHEDULE_PURGE Procedure Parameters **(Page 1 of 2)**

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`.

Table 14–26 SCHEDULE_PURGE Procedure Parameters **(Page 2 of 2)**

Parameter	Description
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. Specify the following for this parameter to use purge_method_quick: dbms_defer_sys.purge_method_quick Specify the following for this parameter to user purge_method_precise: dbms_defer_sys.purge_method_precise 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. For more information, see "Usage Notes" on page 14-15. These usage notes are for the DBMS_DEFER_SYS.PURGE function, but they also apply to the DBMS_DEFER_SYS.SCHEDULE_PURGE procedure.
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 site. This procedure performs a `COMMIT`.

> **See Also:** *Oracle9i Replication* for information about using this procedure to schedule continuous or periodic push of your deferred transaction queue

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 14–27 SCHEDULE_PUSH Procedure Parameters **(Page 1 of 2)**

Parameter	Description
`destination`	The fully qualified database name of the master site or master materialized view site 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`.

Table 14–27 SCHEDULE_PUSH Procedure Parameters **(Page 2 of 2)**

Parameter	Description
next_date	Allows you to specify a time to push deferred transactions to the remote site. This value is stored in the next_date field of the DEFSCHEDULE view. If you use the default value for this parameter, NULL, then the value of this field remains unchanged. If this field had no previous value, then it is created with a value of NULL. If you do not supply a value for this field, then you must supply a value for interval.
reset	Set to true to reset LAST_TXN_COUNT, LST_ERROR, and LAST_MSG to NULL.
parallelism	0 specifies serial propagation. *n* > 1 specifies parallel propagation with *n* parallel processes. 1 specifies parallel propagation using only one parallel process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. **Note:** Do not set the parameter unless so directed by Oracle Support Services.
stop_on_error	The default, false, indicates that the executor should continue even if errors, such as conflicts, are encountered. If true, then stops propagation 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,
   catchup       IN   RAW := '00',
   override      IN   BOOLEAN := false);
```

Parameters

Table 14–28 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.
catchup	The extension identifier for adding new master sites to a master group without quiescing the master group. The new master site is the destination. Query the `DEFSCHEDULE` data dictionary view for the existing extension identifiers.
override	A `false` setting, the default, specifies that Oracle raises the `cantsetdisabled` exception if the `disabled` parameter is set to `false` and propagation was disabled internally by Oracle. A `true` setting specifies that Oracle ignores whether the disabled state was set internally for synchronization and always tries to set the state as specified by the `disabled` parameter. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 14–29 SET_DISABLED Procedure Exceptions

Exception	Description
NO_DATA_FOUND	No entry was found in the `DEFSCHEDULE` view for the specified `destination`.
cantsetdisabled	The disabled status for this site is set internally by Oracle for synchronization during adding a new master site to a master group without quiescing the master group. Ensure that adding a new master site without quiescing finished before invoking this procedure.

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 14–30 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 14–31 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 master site or materialized view site.

Syntax

```
DBMS_DEFER_SYS.UNSCHEDULE_PURGE();
```

Parameters

None

UNSCHEDULE_PUSH Procedure

This procedure stops automatic pushes of the deferred transaction queue from a master site or materialized view site to a remote site.

Syntax

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

Parameters

Table 14–32 UNSCHEDULE_PUSH Procedure Parameters

Parameter	Description
`dblink`	Fully qualified path name for the database at which you want to unschedule periodic execution of deferred remote procedure calls.

Table 14–33 UNSCHEDULE_PUSH Procedure Exceptions

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

15

DBMS_MVIEW

DBMS_MVIEW enables you to understand capabilities for materialized views and potential materialized views, including their rewrite availability. It also enables you to refresh materialized views that are not part of the same refresh group and purge logs.

This chapter discusses the following topics:

- Summary of DBMS_MVIEW Subprograms

Note: DBMS_SNAPSHOT is a synonym for DBMS_MVIEW.

See Also:

- *Oracle9i Replication* for more information about using materialized views in a replication environment
- *Oracle9i Data Warehousing Guide* for more information about using materialized views in a data warehousing environment

Summary of DBMS_MVIEW Subprograms

Table 15–1 DBMS_MVIEW Package Subprograms

Subprogram	Description
BEGIN_TABLE_REORGANIZATION Procedure on page 15-3	Performs a process to preserve materialized view data needed for refresh.
END_TABLE_REORGANIZATION Procedure on page 15-4	Ensures that the materialized view data for the master table is valid and that the master table is in the proper state.
EXPLAIN_MVIEW Procedure on page 15-5	Explains what is possible with a materialized view or potential materialized view.
EXPLAIN_REWRITE Procedure on page 15-6	Explains why a query failed to rewrite.
I_AM_A_REFRESH Function on page 15-7	Returns the value of the I_AM_REFRESH package state.
PMARKER Function on page 15-8	Returns a partition marker from a rowid. This function is used for Partition Change Tracking (PCT).
PURGE_DIRECT_LOAD_LOG Procedure on page 15-8	Purges rows from the direct loader log after they are no longer needed by any materialized views (used with data warehousing).
PURGE_LOG Procedure on page 15-9	Purges rows from the materialized view log.
PURGE_MVIEW_FROM_LOG Procedure on page 15-10	Purges rows from the materialized view log.
REFRESH Procedure on page 15-12	Consistently refreshes one or more materialized views that are not members of the same refresh group.
REFRESH_ALL_MVIEWS Procedure on page 15-15	Refreshes all materialized views that do not reflect changes to their master table or master materialized view.
REFRESH_DEPENDENT Procedure on page 15-16	Refreshes all table-based materialized views that depend on a specified master table or master materialized view, or list of master tables or master materialized views.
REGISTER_MVIEW Procedure on page 15-18	Enables the administration of individual materialized views.
UNREGISTER_MVIEW Procedure on page 15-21	Enables the administration of individual materialized views. Invoked at a master site or master materialized view site to unregister a materialized view.

BEGIN_TABLE_REORGANIZATION Procedure

This procedure performs a process to preserve materialized view data needed for refresh. It must be called before a master table is reorganized.

See Also: "Reorganizing Master Tables that Have Materialized View Logs" on page 8-19

Syntax

```
DBMS_MVIEW.BEGIN_TABLE_REORGANIZATION (
   tabowner    IN   VARCHAR2,
   tabname     IN   VARCHAR2);
```

Parameters

Table 15–2 BEGIN_TABLE_REORGANIZATION Procedure Parameters

Parameter	Description
tabowner	Owner of the table being reorganized.
tabname	Name of the table being reorganized.

END_TABLE_REORGANIZATION Procedure

This procedure ensures that the materialized view data for the master table is valid and that the master table is in the proper state. It must be called after a master table is reorganized.

See Also: "Reorganizing Master Tables that Have Materialized View Logs" on page 8-19

Syntax

```
DBMS_MVIEW.END_TABLE_REORGANIZATION (
   tabowner    IN   VARCHAR2,
   tabname     IN   VARCHAR2);
```

Parameters

Table 15–3 END_TABLE_REORGANIZATION Procedure Parameters

Parameter	Description
tabowner	Owner of the table being reorganized.
tabname	Name of the table being reorganized.

EXPLAIN_MVIEW Procedure

This procedure enables you to learn what is possible with a materialized view or potential materialized view. For example, you can determine if a materialized view is fast refreshable and what types of query rewrite you can perform with a particular materialized view.

Using this procedure is straightforward. You simply call DBMS_MVIEW.EXPLAIN_MVIEW, passing in as parameters the schema and materialized view name for an existing materialized view. Alternatively, you can specify the SELECT string for a potential materialized view. The materialized view or potential materialized view is then analyzed and the results are written into either a table called MV_CAPABILITIES_TABLE, which is the default, or to an array called MSG_ARRAY.

Note that you must run the utlxmv.sql script prior to calling EXPLAIN_MVIEW except when you direct output to a VARRAY. The script is found in the admin directory. In addition, you must create MV_CAPABILITIES_TABLE in the current schema.

Syntax

The following PL/SQL declarations that are made for you in the DBMS_MVIEW package show the order and datatypes of these parameters for explaining an existing materialized view and a potential materialized view with output to a table and to a VARRAY.

To explain an existing or potential materialized view with output to MV_CAPABILITIES_TABLE:

```
DBMS_MVIEW.EXPLAIN_MVIEW (
 mv            IN VARCHAR2,
 statement_id  IN VARCHAR2:= NULL);
```

To explain an existing or potential materialized view with output to a VARRAY:

```
DBMS_MVIEW.EXPLAIN_MVIEW (
 mv          IN VARCHAR2,
 msg_array   OUT SYS.ExplainMVArrayType);
```

Parameters

Table 15–4 EXPLAIN_MVIEW Procedure Parameters

Parameter	Description
mv	The name of an existing materialized view (optionally qualified with the owner name separated by a ".") or a SELECT statement for a potential materialized view.
statement_id	A client-supplied unique identifier to associate output rows with specific invocations of EXPLAIN_MVIEW.
msg_array	The PL/SQL varray that receives the output. Use this parameter to direct EXPLAIN_MVIEW's output to a PL/SQL VARRAY rather than MV_CAPABILITIES_TABLE.

EXPLAIN_REWRITE Procedure

This procedure enables you to learn why a query failed to rewrite, or, if it rewrites, which materialized views will be used. Using the results from the procedure, you can take the appropriate action needed to make a query rewrite if at all possible. The query specified in the EXPLAIN_REWRITE statement is never actually executed.

To obtain the output into a table, you must run the admin/utlxrw.sql script before calling EXPLAIN_REWRITE. This script creates a table named REWRITE_TABLE in the current schema.

Syntax

You can obtain the output from EXPLAIN_REWRITE in two ways. The first is to use a table, while the second is to create a VARRAY. The following shows the basic syntax for using an output table:

```
DBMS_MVIEW.EXPLAIN_REWRITE (
    query           IN VARCHAR2,
    mv              IN VARCHAR2,
    statement_id    IN VARCHAR2;
```

If you want to direct the output of EXPLAIN_REWRITE to a varray, instead of a table, then the procedure should be called as follows:

```
DBMS_MVIEW.EXPLAIN_REWRITE (
    query           IN VARCHAR2(2000),
    mv              IN VARCHAR2(30),
    msg_array       IN OUT SYS.RewriteArrayType);
```

Parameters

Table 15–5 EXPLAIN_REWRITE Procedure Parameters

Parameter	Description
query	SQL select statement to be explained.
mv	The fully qualified name of an existing materialized view in the form of SCHEMA.MV
statement_id	A client-supplied unique identifier to distinguish output messages
msg_array	The PL/SQL varray that receives the output. Use this parameter to direct EXPLAIN_REWRITE's output to a PL/SQL VARRAY

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 materialized views 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_MVIEW.I_AM_A_REFRESH()
  RETURN BOOLEAN;
```

Parameters

None.

PMARKER Function

This function returns a partition marker from a rowid. It is used for Partition Change Tracking (PCT).

Syntax

```
DBMS_MVIEW.PMARKER(rid IN ROWID)
  RETURN NUMBER;
```

Parameters

Table 15–6 PMARKER Procedure Parameters

Parameter	Description
`rid`	The rowid of a row entry in a master table.

PURGE_DIRECT_LOAD_LOG Procedure

This procedure removes entries from the direct loader log after they are no longer needed for any known materialized view. This procedure usually is used in environments using Oracle's data warehousing technology.

See Also: *Oracle9i Data Warehousing Guide* for more information

Syntax

```
DBMS_MVIEW.PURGE_DIRECT_LOAD_LOG();
```

Parameters

None.

PURGE_LOG Procedure

This procedure purges rows from the materialized view log.

Syntax

```
DBMS_MVIEW.PURGE_LOG (
   master        IN   VARCHAR2,
   num           IN   BINARY_INTEGER := 1,
   flag          IN   VARCHAR2       := 'NOP');
```

Parameters

Table 15–7 PURGE_LOG Procedure Parameters

Parameter	Description
`master`	Name of the master table or master materialized view.

`num`	Number of least recently refreshed materialized views whose rows you want to remove from materialized view log. For example, the following statement deletes rows needed to refresh the two least recently refreshed materialized views:
	`DBMS_MVIEW.PURGE_LOG('master_table', 2);`
	To delete all rows in the materialized view log, indicate a high number of materialized views to disregard, as in this example:
	`DBMS_MVIEW.PURGE_LOG('master_table',9999);`
	This statement completely purges the materialized view log that corresponds to `master_table` if fewer than 9999 materialized views are based on `master_table`. A simple materialized view whose rows have been purged from the materialized view log must be completely refreshed the next time it is refreshed.

`flag`	Specify `delete` to guarantee that rows are deleted from the materialized view log for at least one materialized view. This parameter can override the setting for the parameter `num`. For example, the following statement deletes rows from the materialized view log that has dependency rows in the least recently refreshed materialized view: `DBMS_MVIEW.PURGE_LOG('master_table',1,'delete');`

PURGE_MVIEW_FROM_LOG Procedure

This procedure is called on the master site or master materialized view site to delete the rows in materialized view refresh related data dictionary tables maintained at the master for the specified materialized view identified by its `mview_id` or the combination of the `mviewowner`, `mviewname`, and the `mviewsite`. If the materialized view specified is the oldest materialized view to have refreshed from any of the master tables or master materialized views, then the materialized view log is also purged. This procedure does not unregister the materialized view.

If there is an error while purging one of the materialized view logs, the successful purge operations of the previous materialized view logs are not rolled back. This is to minimize the size of the materialized view logs. In case of an error, this procedure can be invoked again until all the materialized view logs are purged.

Syntax

```
DBMS_MVIEW.PURGE_MVIEW_FROM_LOG (
   mview_id       IN   BINARY_INTEGER  |
   mviewowner     IN   VARCHAR2,
   mviewname      IN   VARCHAR2,
   mviewsite      IN   VARCHAR2);
```

Note: This procedure is overloaded. The `mview_id` parameter is mutually exclusive with the three remaining parameters: `mviewowner`, `mviewname`, and `mviewsite`.

Parameters

Table 15–8 PURGE_MVIEW_FROM_LOG Procedure Parameters

Parameter	Description
mview_id	If you want to execute this procedure based on the identification of the target materialized view, specify the materialized view identification using the mview_id parameter. Query the DBA_BASE_TABLE_MVIEWS view at the materialized view log site for a listing of materialized view IDs. Executing this procedure based on the materialized view identification is useful if the target materialized view is not listed in the list of registered materialized views (DBA_REGISTERED_MVIEWS).
mviewowner	If you do not specify a mview_id, enter the owner of the target materialized view using the mviewowner parameter. Query the DBA_REGISTERED_MVIEWS view at the materialized view log site to view the materialized view owners.
mviewname	If you do not specify a mview_id, enter the name of the target materialized view using the mviewname parameter. Query the DBA_REGISTERED_MVIEWS view at the materialized view log site to view the materialized view names.
mviewsite	If you do not specify a mview_id, enter the site of the target materialized view using the mviewsite parameter. Query the DBA_REGISTERED_MVIEWS view at the materialized view log site to view the materialized view sites.

REFRESH Procedure

This procedure refreshes a list of materialized views.

Syntax

```
DBMS_MVIEW.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 15–9 REFRESH Procedure Parameters **(Page 1 of 2)**

Parameter	Description
list | tab	Comma-separated list of materialized views that you want to refresh. (Synonyms are not supported.) These materialized views can be located in different schemas and have different master tables or master materialized views. However, all of the listed materialized views must be in your local database. Alternatively, you may pass in a PL/SQL index-by table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a materialized view.
method	A string of refresh methods indicating how to refresh the listed materialized views. An f indicates fast refresh, ? indicates force refresh, C or c indicates complete refresh, and A or a indicates always refresh. A and C are equivalent. If a materialized view does not have a corresponding refresh method (that is, if more materialized views are specified than refresh methods), then that materialized view is refreshed according to its default refresh method. For example, consider the following EXECUTE statement within SQL*Plus: DBMS_MVIEW.REFRESH ('countries_mv,regions_mv,hr.employees_mv','cf'); This statement performs a complete refresh of the countries_mv materialized view, a fast refresh of the regions_mv materialized view, and a default refresh of the hr.employees materialized view.
rollback_seg	Name of the materialized view site rollback segment to use while refreshing materialized views.
push_deferred_rpc	Used by updatable materialized views only. Set this parameter to true if you want to push changes from the materialized view to its associated master tables or master materialized views before refreshing the materialized view. Otherwise, these changes may appear to be temporarily lost.
refresh_after_errors	If this parameter is true, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the materialized view's master table or master materialized view. If this parameter is true and atomic_refresh is false, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.

Table 15–9 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, then set this parameter to `0` and occasionally execute `PUSH` with this parameter set to `2` to reduce the queue.
parallelism	0 specifies serial propagation. *n* > 1 specifies parallel propagation with *n* parallel processes. 1 specifies parallel propagation using only one parallel process.
heap_size	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.
atomic_refresh	If this parameter is set to `true`, then the list of materialized views is refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to `false`, then each of the materialized views 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_ALL_MVIEWS Procedure

This procedure refreshes all materialized views that have the following properties:

- The materialized view has not been refreshed since the most recent change to a master table or master materialized view on which it depends.
- The materialized view and all of the master tables or master materialized views on which it depends are local.
- The materialized view is in the view DBA_MVIEWS.

This procedure is intended for use with data warehouses.

Syntax

```
DBMS_MVIEW.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 15–10 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 materialized view 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. A and C are equivalent. If no method is specified, a materialized view is refreshed according to its default refresh method.
rollback_seg	Name of the materialized view site rollback segment to use while refreshing materialized views.
refresh_after_errors	If this parameter is true, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the DEFERROR view for the materialized view's master table or master materialized view. If this parameter is true and atomic_refresh is false, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.

Table 15–10 REFRESH_ALL_MVIEWS Procedure Parameters **(Page 2 of 2)**

Parameter	Description
atomic_refresh	If this parameter is set to `true`, then the refreshed materialized views are refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to `false`, then each of the refreshed materialized views 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 materialized views that have the following properties:

- The materialized view depends on a master table or master materialized view in the list of specified masters.
- The materialized view has not been refreshed since the most recent change to a master table or master materialized view on which it depends.
- The materialized view and all of the master tables or master materialized views on which it depends are local.
- The materialized view is in the view `DBA_MVIEWS`.

This procedure is intended for use with data warehouses.

Syntax

```
DBMS_MVIEW.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 15–11 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 or master materialized views on which materialized views can depend. (Synonyms are not supported.) These tables and the materialized views that depend on them can be located in different schemas. However, all of the tables and materialized views must be in your local database. Alternatively, you may pass in a PL/SQL index-by 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 materialized views. All of the materialized views 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. `A` and `C` are equivalent.
	
	If a table does not have a corresponding refresh method (that is, if more tables are specified than refresh methods), then any materialized view that depends on that table is refreshed according to its default refresh method. For example, the following `EXECUTE` statement within SQL*Plus:
	
	`DBMS_MVIEW.REFRESH_DEPENDENT`
	`('employees,deptartments,hr.regions','cf');`
	
	performs a complete refresh of the materialized views that depend on the `employees` table, a fast refresh of the materialized views that depend on the `departments` table, and a default refresh of the materialized views that depend on the `hr.regions` table.

`rollback_seg`	Name of the materialized view site rollback segment to use while refreshing materialized views.
`refresh_after_errors`	If this parameter is `true`, an updatable materialized view continues to refresh even if there are outstanding conflicts logged in the `DEFERROR` view for the materialized view's master table or master materialized view. If this parameter is `true` and `atomic_refresh` is `false`, this procedure continues to refresh other materialized views if it fails while refreshing a materialized view.

Table 15–11 REFRESH_DEPENDENT Procedure Parameters (Page 2 of 2)

Parameter	Description
atomic_refresh	If this parameter is set to `true`, then the refreshed materialized views are refreshed in a single transaction. All of the refreshed materialized views are updated to a single point in time. If the refresh fails for any of the materialized views, none of the materialized views are updated. If this parameter is set to `false`, then each of the refreshed materialized views is refreshed in a separate transaction. The number of job queue processes must be set to 1 or greater if this parameter is `false`.

REGISTER_MVIEW Procedure

This procedure enables the administration of individual materialized views. It is invoked at a master site or master materialized view site to register a materialized view.

> **Note:** Typically, a materialized view is registered automatically during materialized view creation. You should only run this procedure to manually register a materialized view if the automatic registration failed or if the registration information was deleted.

Syntax

```
DBMS_MVIEW.REGISTER_MVIEW (
   mviewowner  IN   VARCHAR2,
   mviewname   IN   VARCHAR2,
   mviewsite   IN   VARCHAR2,
   mview_id    IN   DATE | BINARY_INTEGER,
   flag        IN   BINARY_INTEGER,
   qry_txt     IN   VARCHAR2,
   rep_type    IN   BINARY_INTEGER := DBMS_MVIEW.REG_UNKNOWN);
```

Parameters

Table 15–12 REGISTER_MVIEW Procedure Parameters

Parameter	Description
mviewowner	Owner of the materialized view.
mviewname	Name of the materialized view.
mviewsite	Name of the materialized view site for a materialized view registering at an Oracle8 and higher master site or master materialized view site. This name should not contain any double quotes.
mview_id	The identification number of the materialized view. Specify an Oracle8 and higher materialized view as a BINARY_INTEGER. Specify an Oracle7 materialized view registering at an Oracle8 and higher master sites or master materialized view sites as a DATE.
flag	A constant that describes the properties of the materialized view being registered. Valid constants that can be assigned include the following: ■ dbms_mview.reg_rowid_mview for a rowid materialized view ■ dbms_mview.reg_primary_key_mview for a primary key materialized view ■ dbms_mview.reg_object_id_mview for an object id materialized view ■ dbms_mview.reg_fast_refreshable_mview for a materialized view that can be fast refreshed ■ dbms_mview.reg_updatable_mview for a materialized view that is updatable A materialized view can have more than one of these properties. In this case, use the plus sign (+) to specify more than one property. For example, if a primary key materialized view can be fast refreshed, you can enter the following for this parameter: dbms_mview.reg_primary_key_mview + dbms_mview.reg_fast_refreshable_mview You can determine the properties of a materialized view by querying the ALL_MVIEWS data dictionary view.
qry_txt	The first 32,000 bytes of the materialized view definition query.

Table 15–12 REGISTER_MVIEW Procedure Parameters

Parameter	Description
`rep_type`	Version of the materialized view. Valid constants that can be assigned include the following: ■ `dbms_mview.reg_v7_snapshot` if the materialized view is at an Oracle7 site ■ `dbms_mview.reg_v8_snapshot` if the materialized view is at an Oracle8 or higher site ■ `dbms_mview.reg_unknown` (the default) if you do not know whether the materialized view is at an Oracle7 site or an Oracle8 (or higher) site

Usage Notes

This procedure is invoked at the master site or master materialized view site by a remote materialized view site using a remote procedure call. If `REGISTER_MVIEW` is called multiple times with the same `mviewowner`, `mviewname`, and `mviewsite`, then the most recent values for `mview_id`, `flag`, and `qry_txt` are stored. If a query exceeds the maximum `VARCHAR2` size, then `qry_txt` contains the first 32000 characters of the query and the remainder is truncated. When invoked manually, the value of `mview_id` must be looked up in the materialized view data dictionary views by the person who calls the procedure.

UNREGISTER_MVIEW Procedure

This procedure enables the administration of individual materialized views. It is invoked at a master site or master materialized view site to unregister a materialized view.

Syntax

```
DBMS_MVIEW.UNREGISTER_MVIEW (
   mviewowner      IN   VARCHAR2,
   mviewname       IN   VARCHAR2,
   mviewsite       IN   VARCHAR2);
```

Parameters

Table 15–13 UNREGISTER_MVIEW Procedure Parameters

Parameters	Description
mviewowner	Owner of the materialized view.
mviewname	Name of the materialized view.
mviewsite	Name of the materialized view site.

16

DBMS_OFFLINE_OG

The DBMS_OFFLINE_OG package contains public APIs for offline instantiation of master groups.

This chapter discusses the following topics:

- Summary of DBMS_OFFLINE_OG Subprograms

Note: These procedures are used in performing an offline instantiation of a master table in a multimaster replication environment.

These procedure should not be confused with the procedures in the DBMS_OFFLINE_SNAPSHOT package (used for performing an offline instantiation of a materialized view) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Summary of DBMS_OFFLINE_OG Subprograms

Table 16–1 DBMS_OFFLINE_OG Package Subprograms

Subprogram	Description
BEGIN_INSTANTIATION Procedure on page 16-3	Starts offline instantiation of a master group.
BEGIN_LOAD Procedure on page 16-4	Disables triggers while data is imported to new master site as part of offline instantiation.
END_INSTANTIATION Procedure on page 16-6	Completes offline instantiation of a master group.
END_LOAD Procedure on page 16-7	Re-enables triggers after importing data to new master site as part of offline instantiation.
RESUME_SUBSET_OF_MASTERS Procedure on page 16-9	Resumes replication activity at all existing sites except the new site during offline instantiation of a master group.

BEGIN_INSTANTIATION Procedure

This procedure starts offline instantiation of a master group. You must call this procedure from the master definition site.

Note: This procedure is used to perform 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 materialized view) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

See Also: "Adding New Master Sites with Offline Instantiation Using Export/Import" on page 7-31 for information about adding a new master site to a master group by performing an offline instantiation of a master site

Syntax

```
DBMS_OFFLINE_OG.BEGIN_INSTANTIATION (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2
   fname     IN   VARCHAR2);
```

Parameters

Table 16–2 BEGIN_INSTANTIATION Procedure Parameters

Parameter	Description
gname	Name of the replication 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 replication group.
fname	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 16–3 BEGIN_INSTANTIATION Procedure Exceptions

Exception	Description
`badargument`	`NULL` or empty string for replication 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 replication group.
`wrongstate`	Status of master definition site must be quiesced.
`dbms_repcat.missingrepgroup`	`gname` does not exist as a master group.
`dbms_repcat.missing_flavor`	If you receive this exception, contact Oracle Support Services.

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 to perform 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 materialized view) or with the procedures in the `DBMS_REPCAT_INSTANTIATE` package (used for instantiating a deployment template). See these respective packages for more information on their usage.

See Also: "Adding New Master Sites with Offline Instantiation Using Export/Import" on page 7-31 for information about adding a new master site to a master group by performing an offline instantiation of a master site

Syntax

```
DBMS_OFFLINE_OG.BEGIN_LOAD (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2);
```

Parameters

Table 16–4 BEGIN_LOAD Procedure Parameters

Parameter	Description
gname	Name of the replication group whose members you are importing.
new_site	The fully qualified database name of the new site at which you will be importing the replication group members.

Exceptions

Table 16–5 BEGIN_LOAD Procedure Exceptions

Exception	Description
badargument	NULL or empty string for replication group or new master site name.
wrongsite	This procedure must be called from the new master site.
unknownsite	Specified site is not recognized by replication group.
wrongstate	Status of the new master site must be quiesced.
dbms_repcat.missingrepgroup	gname does not exist as a master group.

END_INSTANTIATION Procedure

This procedure completes offline instantiation of a master group. You must call this procedure from the master definition site.

Note: This procedure is used to perform 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 materialized view) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

See Also: "Adding New Master Sites with Offline Instantiation Using Export/Import" on page 7-31 for information about adding a new master site to a master group by performing an offline instantiation of a master site

Syntax

```
DBMS_OFFLINE_OG.END_INSTANTIATION (
  gname     IN  VARCHAR2,
  new_site  IN  VARCHAR2);
```

Parameters

Table 16–6 END_INSTANTIATION Procedure Parameters

Parameter	Description
gname	Name of the replication 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 replication group.

Exceptions

Table 16–7 END_INSTANTIATION Procedure Exceptions

Exception	Description
badargument	NULL or empty string for replication 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 replication group.
wrongstate	Status of master definition site must be quiesced.
dbms_repcat.missingrepgroup	gname does not exist as a 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 to perform 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 materialized view) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

See Also: "Adding New Master Sites with Offline Instantiation Using Export/Import" on page 7-31 for information about adding a new master site to a master group by performing an offline instantiation of a master site

Syntax

```
DBMS_OFFLINE_OG.END_LOAD (
   gname     IN   VARCHAR2,
   new_site  IN   VARCHAR2
   fname     IN   VARCHAR2);
```

Parameters

Table 16–8 END_LOAD Procedure Parameters

Parameter	Description
gname	Name of the replication group whose members you have finished importing.
new_site	The fully qualified database name of the new site at which you have imported the replication group members.
fname	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 16–9 END_LOAD Procedure Exceptions

Exception	Description
badargument	NULL or empty string for replication group or new master site name.
wrongsite	This procedure must be called from the new master site.
unknownsite	Specified site is not recognized by replication group.
wrongstate	Status of the new master site must be quiesced.
dbms_repcat.missingrepgroup	gname does not exist as a master group.
dbms_repcat.flavor_noobject	If you receive this exception, contact Oracle Support Services.
dbms_repcat.flavor_contains	If you receive this exception, contact Oracle Support Services.

RESUME_SUBSET_OF_MASTERS Procedure

When you add a new master site to a master group by performing an offline instantiation of a master site, it may take some time to complete the offline instantiation process. This procedure resumes replication activity at all existing sites, except the new site, during offline instantiation of a master group. You typically execute this procedure after executing the `DBMS_OFFLINE_OG.BEGIN_INSTANTIATION` procedure. You must call this procedure from the master definition site.

Note: This procedure is used to perform 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 materialized view) or with the procedures in the `DBMS_REPCAT_INSTANTIATE` package (used for instantiating a deployment template). See these respective packages for more information on their usage.

See Also: "Adding New Master Sites with Offline Instantiation Using Export/Import" on page 7-31 for information about adding a new master site to a master group by performing an offline instantiation of a master site

Syntax

```
DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS (
   gname     IN  VARCHAR2,
   new_site  IN  VARCHAR2
   override  IN  BOOLEAN := false);
```

Parameters

Table 16–10 RESUME_SUBSET_OF_MASTERS Procedure Parameters

Parameter	Description
gname	Name of the replication 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 replication group.
override	If this is `true`, then any pending `RepCat` administrative 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` administrative request for `gname` at that master.

Exceptions

Table 16–11 RESUME_SUBSET_OF_MASTERS Procedure Exceptions

Exception	Description
badargument	`NULL` or empty string for replication 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 replication group.
wrongstate	Status of master definition site must be quiesced.
dbms_repcat.missingrepgroup	`gname` does not exist as a master group.

17

DBMS_OFFLINE_SNAPSHOT

The DBMS_OFFLINE_SNAPSHOT package contains public APIs for offline instantiation of materialized views.

This chapter discusses the following topics:

- Summary of DBMS_OFFLINE_SNAPSHOT Subprograms

Note: These procedure are used in performing an offline instantiation of a materialized view.

These procedures should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

Summary of DBMS_OFFLINE_SNAPSHOT Subprograms

Table 17–1 DBMS_OFFLINE_SNAPSHOT Package Subprograms

Subprogram	Description
BEGIN_LOAD Procedure on page 17-3	Prepares a materialized view site for import of a new materialized view as part of offline instantiation.
END_LOAD Procedure on page 17-5	Completes offline instantiation of a materialized view.

BEGIN_LOAD Procedure

This procedure prepares a materialized view site for import of a new materialized view as part of offline instantiation. You must call this procedure from the materialized view site for the new materialized view.

Note: This procedure is used to perform an offline instantiation of a materialized view.

These procedures should not be confused with the procedures in the `DBMS_OFFLINE_OG` package (used for performing an offline instantiation of a master table) or with the procedures in the `DBMS_REPCAT_INSTANTIATE` package (used for instantiating a deployment template). See these respective packages for more information on their usage.

See Also: "Performing an Offline Instantiation of a Materialized View Site Using Export/Import" on page 8-23 for information about adding a new materialized view site by performing an offline instantiation using Export/Import

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 17–2 BEGIN_LOAD Procedure Parameters

Parameter	Description
gname	Name of the replication group for the materialized view that you are creating using offline instantiation.
sname	Name of the schema for the new materialized view.
master_site	Fully qualified database name of the materialized view's master site.
snapshot_oname	Name of the temporary materialized view created at the master site.
storage_c	Storage options to use when creating the new materialized view at the materialized view 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 17–3 BEGIN_LOAD Procedure Exceptions

Exception	Description
badargument	`NULL` or empty string for replication group, schema, master site, or materialized view name.
dbms_repcat.missingrepgroup	`gname` does not exist as a replication group.
missingremotemview	Could not locate specified materialized view at specified master site.
dbms_repcat.missingschema	Specified schema does not exist.
mviewtabmismatch	Base table name of the materialized view at the master and materialized view do not match.

END_LOAD Procedure

This procedure completes offline instantiation of a materialized view. You must call this procedure from the materialized view site for the new materialized view.

Note: This procedure is used to perform an offline instantiation of a materialized view.

These procedures should not be confused with the procedures in the DBMS_OFFLINE_OG package (used for performing an offline instantiation of a master table) or with the procedures in the DBMS_REPCAT_INSTANTIATE package (used for instantiating a deployment template). See these respective packages for more information on their usage.

See Also: "Performing an Offline Instantiation of a Materialized View Site Using Export/Import" on page 8-23 for information about adding a new materialized view site by performing an offline instantiation using Export/Import

Syntax

```
DBMS_OFFLINE_SNAPSHOT.END_LOAD (
   gname           IN  VARCHAR2,
   sname           IN  VARCHAR2,
   snapshot_oname  IN  VARCHAR2);
```

Parameters

Table 17–4 END_LOAD Procedure Parameters

Parameter	Description
gname	Name of the replication group for the materialized view that you are creating using offline instantiation.
sname	Name of the schema for the new materialized view.
snapshot_oname	Name of the materialized view.

Exceptions

Table 17–5 END_LOAD Procedure Exceptions

Exception	Description
badargument	NULL or empty string for replication group, schema, or materialized view name.
dbms_repcat.missingrepgroup	gname does not exist as a replication group.
dbms_repcat.nonmview	This procedure must be called from the materialized view site.

18

DBMS_RECTIFIER_DIFF

The DBMS_RECTIFIER_DIFF package contains APIs used to detect and resolve data inconsistencies between two replicated sites.

This chapter discusses the following topics:

- Summary of DBMS_RECTIFIER_DIFF Subprograms

Summary of DBMS_RECTIFIER_DIFF Subprograms

Table 18–1 DBMS_RECTIFIER_DIFF Package Subprograms

Subprogram	Description
DIFFERENCES Procedure on page 18-3	Determines the differences between two tables.
RECTIFY Procedure on page 18-6	Resolves the differences between two tables.

DIFFERENCES Procedure

This procedure determines the differences between two tables. It accepts the storage table of a nested table.

Note: This procedure cannot be used on LOB columns, nor on columns based on user-defined types.

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 18–2 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.
where_clause	Only rows satisfying this clause 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 spaces before or after a comma. The default, NULL, indicates that all columns will be compared.
array_columns	A PL/SQL index-by 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 an existing 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 site that are missing from the table at reference_site.
missing_rows_oname2	Name of an existing table at missing_rows_site that stores information about the missing rows. This table has three columns: the R_ID column shows the rowid of the row in the missing_rows_oname1 table, the PRESENT column shows the name of the site where the row is present, and the ABSENT column shows name of the site from which the row is absent.
missing_rows_site	Name of the site where the missing_rows_oname1 and missing_rows_oname2 tables are located. The default, NULL, indicates that the tables are located at the current site.

Table 18–2 DIFFERENCES Procedure Parameters **(Page 2 of 2)**

Parameter	Description
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 parameter 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 18–3 DIFFERENCES Procedure Exceptions

Exception	Description
nosuchsite	Database site could not be found.
badnumber	The commit_rows parameter is 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 table.

RECTIFY Procedure

This procedure resolves the differences between two tables. It accepts the storage table of a nested table.

Note: This procedure cannot be used on LOB columns, nor on columns based on user-defined types.

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 18–4 RECTIFY Procedure Parameters

Parameter	Description
sname1	Name of the schema at reference_site.
oname1	Name of the table at reference_site.
reference_site	Name of the reference database site. The default, NULL, indicates the current site.
sname2	Name of the schema at comparison_site.
oname2	Name of the table at comparison_site.
comparison_site	Name of the comparison database site. The default, NULL, indicates the current site.
column_list	A comma-separated list of one or more column names being compared for the two tables. You must not have any spaces before or after a comma. The default, NULL, indicates that all columns will be compared.
array_columns	A PL/SQL index-by 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 18–5 RECTIFY Procedure Exceptions

Exception	Description
`nosuchsite`	Database site could not be found.
`badnumber`	The `commit_rows` parameter is 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.

19

DBMS_REFRESH

DBMS_REFRESH enables you to create groups of materialized views that can be refreshed together to a transactionally consistent point in time.

This chapter discusses the following topics:

Summary of DBMS_REFRESH Subprograms

Table 19–1 DBMS_REFRESH Package Subprograms

Subprogram	Description
ADD Procedure on page 19-3	Adds materialized views to a refresh group.
CHANGE Procedure on page 19-4	Changes the refresh interval for a refresh group.
DESTROY Procedure on page 19-6	Removes all of the materialized views from a refresh group and deletes the refresh group.
MAKE Procedure on page 19-7	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 19-10	Manually refreshes a refresh group.
SUBTRACT Procedure on page 19-10	Removes materialized views from a refresh group.

ADD Procedure

This procedure adds materialized views to a refresh group.

> **See Also:** Step 6, "Add objects to refresh group," on page 5-10 and *Oracle9i 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 19–2 ADD Procedures Parameters

Parameter	Description
`name`	Name of the refresh group to which you want to add members.
`list`	Comma-separated list of materialized views 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 index-by table of type `DBMS_UTILITY.UNCL_ARRAY`, where each element is the name of a materialized view. The first materialized view should be in position 1. The last position must be `NULL`.
`lax`	A materialized view can belong to only one refresh group at a time. If you are moving a materialized view from one group to another, then you must set the `lax` flag to `true` to succeed. Oracle then automatically removes the materialized view 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:** *Oracle9i Replication* for more information about refresh groups

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 19–3 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 materialized views 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 19–3 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 materialized views only. Set this parameter to `true` if you want to push changes from the materialized view to its associated master table or master materialized view before refreshing the materialized view. Otherwise, these changes may appear to be temporarily lost. By default, this flag remains unchanged.
`refresh_after_errors`	Used by updatable materialized views 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 materialized view's master table or master materialized view. 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 processes. 1 specifies parallel propagation using only one parallel process.
`heap_size`	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

DESTROY Procedure

This procedure removes all of the materialized views from a refresh group and delete the refresh group.

See Also: *Oracle9i Replication* for more information refresh groups

Syntax

```
DBMS_REFRESH.DESTROY (
   name   IN   VARCHAR2);
```

Parameters

Table 19–4 DESTROY Procedure Parameters

Parameter	Description
name	Name of the refresh group that you want to destroy.

MAKE Procedure

This procedure specifies the members of a refresh group and the time interval used to determine when the members of this group should be refreshed.

See Also: Step 4, "Create the refresh group," on page 5-7 and *Oracle9i Replication* for more information

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.

Parameters

Table 19–5 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 materialized views that you want to refresh. (Synonyms are not supported.) These materialized views can be located in different schemas and have different master tables or master materialized views. However, all of the listed materialized views must be in your current database.
tab	Instead of a comma separated list, you can supply a PL/SQL index-by table of names of materialized views that you want to refresh using the datatype DBMS_UTILITY.UNCL_ARRAY. If the table contains the names of *n* materialized views, then the first materialized view 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 materialized views 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 materialized views 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 enables you to create an empty refresh group.
lax	A materialized view can belong to only one refresh group at a time. If you are moving a materialized view from an existing group to a new refresh group, then you must set this to true to succeed. Oracle then automatically removes the materialized view 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 materialized views. The default, NULL, uses the default rollback segment.

Table 19–5 MAKE Procedure Parameters **(Page 2 of 2)**

Parameter	Description
`push_deferred_rpc`	Used by updatable materialized views only. Use the default value, `true`, if you want to push changes from the materialized view to its associated master table or master materialized view before refreshing the materialized view. Otherwise, these changes may appear to be temporarily lost.
`refresh_after_errors`	Used by updatable materialized views 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 materialized view's master table or master materialized view.
`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 processes. 1 specifies parallel propagation using only one parallel process.
`heap_size`	Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

REFRESH Procedure

This procedure manually refreshes a refresh group.

> **See Also:** *Oracle9i Replication* for more information about refresh groups

Syntax

```
DBMS_REFRESH.REFRESH (
   name   IN    VARCHAR2);
```

Parameters

Table 19–6 REFRESH Procedure Parameters

Parameter	Description
name	Name of the refresh group that you want to refresh manually.

SUBTRACT Procedure

This procedure removes materialized views from a refresh group.

> **See Also:** *Oracle9i Replication* for more information about refresh groups

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 19–7 SUBTRACT Procedure Parameters

Parameter	Description
`name`	Name of the refresh group from which you want to remove members.
`list`	Comma-separated list of materialized views that you want to remove from the refresh group. (Synonyms are not supported.) These materialized views can be located in different schemas and have different master tables or master materialized views. However, all of the listed materialized views must be in your current database.
`tab`	Instead of a comma-separated list, you can supply a PL/SQL index-by table of names of materialized views that you want to refresh using the datatype `DBMS_UTILITY.UNCL_ARRAY`. If the table contains the names of *n* materialized views, then the first materialized view 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 materialized view you are attempting to remove is not a member of the refresh group.

20

DBMS_REPCAT

DBMS_REPCAT provides routines to administer and update the replication catalog and environment.

This chapter discusses the following topics:

- Summary of DBMS_REPCAT Subprograms

Summary of DBMS_REPCAT Subprograms

Table 20–1 DBMS_REPCAT Package Subprograms **(Page 1 of 6)**

Subprogram	Description
ADD_GROUPED_COLUMN Procedure on page 20-8	Adds members to an existing column group.
ADD_MASTER_DATABASE Procedure on page 20-9	Adds another master site to your replication environment.
ADD_NEW_MASTERS Procedure on page 20-11	Adds the master sites in the DBA_REPSITES_NEW data dictionary view to the replication catalog at all available master sites.
ADD_PRIORITY_datatype Procedure on page 20-17	Adds a member to a priority group.
ADD_SITE_PRIORITY_SITE Procedure on page 20-19	Adds a new site to a site priority group.
ADD_conflicttype_RESOLUTION Procedure on page 20-20	Designates a method for resolving an update, delete, or uniqueness conflict.
ALTER_CATCHUP_PARAMETERS Procedure on page 20-26	Alters the values for parameters stored in the DBA_REPEXTENSIONS data dictionary view.
ALTER_MASTER_PROPAGATION Procedure on page 20-28	Alters the propagation method for a specified replication group at a specified master site.
ALTER_MASTER_REPOBJECT Procedure on page 20-29	Alters an object in your replication environment.
ALTER_MVIEW_PROPAGATION Procedure on page 20-32	Alters the propagation method for a specified replication group at the current materialized view site.
ALTER_PRIORITY Procedure on page 20-34	Alters the priority level associated with a specified priority group member.
ALTER_PRIORITY_datatype Procedure on page 20-35	Alters the value of a member in a priority group.
ALTER_SITE_PRIORITY Procedure on page 20-37	Alters the priority level associated with a specified site.
ALTER_SITE_PRIORITY_SITE Procedure on page 20-39	Alters the site associated with a specified priority level.

Table 20–1 DBMS_REPCAT Package Subprograms (Page 2 of 6)

Subprogram	Description
CANCEL_STATISTICS Procedure on page 20-40	Stops collecting statistics about the successful resolution of update, uniqueness, and delete conflicts for a table.
COMMENT_ON_COLUMN_GROUP Procedure on page 20-41	Updates the comment field in the ALL_REPCOLUMN_GROUP view for a column group.
COMMENT_ON_conflicttype_RESOLUTION Procedure on page 20-48	Updates the SCHEMA_COMMENT field in the ALL_REPGROUP view for a materialized view site.
COMMENT_ON_PRIORITY_GROUP/COMMENT_ON_SITE_PRIORITY Procedures on page 20-43	Updates the comment field in the ALL_REPPRIORITY_GROUP view for a (site) priority group.
COMMENT_ON_REPGROUP Procedure on page 20-44	Updates the comment field in the ALL_REPGROUP view for a master group.
COMMENT_ON_REPOBJECT Procedure on page 20-45	Updates the comment field in the ALL_REPOBJECT view for a replicated object.
COMMENT_ON_REPSITES Procedure on page 20-46	Updates the comment field in the ALL_REPSITE view for a replicated site.
COMMENT_ON_conflicttype_RESOLUTION Procedure on page 20-48	Updates the comment field in the ALL_REPRESOLUTION view for a conflict resolution routine.
COMPARE_OLD_VALUES Procedure on page 20-50	Specifies whether to compare old column values at each master site for each nonkey column of a replicated table for updates and deletes.
CREATE_MASTER_REPGROUP Procedure on page 20-52	Creates a new, empty, quiesced master group.
CREATE_MASTER_REPOBJECT Procedure on page 20-53	Specifies that an object is a replicated object.
CREATE_MVIEW_REPGROUP Procedure on page 20-57	Creates a new, empty materialized view group in your local database.
CREATE_MVIEW_REPOBJECT Procedure on page 20-58	Adds a replicated object to a materialized view group.
DEFINE_COLUMN_GROUP Procedure on page 20-61	Creates an empty column group.
DEFINE_PRIORITY_GROUP Procedure on page 20-62	Creates a new priority group for a master group.

Table 20–1 DBMS_REPCAT Package Subprograms **(Page 3 of 6)**

Subprogram	Description
DEFINE_SITE_PRIORITY Procedure on page 20-64	Creates a new site priority group for a master group.
DO_DEFERRED_REPCAT_ADMIN Procedure on page 20-65	Executes the local outstanding deferred administrative procedures for the specified master group at the current master site, or for all master sites.
DROP_COLUMN_GROUP Procedure on page 20-66	Drops a column group.
DROP_GROUPED_COLUMN Procedure on page 20-67	Removes members from a column group.
DROP_MASTER_REPGROUP Procedure on page 20-68	Drops a master group from your current site.
DROP_MASTER_REPOBJECT Procedure on page 20-69	Drops a replicated object from a master group.
DROP_PRIORITY Procedure on page 20-73	Drops a replicated object from a master group.
DROP_MVIEW_REPGROUP Procedure on page 20-71	Drops a materialized view site from your replication environment.
DROP_MVIEW_REPOBJECT Procedure on page 20-72	Drops a replicated object from a materialized view site.
DROP_PRIORITY Procedure on page 20-73	Drops a member of a priority group by priority level.
DROP_PRIORITY_GROUP Procedure on page 20-74	Drops a priority group for a specified master group.
DROP_PRIORITY_datatype Procedure on page 20-75	Drops a member of a priority group by value.
DROP_SITE_PRIORITY Procedure on page 20-76	Drops a site priority group for a specified master group.
DROP_SITE_PRIORITY_SITE Procedure on page 20-77	Drops a specified site, by name, from a site priority group.
DROP_conflicttype_RESOLUTION Procedure on page 20-78	Drops an update, delete, or uniqueness conflict resolution method.
EXECUTE_DDL Procedure on page 20-80	Supplies DDL that you want to have executed at each master site.

Table 20–1 DBMS_REPCAT Package Subprograms **(Page 4 of 6)**

Subprogram	Description
GENERATE_MVIEW_SUPPORT Procedure on page 20-82	Activates triggers and generate packages needed to support the replication of updatable materialized views or procedural replication.
GENERATE_REPLICATION_SUPPORT Procedure on page 20-83	Generates the triggers, packages, and procedures needed to support replication for a specified object.
MAKE_COLUMN_GROUP Procedure on page 20-85	Creates a new column group with one or more members.
PREPARE_INSTANTIATED_MASTER Procedure on page 20-87	Changes the global name of the database you are adding to a master group.
PURGE_MASTER_LOG Procedure on page 20-88	Removes local messages in the DBA_REPCATLOG associated with a specified identification number, source, or master group.
PURGE_STATISTICS Procedure on page 89	Removes information from the ALL_ REPRESOLUTION_STATISTICS view.
REFRESH_MVIEW_REPGROUP Procedure on page 20-90	Refreshes a materialized view group with the most recent data from its associated master site or master materialized view site.
REGISTER_MVIEW_REPGROUP Procedure on page 20-92	Facilitates the administration of materialized views at their respective master sites or master materialized view sites by inserting, modifying, or deleting from DBA_REGISTERED_MVIEW_ GROUPS.
REGISTER_STATISTICS Procedure on page 20-94	Collects information about the successful resolution of update, delete, and uniqueness conflicts for a table.
RELOCATE_MASTERDEF Procedure on page 95	Changes your master definition site to another master site in your replication environment.
REMOVE_MASTER_DATABASES Procedure on page 20-97	Removes one or more master databases from a replication environment.
RENAME_SHADOW_COLUMN_GROUP Procedure on page 20-98	Renames the shadow column group of a replicated table to make it a named column group.

Table 20–1 DBMS_REPCAT Package Subprograms **(Page 5 of 6)**

Subprogram	Description
REPCAT_IMPORT_CHECK Procedure on page 20-99	Ensures that the objects in the 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 20-100	Resumes normal replication activity after quiescing a replication environment.
RESUME_PROPAGATION_TO_MDEF Procedure on page 20-101	Indicates that export is effectively finished and propagation for both extended and unaffected replication groups existing at master sites can be enabled.
SEND_OLD_VALUES Procedure on page 20-102	Specifies whether to send old column values for each nonkey column of a replicated table for updates and deletes.
SET_COLUMNS Procedure on page 20-105	Specifies use of 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.
SPECIFY_NEW_MASTERS Procedure on page 20-107	Specifies the master sites you intend to add to an existing replication group without quiescing the group.
SUSPEND_MASTER_ACTIVITY Procedure on page 20-109	Suspends replication activity for a master group.
SWITCH_MVIEW_MASTER Procedure on page 20-110	Changes the master site of a materialized view group to another master site.
UNDO_ADD_NEW_MASTERS_REQUEST Procedure on page 20-111	Undoes all of the changes made by the SPECIFY_NEW_MASTERS and ADD_NEW_MASTERS procedures for a specified extension_id.
UNREGISTER_MVIEW_REPGROUP Procedure on page 20-113	Facilitates the administration of materialized views at their respective master sites and master materialized view sites by inserting, modifying, or deleting from DBA_REGISTERED_MVIEW_GROUPS.
VALIDATE Function on page 20-114	Validates the correctness of key conditions of a multimaster replication environment.

Table 20–1 DBMS_REPCAT Package Subprograms **(Page 6 of 6)**

Subprogram	Description
`WAIT_MASTER_LOG Procedure on page 20-117`	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 20–2 ADD_GROUPED_COLUMN Procedure Parameters

Parameter	Description
`sname`	Schema in which the replicated table is located.
`oname`	Name of the replicated table with which the column group is associated. The table can be the storage table of a nested table.
`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 index-by table of column names. The PL/SQL index-by table must be of type `DBMS_REPCAT.VARCHAR2`. Use the single value `'*'` to create a column group that contains all of the columns in your table. You can specify column objects, but you cannot specify attributes of column objects. If the table is an object, then you can specify `SYS_NC_OID$` to add the object identifier column to the column group. This column tracks the object identifier of each row object. If the table is a storage table of a nested table, then you can specify `NESTED_TABLE_ID` to add the column that tracks the identifier for each row of the nested table.

Table 20–3 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	Replication group to which the specified table belongs is not quiesced.

ADD_MASTER_DATABASE Procedure

This procedure adds another master site to your replication 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 20–4 ADD_MASTER_DATABASE Procedure Parameters

Parameter	Description
gname	Name of the replication group being replicated. This replication 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.
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 comment 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. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 20–5 ADD_MASTER_DATABASE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Replication has not been suspended for the master group.
missingrepgroup	Replication 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_NEW_MASTERS Procedure

This procedure adds the master sites in the DBA_REPSITES_NEW data dictionary view to the master groups specified when the SPECIFY_NEW_MASTERS procedure was run. Information about these new master sites are added to the replication catalog at all available master sites.

All master sites instantiated with object-level export/import must be accessible at this time. Their new replication groups are added in the quiesced state. Master sites instantiated through full database export/import or through changed-based recovery do not need to be accessible.

Run this procedure after you run the SPECIFY_NEW_MASTERS procedure.

Caution: After running this procedure, do not disable or enable propagation of the deferred transactions queue until after the new master sites are added. The DBA_REPEXTENSIONS data dictionary view must be clear before you disable or enable propagation. You can use the Replication Management tool or the SET_DISABLED procedure in the DBMS_DEFER_SYS package to disable or enable propagation.

See Also:

- "SPECIFY_NEW_MASTERS Procedure" on page 20-107
- "Adding New Master Sites" on page 7-4 for more information about adding master sites to a master group

Syntax

```
DBMS_REPCAT.ADD_NEW_MASTERS (
   export_required               IN    BOOLEAN,
   { available_master_list       IN    VARCHAR2,
    | available_master_table      IN    DBMS_UTILITY.DBLINK_ARRAY,}
   masterdef_flashback_scn       OUT   NUMBER,
   extension_id                  OUT   RAW,
   break_trans_to_masterdef      IN    BOOLEAN   := false,
   break_trans_to_new_masters    IN    BOOLEAN   := false,
   percentage_for_catchup_mdef   IN    BINARY_INTEGER  := 100,
   cycle_seconds_mdef            IN    BINARY_INTEGER  := 60,
   percentage_for_catchup_new    IN    BINARY_INTEGER  := 100,
   cycle_seconds_new             IN    BINARY_INTEGER  := 60);
```

Note: This procedure is overloaded. The `available_master_list` and `available_master_table` parameters are mutually exclusive.

Parameters

Table 20–6 ADD_NEW_MASTERS Procedure Parameters **(Page 1 of 3)**

Parameter	Description
export_required	Set to true if either object-level or full database export is required for at least one of the new master sites. Set to false if you are using change-based recovery for all of the new master sites.
available_master_list	A comma-separated list of the new master sites to be instantiated using object-level export/import. The sites listed must match the sites specified in the SPECIFY_NEW_MASTERS procedure. List only the new master sites, not the existing master sites. Do not put any spaces between site names. Specify NULL if all masters will be instantiated using full database export/import or change-based recovery.
available_master_table	A table that lists the new master sites to be instantiated using object-level export/import. The sites in the table must match the sites specified in the SPECIFY_NEW_MASTERS procedure. Do not specify masters that will be instantiated using full database export/import or change-based recovery. In the table that lists the master sites to be instantiated using object-level export/import, list only the new master sites for the master groups being extended. Do not list the existing master sites in the master groups being extended. The first master site should be at position 1, the second at position 2, and so on.
masterdef_flashback_scn	This OUT parameter returns a system change number (SCN) that must be used during export or change-based recovery. Use the value returned by this parameter for the FLASHBACK_SCN export parameter when you perform the export. You can find the flashback_scn value by querying the DBA_REPEXTENSIONS data dictionary view.
extension_id	This OUT parameter returns an identifier for the current pending request to add master databases without quiesce. You can find the extension_id by querying the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.

Table 20–6 ADD_NEW_MASTERS Procedure Parameters **(Page 2 of 3)**

Parameter	Description
break_trans_to_masterdef	This parameter is meaningful only if export_ required is set to true. If break_trans_to_masterdef is set to true, then existing masters may continue to propagate their deferred transactions to the master definition site for replication groups that are not adding master sites. Deferred transactions for replication groups that are adding master sites cannot be propagated until the export completes. Each deferred transaction is composed of one or more remote procedure calls (RPCs). If set to false and a transaction occurs that references objects in both unaffected master groups and master groups that are being extended, then the transaction may be split into two parts and sent to a destination in two separate transactions at different times. Such transactions are called split-transactions. If split-transactions are possible, then you must disable integrity constraints that may be violated by this behavior until the new master sites are added. If break_trans_to_masterdef is set to false, then existing masters cannot propagate their deferred transactions to the master definition site.
break_trans_to_new_masters	If break_trans_to_new_masters is set to true, then existing master sites may continue to propagate deferred transactions to the new master sites for replication groups that are not adding master sites. Each deferred transaction is composed of one or more remote procedure calls (RPCs). If set to true and a transaction occurs that references objects in both unaffected master groups and master groups that are being extended, then the transaction may be split into two parts and sent to a destination in two separate transactions at different times. Such transactions are called split-transactions. If split-transactions are possible, then you must disable integrity constraints that may be violated by this behavior until the new master sites are added. If break_trans_to_new_masters is set to false, then propagation of deferred transaction queues to the new masters is disabled.

Table 20–6 ADD_NEW_MASTERS Procedure Parameters **(Page 3 of 3)**

Parameter	Description
`percentage_for_catchup_mdef`	This parameter is meaningful only if `export_required` and `break_trans_to_masterdef` are both set to `true`. The percentage of propagation resources that should be used for catching up propagation to the master definition site. Must be a multiple of 10 and must be between 0 and 100.
`cycle_seconds_mdef`	This parameter is meaningful when `percentage_for_catchup_mdef` is both meaningful and set to a value between 10 and 90, inclusive. In this case, propagation to the masterdef alternates between replication groups that are not being extended and replication groups that are being extended, with one push to each during each cycle. This parameter indicates the length of the cycle in seconds.
`percentage_for_catchup_new`	This parameter is meaningful only if `break_trans_to_new_masters` is set to `true`. The percentage of propagation resources that should be used for catching up propagation to new master sites. Must be a multiple of 10 and must be between 0 and 100.
`cycle_seconds_new`	This parameter is meaningful when `percentage_for_catchup_new` is both meaningful and set to a value between 10 and 90, inclusive. In this case, propagation to a new master alternates between replication groups that are not being extended and replication groups that are being extended, with one push to each during each cycle. This parameter indicates the length of the cycle in seconds.

Exceptions

Table 20–7 ADD_NEW_MASTERS Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
typefailure	The parameter value specified for one of the parameters is not appropriate.
novalidextreq	No valid extension request. The `extension_id` is not valid.
nonewsites	No new master sites to be added for the specified extension request.
notanewsite	Not a new site for extension request. A site was specified that was not specified when you ran the `SPECIFY_NEW_MASTERS` procedure.
dbnotcompatible	Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.

Usage Notes

For a new master site to be instantiated using change-based recovery or full database export/import, the following conditions apply:

- The new master sites cannot have any existing replication groups.
- The master definition site cannot have any materialized view groups.
- The master definition site must be the same for all of the master groups. If one or more of these master groups have a different master definition site, then do not use change-based recovery or full database export/import. Use object-level export/import instead.
- The new master site must include all of the replication groups in the master definition site when the extension process is complete. That is, you cannot add a subset of the master groups at the master definition site to the new master site; all of the groups must be added.

> **Note:** To use change-based recovery, the existing master site and the new master site must be running under the same operating system, although the release of the operating system can differ.

For object-level export/import, before importing ensure that all the requests in the DBA_REPCATLOG data dictionary view for the extended groups have been processed without any error.

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:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–8 ADD_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	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 20–9 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 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 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:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–10 ADD_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	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 20–11 ADD_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
`nonmasterdef`	Invocation site is not the master definition site.
`missingrepgroup`	Specified 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`	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 20–12 ADD_conflicttype_RESOLUTION Procedures

Conflict Type	Procedure Name
`update`	`ADD_UPDATE_RESOLUTION`
`uniqueness`	`ADD_UNIQUE_RESOLUTION`
`delete`	`ADD_DELETE_RESOLUTION`

See Also: Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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
                                | DBMS_UTILITY.LNAME_ARRAY,
   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
                               | DBMS_UTILITY.LNAME_ARRAY,
   function_name         IN   VARCHAR2     := NULL,
   comment               IN   VARCHAR2     := NULL);
```

Parameters

Table 20–13 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 to which you are adding a conflict resolution routine. The table can be the storage table of a nested table.
column_group	Name of the column group to 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 method as the `function_name` parameter. The standard methods supported in this release for update conflicts are: ■ `minimum` ■ `maximum` ■ `latest timestamp` ■ `earliest timestamp` ■ `additive`, `average` ■ `priority group` ■ `site priority` ■ `overwrite` ■ `discard` The standard methods supported in this release for uniqueness conflicts are: `append site name`, `append sequence`, and `discard`. There are no built-in (Oracle supplied) methods for delete conflicts.

Table 20–13 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 parameter. If your are using a user function, then you can resolve the conflict using any number of columns. For update or unique conflicts, this parameter accepts either a comma-separated list of column names, or a PL/SQL index-by table of type DBMS_REPCAT.VARCHAR2 or DBMS_UTILITY.LNAME_ARRAY. Use DBMS_UTILITY.LNAME_ARRAY if any column name is greater than or equal to 30 bytes, which may occur when you specify the attributes of column objects. For delete conflicts, this parameter accepts either a comma-separated list of column names or a PL/SQL index-by 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. LOB columns cannot be specified for this parameter. **See Also:** "Usage Notes" on page 20-25 if you are using column objects
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 Chapter 6, "Configure Conflict Resolution" and *Oracle9i Replication* for more information. If you are using a different method, you can use the default value for this parameter, NULL. This parameter 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 parameter, NULL.
comment	This user comment is added to the DBA_REPRESOLUTION view.

Exceptions

Table 20–14 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` parameter 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` parameter 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`	Replication 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.

Usage Notes

If you are using column objects, then whether you can specify the attributes of the column objects for the parameter_column_name parameter depends on whether the conflict resolution method is built-in (Oracle supplied) or user-created:

- If you are using a built-in conflict resolution method, then you can specify attributes of objects for this parameter. For example, if a column object named cust_address has street_address as an attribute, then you can specify cust_address.street_address for this parameter.
- If you are using a built-in conflict resolution method, the following types of columns cannot be specified for this parameter: LOB attribute of a column object, collection or collection attribute of a column object, REF, or an entire column object.
- If you are using a user-created conflict resolution method, then you must specify an entire column object. You cannot specify the attributes of a column object. For example, if a column object named cust_address has street_address as an attribute (among other attributes), then you can specify only cust_address for this parameter.

ALTER_CATCHUP_PARAMETERS Procedure

This procedure alters the values for the following parameters stored in the DBA_REPEXTENSIONS data dictionary view:

- percentage_for_catchup_mdef
- cycle_seconds_mdef
- percentage_for_catchup_new
- cycle_seconds_new

These parameters were originally set by the ADD_NEW_MASTERS procedure. The new values you specify for these parameters are used during the remaining steps in the process of adding new master sites to a master group. These changes are only to the site at which it is executed. Therefore, it must be executed at each master site, including the master definition site, if you want to alter parameters at all sites.

See Also:

- "ADD_NEW_MASTERS Procedure" on page 20-11
- "Adding New Master Sites" on page 7-4 for more information about adding master sites to a master group

Syntax

```
DBMS_REPCAT.ALTER_CATCHUP_PARAMETERS (
   extension_id                  IN    RAW,
   percentage_for_catchup_mdef   IN    BINARY_INTEGER  := NULL,
   cycle_seconds_mdef            IN    BINARY_INTEGER  := NULL,
   percentage_for_catchup_new    IN    BINARY_INTEGER  := NULL,
   cycle_seconds_new             IN    BINARY_INTEGER  := NULL);
```

Parameters

Table 20–15 ALTER_CATCHUP_PARAMETERS Procedure Parameters

Parameter	Description
extension_id	The identifier for the current pending request to add master database without quiesce. You can find the extension_id by querying the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.
percentage_for_catchup_mdef	The percentage of propagation resources that should be used for catching up propagation to the master definition site. Must be a multiple of 10 and must be between 0 and 100.
cycle_seconds_mdef	This parameter is meaningful when percentage_for_catchup_mdef is both meaningful and set to a value between 10 and 90, inclusive. In this case, propagation to the masterdef alternates between replication groups that are not being extended and replication groups that are being extended, with one push to each during each cycle. This parameter indicates the length of the cycle in seconds.
percentage_for_catchup_new	The percentage of propagation resources that should be used for catching up propagation to new master sites. Must be a multiple of 10 and must be between 0 and 100.
cycle_seconds_new	This parameter is meaningful when percentage_for_catchup_new is both meaningful and set to a value between 10 and 90, inclusive. In this case, propagation to a new master alternates between replication groups that are not being extended and replication groups that are being extended, with one push to each during each cycle. This parameter indicates the length of the cycle in seconds.

Exceptions

Table 20–16 ALTER_CATCHUP_PARAMETERS Procedure Exceptions

Exception	Description
typefailure	The parameter value specified for one of the parameters is not appropriate.
dbnotcompatible	Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.

ALTER_MASTER_PROPAGATION Procedure

This procedure alters the propagation method for a specified replication group at a specified master site. This replication 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 20–17 ALTER_MASTER_PROPAGATION Procedure Parameters

Parameter	Description
gname	Name of the replication 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 the propagation method. If NULL, then all masters except the master site being altered are used by default.
dblink_table	A PL/SQL index-by 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 20–18 ALTER_MASTER_PROPAGATION Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Invocation 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 replication environment. You must call this procedure from the master definition site.

This procedure requires that you quiesce the master group of the object if either of the following conditions is true:

- You are altering a table in a multimaster replication environment.
- You are altering a table with the `safe_table_change` parameter set to `false` in a single master replication environment.

You can use this procedure to alter nontable objects without quiescing the master group.

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
   safe_table_change   IN   BOOLEAN       := false);
```

Parameters

Table 20–19 ALTER_MASTER_REPOBJECT Procedure Parameters **(Page 1 of 2)**

Parameter	Description	
sname	Schema containing the object that you want to alter.	
oname	Name of the object that you want to alter. The object cannot be a storage table for a nested table.	
type	Type of the object that you are altering. The following types are supported:	
	FUNCTION	SYNONYM
	INDEX	TABLE
	INDEXTYPE	TRIGGER
	OPERATOR	TYPE
	PACKAGE	TYPE BODY
	PACKAGE BODY	VIEW
	PROCEDURE	
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.	

Table 20–19 ALTER_MASTER_REPOBJECT Procedure Parameters **(Page 2 of 2)**

Parameter	Description
`safe_table_change`	Specify `true` if the change to a table is safe. Specify `false` if the change to a table is unsafe. You can make safe changes to a master table in a single master replication environment without quiescing the master group that contains the table. To make unsafe changes, you must quiesce the master group. Only specify this parameter for tables in single master replication environments. This parameter is ignored in multimaster replication environments and when the object specified is not a table. In multimaster replication environments, you must quiesce the master group to run the `ALTER_MASTER_REPOBJECT` procedure on a table. The following are safe changes: ■ Changing storage and extent information ■ Making existing columns larger. For example, changing a `VARCHAR2(20)` column to a `VARCHAR2(50)` column. ■ Adding non primary key constraints ■ Altering non primary key constraints ■ Enabling and disabling non primary key constraints The following are unsafe changes: ■ Changing the primary key by adding or deleting columns in the key ■ Adding or deleting columns ■ Making existing columns smaller. For example, changing a `VARCHAR2(50)` column to a `VARCHAR2(20)` column. ■ Disabling a primary key constraint ■ Changing the datatype of an existing column ■ Dropping an existing column If you are unsure whether a change is safe or unsafe, then quiesce the master group before you run the `ALTER_MASTER_REPOBJECT` procedure.

Exceptions

Table 20–20 ALTER_MASTER_REPOBJECT Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notquiesced	Associated replication 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_MVIEW_PROPAGATION Procedure

This procedure alters the propagation method for a specified replication group at the current materialized view site. This procedure pushes the deferred transaction queue at the materialized view site, locks the materialized view base tables, and regenerates any triggers and their associated packages. You must call this procedure from the materialized view site.

Syntax

```
DBMS_REPCAT.ALTER_MVIEW_PROPAGATION (
   gname                IN  VARCHAR2,
   propagation_mode     IN  VARCHAR2,
   comment              IN  VARCHAR2   := ''
   gowner               IN  VARCHAR2   := 'PUBLIC');
```

Parameters

Table 20–21 ALTER_MVIEW_PROPAGATION Procedure Parameters

Parameter	Description
gname	Name of the replication group for which to alter the propagation method.
propagation_mode	Manner in which changes from the current materialized view site are propagated to its associated master site or master materialized view site. Appropriate values are synchronous and asynchronous.
comment	This comment is added to the DBA_REPPROP view.
gowner	Owner of the materialized view group.

Exceptions

Table 20–22 ALTER_MVIEW_PROPAGATION Procedure Exceptions

Exception	Description
missingrepgroup	Specified replication group does not exist.
typefailure	Propagation mode was specified incorrectly.
nonmview	Current site is not a materialized view site for the specified replication group.
commfailure	Cannot contact master site or master materialized view site.
notcompat	Compatibility mode must be 7.3.0.0 or greater.
failaltermviewrop	Materialized view group propagation can be altered only when there are no other materialized view groups with the same master site or master materialized view site sharing the materialized view site.

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:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–23   ALTER_PRIORITY Procedure Parameters

Parameter	Description
gname	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 20–24 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 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 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:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–25 ALTER_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	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 20–26 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 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 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:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–27 ALTER_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	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 20–28 ALTER_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified 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	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: Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–29   ALTER_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	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 20–30   ALTER_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
missingpriority	Specified site priority group does not exist.
missingvalue	Old site is not a group member.
notquiesced	Master group is not quiesced

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 20–31 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 20–32 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 20–33 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 20–34 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_MVIEW_REPSITES Procedure

This procedure updates the SCHEMA_COMMENT field in the DBA_REPGROUP data dictionary view for the specified materialized view group. The group name must be registered locally as a replicated materialized view group. This procedure must be executed at the materialized view site.

Syntax

```
DBMS_REPCAT.COMMENT_ON_MVIEW_REPSITES (
   gowner    IN   VARCHAR2,
   gname     IN   VARCHAR2,
   comment   IN   VARCHAR2);
```

Parameters

Table 20–35 COMMENT_ON_MVIEW_REPSITES Procedure Parameters

Parameter	Description
gowner	Owner of the materialized view group.
gname	Name of the materialized view group.
comment	Updated comment to include in the SCHEMA_COMMENT field of the DBA_REPGROUP view.

Table 20–36 COMMENT_ON_MVIEW_REPSITES Procedure Exceptions

Parameter	Description
missingrepgroup	The materialized view group does not exist.
nonmview	The connected site is not a materialized view site.

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 20–37 COMMENT_ON_PRIORITY_GROUP and COMMENT_ON_SITE_PRIORITY Parameters

Parameter	Description
gname	Name of the 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 20–38 COMMENT_ON_PRIORITY_GROUP and COMMENT_ON_SITE_PRIORITY Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified 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 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 20–39 COMMENT_ON_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replication group that you want to comment on.
comment	Updated comment to include in the SCHEMA_COMMENT field of the DBA_REPGROUP view.

Exceptions

Table 20–40 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 20–41 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. The object cannot be a storage table for a nested table.	
type	Type of the object. The following types are supported:	
	FUNCTION	SYNONYM
	INDEX	TABLE
	INDEXTYPE	TRIGGER
	OPERATOR	TYPE
	PACKAGE	TYPE BODY
	PACKAGE BODY	VIEW
	PROCEDURE	
comment	Text of the updated comment that you want to include in the OBJECT_COMMENT field of the DBA_REPOBJECT view.	

Exceptions

Table 20–42 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 replication group is a master group, then this procedure updates the MASTER_COMMENT field in the DBA_REPSITES view for a master site. If the replication group is a materialized view group, this procedure updates the SCHEMA_COMMENT field in the DBA_REPGROUP view for a materialized view site.

This procedure can be executed at either a master site or a materialized view site. If you execute this procedure on a a materialized view site, then the materialized view group owner must be PUBLIC.

> **See Also:** "COMMENT_ON_conflicttype_RESOLUTION Procedure" on page 20-48 for instructions on placing a comment in the SCHEMA_COMMENT field of the DBA_REPGROUP view for a materialized view site if the materialized view group owner is not PUBLIC

Syntax

```
DBMS_REPCAT.COMMENT_ON_REPSITES (
   gname        IN   VARCHAR2,
   [ master     IN   VARCHAR,]
   comment      IN   VARCHAR2);
```

Parameters

Table 20–43 COMMENT_ON_REPSITES Procedure Parameters

Parameter	Description
gname	Name of the replication group. This avoids confusion if a database is a master site in more than one replication environment.
master	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, then this parameter is required. To update comments at a materialized view site, omit this parameter. This parameter is optional.
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, then this procedure updates the MASTER_COMMENT field of the DBA_REPSITES view. If the site is a materialized view site, then this procedure updates the SCHEMA_COMMENT field of the DBA_REPGROUP view.

Exceptions

Table 20–44 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	Replication 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_*conflicttype*_RESOLUTION Procedure

This procedure updates the RESOLUTION_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 20–45 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 20–46 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 20–47 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 specifies whether to compare old column values during propagation of deferred transactions at each master site for each nonkey 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 sites and materialized view sites by invoking DBMS_REPCAT.COMPARE_OLD_VALUES at the master definition site.

When you use user-defined types, you can specify leaf attributes of a column object, or you can specify an entire column object. For example, if a column object named cust_address has street_address as an attribute, then you can specify cust_address.street_address for the column_list parameter or as part of the column_table parameter, or you can specify only cust_address.

When performing equality comparisons for conflict detection, Oracle treats objects as equal only if one of the following conditions is true:

- Both objects are atomically NULL (the entire object is NULL)
- All of the corresponding attributes are equal in the objects

Given these conditions, if one object is atomically NULL while the other is not, then Oracle does not consider the objects to be equal. Oracle does not consider MAP and ORDER methods when performing equality comparisons.

Syntax

```
DBMS_REPCAT.COMPARE_OLD_VALUES(
   sname           IN  VARCHAR2,
   oname           IN  VARCHAR2,
   { column_list   IN  VARCHAR2,
   | column_table  IN  DBMS_UTILITY.VARCHAR2s | DBMS_UTILITY.LNAME_ARRAY,}
   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 20–48 COMPARE_OLD_VALUES Procedure Parameters

Parameter	Description
`sname`	Schema in which the table is located.
`oname`	Name of the replicated table. The table can be the storage table of a nested table.
`column_list`	A comma-separated list of the columns in the table. There must be no spaces between entries.
`column_table`	Instead of a list, you can use a PL/SQL index-by table of type `DBMS_REPCAT.VARCHAR2` or `DBMS_UTILITY.LNAME_ARRAY` to contain the column names. The first column name should be at position 1, the second at position 2, and so on. Use `DBMS_UTILITY.LNAME_ARRAY` if any column name is greater than or equal to 30 bytes, which may occur when you specify the attributes of column objects.
`operation`	Possible values are: `update`, `delete`, or the asterisk wildcard `'*'`, which means update and delete.
`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 materialized view site the next time replication support is generated at that site with `min_communication true`.

Note: The `operation` parameter enables you to decide whether or not to compare old values for nonkey columns when rows are deleted or updated. If you do not compare the old value, then Oracle 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 *Oracle9i Replication* for more information about reduced data propagation using the `COMPARE_OLD_VALUES` procedure before changing the default behavior of Oracle.

Exceptions

Table 20–49 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 waiting for row-level replication information.
missingcolumn	At least one column is not in the table.
notquiesced	Master group has not been quiesced.
typefailure	An illegal operation is specified.
keysendcomp	A specified column is a key column in a table.
dbnotcompatible	Feature is incompatible with database version. Typically, this exception arises when you are trying to compare the attributes of column objects. In this case, all databases must be at 9.0.0 or higher compatibility level.

CREATE_MASTER_REPGROUP Procedure

This procedure creates a new, empty, quiesced master group.

Syntax

```
DBMS_REPCAT.CREATE_MASTER_REPGROUP (
   gname            IN   VARCHAR2,
   group_comment    IN   VARCHAR2     := '',
   master_comment   IN   VARCHAR2     := ''),
   qualifier        IN   VARCHAR2     := '');
```

Parameters

Table 20–50 CREATE_MASTER_REPGROUP Procedure Parameters

Parameter	Description
`gname`	Name of the master 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 master group. Be sure to use the @ sign. See *Oracle9i Replication* and *Oracle9i Database Administrator's Guide* for more information about connection qualifiers.

Exceptions

Table 20–51 CREATE_MASTER_REPGROUP Procedure Exceptions

Exception	Description
`duplicaterepgroup`	Master group already exists.
`norepopt`	Advanced replication option is not installed.
`missingrepgroup`	Master group name was not specified.
`qualifiertoolong`	Connection qualifier is too long.

CREATE_MASTER_REPOBJECT Procedure

This procedure makes an object a replicated object by adding the object to a master group. This procedure preserves the object identifier for user-defined types and object tables at all replication sites.

Replication of clustered tables is supported, but the `use_existing_object` parameter cannot be set to `false` for clustered tables. In other words, you must create the clustered table at all master sites participating in the master group before you execute the `CREATE_MASTER_REPOBJECT` procedure. However, these 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 20–52 CREATE_MASTER_REPOBJECT Procedure Parameters **(Page 1 of 2)**

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. The object cannot be a storage table for a nested table.
type	Type of the object that you are replicating. The following types are supported: FUNCTION, SYNONYM INDEX, TABLE INDEXTYPE, TRIGGER OPERATOR, TYPE PACKAGE, TYPE BODY PACKAGE BODY, VIEW PROCEDURE

Table 20–52 CREATE_MASTER_REPOBJECT Procedure Parameters **(Page 2 of 2)**

Parameters	Description
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 20–54 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. **Note:** Do not use the ddl_text parameter to add user-defined types or object tables. Instead, create the object first and then add the object.
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, or 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 20–54 for more information.
gname	Name of the replication group in which you want to create the replicated object. The schema name is used as the default replication group name if none is specified, and a replication group with the same name as the schema must exist for the procedure to complete successfully in that case.

Table 20–53 CREATE_MASTER_REPOBJECT Procedure Exceptions

Exceptions	Description
`nonmasterdef`	Invocation site is not the master definition site.
`notquiesced`	Master group is not quiesced.
`duplicateobject`	Specified object already exists in the 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 20–54 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_MVIEW_REPGROUP Procedure

This procedure creates a new, empty materialized view group in your local database. CREATE_MVIEW_REPGROUP automatically calls REGISTER_MIEW_REPGROUP, but ignores any errors that may have happened during registration.

Syntax

```
DBMS_REPCAT.CREATE_MVIEW_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 20–55 CREATE_MVIEW_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replication group. This group must exist at the specified master site or master materialized view site.
master	Fully qualified database name of the database in the replication environment to use as the master site or master materialized view site. You can include a connection qualifier if necessary. See *Oracle9i Replication* and *Oracle9i Database Administrator's Guide* for information about using connection qualifiers.
comment	This comment is added to the DBA_REPGROUP view.
propagation_mode	Method of propagation for all updatable materialized views in the replication group. Acceptable values are synchronous and asynchronous.
fname	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.
gowner	Owner of the materialized view group.

Exceptions

Table 20–56 CREATE_MVIEW_REPGROUP Procedure Exceptions

Exception	Description
`duplicaterepgroup`	Replication group already exists at the invocation site.
`nonmaster`	Specified database is not a master site or master materialized view site.
`commfailure`	Specified database is not accessible.
`norepopt`	Advanced replication option is not installed.
`typefailure`	Propagation mode was specified incorrectly.
`missingrepgroup`	Replication group does not exist at master site.
`invalidqualifier`	Connection qualifier specified for the master site or master materialized view site is not valid for the replication group.
`alreadymastered`	At the local site, there is another materialized view group with the same group name, but different master site or master materialized view site.

CREATE_MVIEW_REPOBJECT Procedure

This procedure adds a replicated object to a materialized view group.

Syntax

```
DBMS_REPCAT.CREATE_MVIEW_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 20–57 CREATE_MVIEW_REPOBJECT Procedure Parameters **(Page 1 of 2)**

Parameter	Description
sname	Name of the schema in which the object is located. The schema must be same as the schema that owns the master table or master materialized view on which this materialized view is based.
oname	Name of the object that you want to add to the replicated materialized view group.
type	Type of the object that you are replicating. The following types are supported: FUNCTION, SNAPSHOT INDEX, SYNONYM INDEXTYPE, TRIGGER OPERATOR, TYPE PACKAGE, TYPE BODY PACKAGE BODY, VIEW PROCEDURE
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 materialized view with the same name already exists, then Oracle ignores the DDL and registers the existing materialized view as a replicated object. If the master table or master materialized view for a materialized view does not exist in the replication group of the master 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. If the object is not of type SNAPSHOT, then the materialized view site connects to the master site or master materialized view site and pulls down the DDL text to create the object. If the object type is TYPE or TYPE BODY, then the object identifier (OID) for the object at the materialized view site is the same as the OID at the master site or master materialized view site.

Table 20–57 CREATE_MVIEW_REPOBJECT Procedure Parameters (Page 2 of 2)

Parameter	Description
`comment`	This comment is added to the `OBJECT_COMMENT` field of the `DBA_REPOBJECT` view.
`gname`	Name of the replicated materialized view group to which you are adding an object. The schema name is used as the default group name if none is specified, and a materialized view group with the same name as the schema must exist for the procedure to complete successfully.
`gen_objs_owner`	Name of the user you want to assign as owner of the transaction.
`min_communication`	Set to `false` if the materialized view's 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 Chapter 6, "Configure Conflict Resolution" and *Oracle9i Replication*.
`generate_80_compatible`	Set to `true` if the materialized view's master site is running a version of Oracle server prior to Oracle8*i* release 8.1.5. Set to `false` if the materialized view's master site or master materialized view site is running Oracle8*i* release 8.1.5 or greater.
`gowner`	Owner of the materialized view group.

Exceptions

Table 20–58 CREATE_MVIEW_REPOBJECT Procedure Exceptions

Exception	Description
nonmview	Invocation site is not a materialized view site.
nonmaster	Master is no longer a master site or master materialized view site.
missingobject	Specified object does not exist in the master's replication 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 or master materialized view site is not accessible.
missingschema	Schema does not exist as a database schema.
badmviewddl	DDL was executed but materialized view does not exist.
onlyonemview	Only one materialized view for master table or master materialized view can be created.
badmviewname	Materialized view base table differs from master table or master materialized view.
missingrepgroup	Replication group at the master 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: Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–59 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 20–60 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	Replication group to which the specified table belongs is not quiesced.

DEFINE_PRIORITY_GROUP Procedure

This procedure creates a new priority group for a master group. You must call this procedure from the master definition site.

> **See Also:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–61 DEFINE_PRIORITY_GROUP Procedure Parameters

Parameter	Description
gname	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 20–62 DEFINE_PRIORITY_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
duplicatepriority group	Specified priority group already exists in the master group.
typefailure	Specified datatype is not supported.
notquiesced	Master group is not quiesced.

DEFINE_SITE_PRIORITY Procedure

This procedure creates a new site priority group for a master group. You must call this procedure from the master definition site.

> **See Also:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–63 DEFINE_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	The 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 20–64 DEFINE_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
duplicate prioritygroup	Specified site priority group already exists in the master group.
notquiesced	Master group is not quiesced.

DO_DEFERRED_REPCAT_ADMIN Procedure

This procedure executes the local outstanding deferred administrative procedures for the specified 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 20–65 DO_DEFERRED_REPCAT_ADMIN Procedure Parameters

Parameter	Description
gname	Name of the master group.
all_sites	If this is true, then use a job to execute the local administrative procedures at each master site.

Exceptions

Table 20–66 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:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–67 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 20–68 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	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:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–69 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. The table can be the storage table of a nested table.
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 index-by table of column names. The PL/SQL index-by table must be of type DBMS_REPCAT.VARCHAR2. You can specify column objects, but you cannot specify attributes of column objects. If the table is an object, then you can specify SYS_NC_OID$ to add the object identifier column to the column group. This column tracks the object identifier of each row object. If the table is a storage table of a nested table, then you can specify NESTED_TABLE_ID to add the column that tracks the identifier for each row of the nested table.

Exceptions

Table 20–70 DROP_GROUPED_COLUMN Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingobject	Specified table does not exist.
notquiesced	Master group that the table belongs to is not quiesced.

DROP_MASTER_REPGROUP Procedure

This procedure drops a master group from your current site. To drop the master group from all master sites, including the master definition site, you can call this procedure at the master definition site, and set `all_sites` to `true`.

Syntax

```
DBMS_REPCAT.DROP_MASTER_REPGROUP (
   gname             IN VARCHAR2,
   drop_contents     IN BOOLEAN    := false,
   all_sites         IN BOOLEAN    := false);
```

Parameters

Table 20–71 DROP_MASTER_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the master group that you want to drop from the current master site.
drop_contents	By default, when you drop the replication 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 replication 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 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 20–72 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 remote procedure call (RPC) queue has entries for the 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 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 20–73 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 master group. The object cannot be a storage table for a nested table.
type	Type of object that you want to drop. The following types are supported: FUNCTION SYNONYM INDEX TABLE INDEXTYPE TRIGGER OPERATOR TYPE PACKAGE TYPE BODY PACKAGE BODY VIEW PROCEDURE
drop_objects	By default, the object remains in the schema, but is dropped from the master group. That is, any changes to the object are no longer replicated to other master and materialized view sites. To completely remove the object from the replication 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 20–74 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_MVIEW_REPGROUP Procedure

This procedure drops a materialized view site from your replication environment. DROP_MVIEW_REPGROUP automatically calls UNREGISTER_MVIEW_REPGROUP at the master site or master materialized view site to unregister the materialized view, but ignores any errors that may have occurred during unregistration. If DROP_MVIEW_REPGROUP is unsuccessful, then connect to the master site or master materialized view site and run UNREGISTER_MVIEW_REPGROUP.

Syntax

```
DBMS_REPCAT.DROP_MVIEW_REPGROUP (
   gname                IN   VARCHAR2,
   drop_contents        IN   BOOLEAN   := false
   gowner               IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 20–75   DROP_MVIEW_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replication group that you want to drop from the current materialized view site. All objects generated to support replication, such as triggers and packages, are dropped.
drop_contents	By default, when you drop the replication group at a materialized view 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 replication group are dropped from their schemas.
gowner	Owner of the materialized view group.

Exceptions

Table 20–76   DROP_MVIEW_REPGROUP Procedure Exceptions

Exception	Description
nonmview	Invocation site is not a materialized view site.
missingrepgroup	Specified replication group does not exist.

DROP_MVIEW_REPOBJECT Procedure

This procedure drops a replicated object from a materialized view site.

Syntax

```
DBMS_REPCAT.DROP_MVIEW_REPOBJECT (
   sname          IN   VARCHAR2,
   oname          IN   VARCHAR2,
   type           IN   VARCHAR2,
   drop_objects   IN   BOOLEAN  := false);
```

Parameters

Table 20–77 DROP_MVIEW_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 replication group.
type	Type of the object that you want to drop. The following types are supported: FUNCTION, SNAPSHOT INDEX, SYNONYM INDEXTYPE, TRIGGER OPERATOR, TYPE PACKAGE, TYPE BODY PACKAGE BODY, VIEW PROCEDURE
drop_objects	By default, the object remains in its associated schema, but is dropped from its associated replication group. To completely remove the object from its schema at the current materialized view site, set this parameter to `true`. If the parameter is set to `true`, the object is dropped from the database at the materialized view site.

Exceptions

Table 20–78 DROP_MVIEW_REPOBJECT Procedure Exceptions

Exception	Description
nonmview	Invocation site is not a materialized view site.
missingobject	Specified object does not exist.
typefailure	Specified type parameter is not supported.

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:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–79 DROP_PRIORITY Procedure Parameters

Parameter	Description
gname	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 20–80 DROP_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
missingprioritygroup	Specified priority group does not exist.
notquiesced	Master group is not quiesced.

DROP_PRIORITY_GROUP Procedure

This procedure drops a priority group for a specified master group. You must call this procedure from the master definition site.

> **See Also:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DROP_PRIORITY_GROUP (
   gname     IN   VARCHAR2,
   pgroup    IN   VARCHAR2);
```

Parameters

Table 20–81 DROP_PRIORITY_GROUP Procedure Parameters

Parameter	Description
gname	Master group with which the priority group is associated.
pgroup	Name of the priority group that you want to drop.

Exceptions

Table 20–82 DROP_PRIORITY_GROUP Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
referenced	Specified priority group is being used in conflict resolution.
notquiesced	Specified 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:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–83 DROP_PRIORITY_datatype Procedure Parameters

Parameter	Description
gname	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 20–84 DROP_PRIORITY_datatype Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified 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 master group is not quiesced

DROP_SITE_PRIORITY Procedure

This procedure drops a site priority group for a specified master group. You must call this procedure from the master definition site.

> **See Also:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i Replication* for more information about conflict resolution methods

Syntax

```
DBMS_REPCAT.DROP_SITE_PRIORITY (
   gname      IN   VARCHAR2,
   name       IN   VARCHAR2);
```

Parameters

Table 20–85 DROP_SITE_PRIORITY Procedure Parameters

Parameter	Description
gname	Master group with which the site priority group is associated.
name	Name of the site priority group that you want to drop.

Exceptions

Table 20–86 DROP_SITE_PRIORITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
referenced	Specified site priority group is being used in conflict resolution.
notquiesced	Specified 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:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–87 DROP_SITE_PRIORITY_SITE Procedure Parameters

Parameter	Description
gname	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 20–88 DROP_SITE_PRIORITY_SITE Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
missingrepgroup	Specified master group does not exist.
missingpriority	Specified site priority group does not exist.
notquiesced	Specified master group is not quiesced.

DROP_*conflicttype*_RESOLUTION Procedure

This procedure drops an update, delete, or uniqueness conflict resolution routine. You must call these procedures from the master definition site. The procedure that you must call is determined by the type of conflict that the routine resolves.

Conflict Resolution Routines

The following table shows the procedure name for each conflict resolution routine.

Table 20–89 Conflict Resolution Routines

Routine	Procedure Name
update	DROP_UPDATE_RESOLUTION
uniqueness	DROP_UNIQUE_RESOLUTION
delete	DROP_DELETE_RESOLUTION

Syntax

```
DBMS_REPCAT.DROP_UPDATE_RESOLUTION (
   sname              IN   VARCHAR2,
   oname              IN   VARCHAR2,
   column_group       IN   VARCHAR2,
   sequence_no        IN   NUMBER);

DBMS_REPCAT.DROP_DELETE_RESOLUTION (
   sname              IN   VARCHAR2,
   oname              IN   VARCHAR2,
   sequence_no        IN   NUMBER);

DBMS_REPCAT.DROP_UNIQUE_RESOLUTION (
   sname              IN   VARCHAR2,
   oname              IN   VARCHAR2,
   constraint_name    IN   VARCHAR2,
   sequence_no        IN   NUMBER);
```

Parameters

Table 20–90 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 20–91 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`	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 20–92 EXECUTE_DDL Procedure Parameters

Parameter	Description
gname	Name of the master group.
master_list	A comma-separated list of master sites at which you want to execute the supplied DDL. Do not put any spaces 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 that lists the master sites where 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 execute 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 20–93 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_MVIEW_SUPPORT Procedure

This procedure activates triggers and generate packages needed to support the replication of updatable materialized views or procedural replication.You must call this procedure from the materialized view site.

> **Note:** CREATE_MVIEW_REPOBJECT automatically generates materialized view support for updatable materialized views.

Syntax

```
DBMS_REPCAT.GENERATE_MVIEW_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 20–94 GENERATE_MVIEW_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 the materialized view's master site is running a version of Oracle server prior to Oracle8*i* release 8.1.5. Set to false if the materialized view's master site or master materialized view site is running Oracle8*i* release 8.1.5 or higher.

Exceptions

Table 20–95 GENERATE_MVIEW_SUPPORT Procedure Exceptions

Exceptions	Descriptions
`nonmview`	Invocation site is not a materialized view site.
`missingobject`	Specified object does not exist as a materialized view in the replicated schema waiting for row/column-level replication information or as a package (body) waiting for wrapper generation.
`typefailure`	Specified type parameter is not supported.
`missingschema`	Specified owner of generated objects does not exist.
`missingremoteobject`	Object at master site or master materialized view site has not yet generated replication support.
`commfailure`	Master site or master materialized view site is not accessible.

GENERATE_REPLICATION_SUPPORT Procedure

This procedure generates the triggers and packages needed to support replication for a specified object. 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 20–96 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 *Oracle9i 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 all master sites are running Oracle8*i* release 8.1.5 or higher.

Exceptions

Table 20–97 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 waiting for row-level replication information or as a package (body) waiting for wrapper generation.
typefailure	Specified type parameter is not supported.
notquiesced	Replication group has not been quiesced.
commfailure	At least one master site is not accessible.
missingschema	Schema does not exist.
dbnotcompatible	One of the master sites is not 7.3.0.0 compatible.
notcompat	One of the master sites is not 7.3.0.0 compatible. (Equivalent to dbnotcompatible.)
duplicateobject	Object already exists.

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:** Chapter 6, "Configure Conflict Resolution" and *Oracle9i 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 20–98 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. The table can be the storage table of a nested table.
`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 index-by table of column names. The PL/SQL index-by table must be of type `DBMS_REPCAT.VARCHAR2`. Use the single value `'*'` to create a column group that contains all of the columns in your table. You can specify column objects, but you cannot specify attributes of column objects. If the table is an object table, then you can specify `SYS_NC_OID$` to add the object identifier column to the column group. This column tracks the object identifier of each row object. If the table is the storage table of a nested table, then you can specify `NESTED_TABLE_ID` to add the column that tracks the identifier for each row of the nested table.

Exceptions

Table 20–99 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`	Master group is not quiesced.

PREPARE_INSTANTIATED_MASTER Procedure

This procedure enables the propagation of deferred transactions from other prepared new master sites and existing master sites to the invocation master site. This procedure also enables the propagation of deferred transactions from the invocation master site to the other prepared new master sites and existing master sites.

If you performed a full database export/import or a change-based recovery, then the new master site includes all of the deferred transactions that were in the deferred transactions queue at the master definition site. Because these deferred transactions should not exist at the new master site, this procedure deletes all transactions in the deferred transactions queue and error queue if full database export/import or change-based recovery was used.

For object-level export/import, ensure that all the requests in the `DBA_REPCATLOG` data dictionary view for the extended groups have been processed without error before running this procedure.

Caution:

- Do not invoke this procedure until instantiation (export/import or change-based recovery) for the new master site is complete.
- Do not allow any data manipulation language (DML) statements directly on the objects in the extended master group in the new master site until execution of this procedure returns successfully. These DML statements may not be replicated.
- Do not use the `DBMS_DEFER` package to create deferred transactions until execution of this procedure returns successfully. These deferred transactions may not be replicated.

Note: To use change-based recovery, the existing master site and the new master site must be running under the same operating system, although the release of the operating system can differ.

Syntax

```
DBMS_REPCAT.PREPARE_INSTANTIATED_MASTER (
   extension_id            IN       RAW);
```

Parameters

Table 20–100 PREPARE_INSTANTIATED_MASTER Procedure Parameters

Parameter	Description
extension_id	The identifier for the current pending request to add master databases to a master group without quiesce. You can find the extension_id by querying the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.

Exceptions

Table 20–101 PREPARE_INSTANTIATED_MASTER Procedure Exceptions

Exception	Description
typefailure	The parameter value specified for one of the parameters is not appropriate.
dbnotcompatible	Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.

PURGE_MASTER_LOG Procedure

This procedure removes local messages in the DBA_REPCATLOG view associated with a specified identification number, source, or master group.

To purge all of the administrative requests from a particular source, specify NULL for the id parameter. To purge all administrative requests from all sources, specify NULL for both the id parameter and the source parameter.

Syntax

```
DBMS_REPCAT.PURGE_MASTER_LOG (
   id      IN   BINARY_INTEGER,
   source  IN   VARCHAR2,
   gname   IN   VARCHAR2);
```

Parameters

Table 20–102 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 master group for which the request was made.

Exceptions

Table 20–103 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 20–104 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 20–105 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_MVIEW_REPGROUP Procedure

This procedure refreshes a materialized view group with the most recent data from its associated master site or master materialized view site.

Syntax

```
DBMS_REPCAT.REFRESH_MVIEW_REPGROUP (
   gname                  IN   VARCHAR2,
   drop_missing_contents  IN   BOOLEAN    := false,
   refresh_mviews         IN   BOOLEAN    := false,
   refresh_other_objects  IN   BOOLEAN    := false
   gowner                 IN   VARCHAR2   := 'PUBLIC');
```

Parameters

Table 20–106 REFRESH_MVIEW_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the replication group.
drop_missing_contents	If an object was dropped from the replication group at the master site or master materialized view site, then it is not automatically dropped from the schema at the materialized view site. It is simply no longer replicated. That is, changes to this object are no longer sent to its associated master site or master materialized view site. Materialized views can continue to be refreshed from their associated master tables or master materialized views. However, any changes to an updatable materialized view are lost. When an object is dropped from the replication group, you can choose to have it dropped from the schema entirely by setting this parameter to true.
refresh_mviews	Set to true to refresh the contents of the materialized views in the replication group.
refresh_other_objects	Set this to true to refresh the contents of the nonmaterialized view objects in the replication group. Nonmaterialized view objects may include the following: ■ Tables ■ Views ■ Indexes ■ PL/SQL packages and package bodies ■ PL/SQL procedures and functions ■ Triggers ■ Synonyms
gowner	Owner of the materialized view group.

Exceptions

Table 20–107 REFRESH_MVIEW_REPGROUP Procedure Exceptions

Exception	Description
nonmview	Invocation site is not a materialized view site.
nonmaster	Master is no longer a master site or master materialized view site.
commfailure	Master site or master materialized view site is not accessible.
missingrepgroup	Replication group name not specified.

REGISTER_MVIEW_REPGROUP Procedure

This procedure facilitates the administration of materialized views at their respective master sites or master materialized view sites by inserting or modifying a materialized view group in DBA_REGISTERED_MVIEW_GROUPS.

Syntax

```
DBMS_REPCAT.REGISTER_MVIEW_REPGROUP (
   gname            IN   VARCHAR2,
   mviewsite         IN   VARCHAR2,
   comment          IN   VARCHAR2  := NULL,
   rep_type         IN   NUMBER    := reg_unknown,
   fname            IN   VARCHAR2  := NULL
   gowner           IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 20–108 REGISTER_MVIEW_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the materialized view group to be registered.
mviewsite	Global name of the materialized view site.
comment	Comment for the materialized view site or update for an existing comment.
rep_type	Version of the materialized view group. Valid constants that can be assigned include the following: ▪ dbms_repcat.reg_unknown (the default) ▪ dbms_repcat.reg_v7_group ▪ dbms_repcat.reg_v8_group
fname	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.
gowner	Owner of the materialized view group.

Exceptions

Table 20–109 REGISTER_MVIEW_REPGROUP Procedure Exceptions

Exception	Description
failregmviewrepgroup	Registration of materialized view group failed.
missingrepgroup	Replication group name not specified.
nullsitename	A materialized view site was not specified.
nonmaster	Procedure must be executed at the materialized view's master site or master materialized view site.
duplicaterepgroup	Replication group 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 20–110 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 20–111 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 replication 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 20–112 RELOCATE_MASTERDEF Procedure Parameters

Parameter	Description
gname	Name of the replication 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.
require_flavor_change	This parameter is for internal use only. **Note:** Do not set this parameter unless directed to do so by Oracle Support Services.

Exceptions

Table 20–113 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 replication 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 20–114 REMOVE_MASTER_DATABASES Procedure Parameters

Parameter	Description
gname	Name of the replication group associated with the replication environment. This prevents confusion if a master database is involved in more than one replication environment.
master_list	A comma-separated list of fully qualified master database names that you want to remove from the replication environment. There must be no spaces between names in the list.
master_table	In place of a list, you can specify the database names in a PL/SQL index-by table of type DBMS_UTILITY.DBLINK_ARRAY.

Exceptions

Table 20–115 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.

RENAME_SHADOW_COLUMN_GROUP Procedure

This procedure renames the shadow column group of a replicated table to make it a named column group. The replicated table's master group does not need to be quiesced to run this procedure.

Syntax

```
DBMS_REPCAT.RENAME_SHADOW_COLUMN_GROUP (
   sname               IN VARCHAR2,
   oname               IN VARCHAR2,
   new_col_group_name  IN VARCHAR2)
```

Parameters

Table 20–116 RENAME_SHADOW_COLUMN_GROUP Procedure Parameters

Parameter	Description
sname	Schema in which the replicated table is located.
oname	Name of the replicated table.
new_col_group_name	Name of the new column group. The columns currently in the shadow group are placed in a column group with the name you specify.

Exceptions

Table 20–117 RENAME_SHADOW_COLUMN_GROUP Procedure Exceptions

Exception	Description
missmview	The specified schema does not exist.
nonmasterdef	Invocation site is not the master definition site.
missingobject	The specified object does not exist.
duplicategroup	The column group that was specified for creation already exists.

REPCAT_IMPORT_CHECK Procedure

This procedure ensures that the objects in the 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 20–118 REPCAT_IMPORT_CHECK Procedure Parameters

Parameter	Description
gname	Name of the master group. If you omit both parameters, then the procedure checks all 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 materialized view site.
gowner	Owner of the master group.

Exceptions

Table 20–119 REPCAT_IMPORT_CHECK Procedure Exceptions

Exception	Description
nonmaster	master is true and either the database is not a master site for the replication group or the database is not the expected database.
nonmview	master is false and the database is not a materialized view site for the replication group.
missingobject	A valid replicated object in the replication group does not exist.
missingrepgroup	The specified replicated replication group does not exist.
missingschema	The specified replicated replication group does not exist.

RESUME_MASTER_ACTIVITY Procedure

This procedure resumes normal replication activity after quiescing a replication environment.

Syntax

```
DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
   gname       IN  VARCHAR2,
   override    IN  BOOLEAN := false);
```

Parameters

Table 20–120 RESUME_MASTER_ACTIVITY Procedure Parameters

Parameter	Description
gname	Name of the master group.
override	If this is true, then it ignores any pending RepCat administrative 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 administrative request for gname at that master.

Exceptions

Table 20–121 RESUME_MASTER_ACTIVITY Procedure Exceptions

Exception	Description
`nonmasterdef`	Invocation site is not the master definition site.
`notquiesced`	Master group is not quiescing or quiesced.
`commfailure`	At least one master site is not accessible.
`notallgenerated`	Generate replication support before resuming replication activity.

RESUME_PROPAGATION_TO_MDEF Procedure

During the process of adding new master sites to a master group without quiesce, this procedure indicates that export is effectively finished and propagation to the master definition site for both extended and unaffected replication groups existing at master sites can be enabled. Run this procedure after the export required to add new master sites to a master group is complete.

> **See Also:** "Adding New Master Sites" on page 7-4 for more information about adding master sites to a master group

Syntax

```
DBMS_REPCAT.RESUME_PROPAGATION_TO_MDEF (
   extension_id                 IN    RAW);
```

Parameters

Table 20–122 RESUME_PROPAGATION_TO_MDEF Procedure Parameters

Parameter	Description
`extension_id`	The identifier for the current pending request to add master databases to a master group without quiesce. You can find the `extension_id` by querying the `DBA_REPSITES_NEW` and `DBA_REPEXTENSIONS` data dictionary views.

Exceptions

Table 20–123 RESUME_PROPAGATION_TO_MDEF Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
extstinapp	Extension status is inappropriate. The extension status should be EXPORTING when you run this procedure. To check the extension status, query the DBA_REPEXTENSIONS data dictionary view.
dbnotcompatible	Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.

SEND_OLD_VALUES Procedure

You have the option of sending old column values during propagation of deferred transactions for each nonkey column of a replicated table when rows are updated or deleted in the table. When min_communication is set to true, the default is the following:

- For a deleted row, to send old values for all columns
- For an updated row, to send old values for key columns and the modified columns in a column group

You can change this behavior at all master sites and materialized view sites by invoking DBMS_REPCAT.SEND_OLD_VALUES at the master definition site. Then, generate replication support at all master sites and at each materialized view site.

When you use user-defined types, you can specify the leaf attributes of a column object, or an entire column object. For example, if a column object named cust_address has street_address as an attribute, then you can specify cust_address.street_address for the column_list parameter or as part of the column_table parameter, or you can specify only cust_address.

Syntax

```
DBMS_REPCAT.SEND_OLD_VALUES(
   sname           IN  VARCHAR2,
   oname           IN  VARCHAR2,
   { column_list   IN  VARCHAR2,
   | column_table  IN  DBMS_UTILITY.VARCHAR2s | DBMS_UTILITY.LNAME_ARRAY,}
   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 20–124 SEND_OLD_VALUES Procedure Parameters

Parameter	Description
sname	Schema in which the table is located.
oname	Name of the replicated table. The table can be the storage table of a nested table.
column_list	A comma-separated list of the columns in the table. There must be no spaces between entries.
column_table	Instead of a list, you can use a PL/SQL index-by table of type DBMS_REPCAT.VARCHAR2 or DBMS_UTILITY.LNAME_ARRAY to contain the column names. The first column name should be at position 1, the second at position 2, and so on. Use DBMS_UTILITY.LNAME_ARRAY if any column name is greater than or equal to 30 bytes, which may occur when you specify the attributes of column objects.
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 materialized view site the next time replication support is generated at that site with min_communication true.

Note: The `operation` parameter enables you to specify whether or not to transmit old values for nonkey columns when rows are deleted or 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 *Oracle9i Replication* for information about reduced data propagation using the `SEND_OLD_VALUES` procedure before changing the default behavior of Oracle.

Exceptions

Table 20–125 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 waiting for row-level replication information.
`missingcolumn`	At least one column is not in the table.
`notquiesced`	Master group has not been quiesced.
`typefailure`	An illegal operation is specified.
`keysendcomp`	A specified column is a key column in a table.
`dbnotcompatible`	Feature is incompatible with database version. Typically, this exception arises when you are trying to send the attributes of column objects. In this case, all databases must be at 9.0.0 or higher compatibility level.

SET_COLUMNS Procedure

This procedure enables you to use an alternate column or group of columns, instead of the primary key, to determine which columns of a table to compare when using row-level replication. You must call this procedure from the master definition site.

When you use column objects, if an attribute of a column object can be used as a primary key or part of a primary key, then the attribute can be part of an alternate key column. For example, if a column object named `cust_address` has `street_address` as a `VARCHAR2` attribute, then you can specify `cust_address.street_address` for the `column_list` parameter or as part of the `column_table` parameter. However, the entire column object, `cust_address`, cannot be specified.

For the storage table of a nested table column, this procedure accepts the `NESTED_TABLE_ID` as an alternate key column.

When you use object tables, you cannot specify alternate key columns. If the object identifier (OID) is system-generated for an object table, then Oracle uses the OID column in the object table as the key for the object table. If the OID is user-defined for an object table, then Oracle uses the primary key in the object table as the key.

The following types of columns cannot be alternate key columns:

- LOB or LOB attribute of a column object
- Collection or collection attribute of a column object
- `REF`
- An entire column object

> **See Also:** The *constraint_clause* in *Oracle9i SQL Reference* for more information about restrictions on primary key columns

Syntax

```
DBMS_REPCAT.SET_COLUMNS (
   sname           IN    VARCHAR2,
   oname           IN    VARCHAR2,
   { column_list   IN    VARCHAR2
   | column_table  IN    DBMS_UTILITY.NAME_ARRAY | DBMS_UTILITY.LNAME_ARRAY } );
```

Note: This procedure is overloaded. The `column_list` and `column_table` parameters are mutually exclusive.

Parameters

Table 20–126 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 spaces between entries.
`column_table`	Instead of a list, you can use a PL/SQL index-by table of type `DBMS_UTILITY.NAME_ARRAY` or `DBMS_UTILITY.LNAME_ARRAY` to contain the column names. The first column name should be at position 1, the second at position 2, and so on. Use `DBMS_UTILITY.LNAME_ARRAY` if any column name is greater than or equal to 30 bytes, which may occur when you specify the attributes of column objects.

Exceptions

Table 20–127 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 waiting for row-level replication information.
`missingcolumn`	At least one column is not in the table.
`notquiesced`	Replication group is not quiescing or quiesced.

SPECIFY_NEW_MASTERS Procedure

This procedure specifies the master sites you intend to add to an existing replication group without quiescing the group. This procedure must be run at the master definition site of the specified master group.

If necessary, this procedure creates an `extension_id` that tracks the process of adding new master sites to a master group. You use this `extension_id` in the other procedures that you run at various stages in the process. You can view information about the `extension_id` in the `DBA_REPSITES_NEW` and `DBA_REPEXTENSIONS` data dictionary views.

This procedure adds the new master sites to the `DBA_REPSITES_NEW` data dictionary view for the specified replication group. This procedure can be run any number of times for a given replication group. If it is run more than once, then it replaces any masters in the local `DBA_REPSITES_NEW` data dictionary view for the specified replication group with the masters specified in the `master_list`/`master_table` parameters.

You must run this procedure before you run the `ADD_NEW_MASTERS` procedure. No new master sites are added to the master group until you run the `ADD_NEW_MASTERS` procedure.

See Also:

- "ADD_NEW_MASTERS Procedure" on page 20-11
- "Adding New Master Sites" on page 7-4 for more information about adding master sites to a master group

Syntax

```
DBMS_REPCAT.SPECIFY_NEW_MASTERS (
   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 20–128 SPECIFY_NEW_MASTERS Procedure Parameters

Parameter	Description
`gname`	Master group to which you are adding new master sites.
`master_list`	A comma-separated list of new master sites that you want to add to the master group. List only the new master sites, not the existing master sites. Do not put any spaces between site names. If `master_list` is `NULL`, all master sites for the given replication group are removed from the `DBA_REPSITES_NEW` data dictionary view. Specify `NULL` to indicate that the master group is not being extended.
`master_table`	A table that lists the new master sites that you want to add to the master group. In the table, list only the new master sites, not the existing master sites. The first master site should be at position 1, the second at position 2, and so on. If the table is empty, then all master sites for the specified replication group are removed from the `DBA_REPSITES_NEW` data dictionary view. Use an empty table to indicate that the master group is not being extended.

Exceptions

Table 20–129 SPECIFY_NEW_MASTERS Procedure Exceptions

Exception	Description
`duplicaterepgroup`	A master site that you are attempting to add is already part of the master group.
`nonmasterdef`	Invocation site is not the master definition site.
`propmodenotallowed`	Synchronous propagation mode not allowed for this operation. Only asynchronous propagation mode is allowed.
`extstinapp`	Extension request with status not allowed. There must either be no `extension_id` for the master group or the `extension_id` status must be `READY`. You can view the status for each `extension_id` at a master site in the `DBA_REPEXTENSIONS` data dictionary view.
`dbnotcompatible`	Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.
`notsamecq`	Master groups do not have the same connection qualifier.

SUSPEND_MASTER_ACTIVITY Procedure

This procedure suspends replication activity for a master group. You use this procedure to quiesce the master group. You must call this procedure from the master definition site.

Syntax

```
DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
   gname   IN   VARCHAR2);
```

Parameters

Table 20–130   SUSPEND_MASTER_ACTIVITY Procedure Parameters

Parameter	Description
gname	Name of the master group for which you want to suspend activity.

Exceptions

Table 20–131   SUSPEND_MASTER_ACTIVITY Procedure Exceptions

Exception	Description
nonmasterdef	Invocation site is not the master definition site.
notnormal	Master group is not in normal operation.
commfailure	At least one master site is not accessible.

SWITCH_MVIEW_MASTER Procedure

This procedure changes the master site of a materialized view group to another master site. This procedure does a full refresh of the affected materialized views and regenerates the triggers and their associated packages as needed. This procedure does not push the queue to the old master site before changing master sites.

If `min_communication` is `true` for the materialized view and the new master site is an Oracle7 master site, then regenerate replication support for the materialized view with `min_communication` set to `false`.

If `generate_80_compatible` is `false` for the materialized view and the new master site is a release lower than Oracle8*i* (Oracle7 or Oracle8), then regenerate replication support for the materialized view with `generate_80_compatible` set to `true`.

You can set both parameters for a materialized view in one call to `DBMS_REPCAT.GENERATE_MVIEW_SUPPORT`.

Note: You cannot switch the master of materialized views that are based on other materialized views (level 2 and greater materialized views). Such a materialized view must be dropped and recreated if you want to base it on a different master.

See Also: "GENERATE_MVIEW_SUPPORT Procedure" on page 20-82

Syntax

```
DBMS_REPCAT.SWITCH_MVIEW_MASTER (
   gname        IN   VARCHAR2,
   master       IN   VARCHAR2
   gowner       IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 20–132 SWITCH_MVIEW_MASTER Procedure Parameters

Parameter	Description
gname	Name of the materialized view group for which you want to change the master site.
master	Fully qualified database name of the new master site to use for the materialized view group.
gowner	Owner of the materialized view group.

Exceptions

Table 20–133 SWITCH_MVIEW_MASTER Procedure Exceptions

Exception	Description
nonmview	Invocation site is not a materialized view site.
nonmaster	Specified database is not a master site.
commfailure	Specified database is not accessible.
missingrepgroup	Materialized view group does not exist.
qrytoolong	Materialized view definition query is greater 32 KB.
alreadymastered	At the local site, there is another materialized view group with the same group name mastered at the old master site.

UNDO_ADD_NEW_MASTERS_REQUEST Procedure

This procedure undoes all of the changes made by the SPECIFY_NEW_MASTERS and ADD_NEW_MASTERS procedures for a specified extension_id.

This procedure is executed at one master site, which may be the master definition site, and it only affects that master site. If you run this procedure at one master site affected by the request, you must run it at all new and existing master sites affected by the request. You can query the DBA_REPSITES_NEW data dictionary view to see the new master sites affected by the extension_id. This data dictionary view also lists the replication group name, and you must run this procedure at all existing master sites in the replication group.

Caution: This procedure is not normally called. Use this procedure only if the adding new masters without quiesce operation cannot proceed at one or more master sites. Run this procedure after you have already run the SPECIFY_NEW_MASTERS and ADD_NEW_MASTERS procedures, but *before* you have run the RESUME_PROPAGATION_TO_MDEF and PREPARE_INSTANTIATED_MASTER procedures.

Do not run this procedure after you have run either RESUME_PROPAGATION_TO_MDEF or PREPARE_INSTANTIATED_MASTER for a particular extension_id.

See Also:

- "SPECIFY_NEW_MASTERS Procedure" on page 20-107
- "ADD_NEW_MASTERS Procedure" on page 20-11
- "RESUME_PROPAGATION_TO_MDEF Procedure" on page 20-101
- "PREPARE_INSTANTIATED_MASTER Procedure" on page 20-87

Syntax

```
DBMS_REPCAT.UNDO_ADD_NEW_MASTERS_REQUEST (
   extension_id         IN RAW,
   drop_contents        IN BOOLEAN := TRUE);
```

Parameters

Table 20–134 UNDO_ADD_NEW_MASTERS_REQUEST Procedure Parameters

Parameter	Description
extension_id	The identifier for the current pending request to add master databases to a master group without quiesce. You can find the extension_id by querying the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.
drop_contents	Specify true, the default, to drop the contents of objects in new replication groups being extended at the local site. Specify false to retain the contents.

Exceptions

Table 20–135 UNDO_ADD_NEW_MASTERS_REQUEST Procedure Exceptions

Exception	Description
dbnotcompatible	Feature is incompatible with database version. All databases must be at 9.0.0 or higher compatibility level.
typefail	A parameter value that you specified is not appropriate.

UNREGISTER_MVIEW_REPGROUP Procedure

This procedure facilitates the administration of materialized views at their respective master sites or master materialized view sites by deleting a materialized view group from DBA_REGISTERED_MVIEW_GROUPS. Run this procedure at the master site or master materialized view site.

Syntax

```
DBMS_REPCAT.UNREGISTER_MVIEW_REPGROUP (
   gname      IN   VARCHAR2,
   mviewsite  IN   VARCHAR2
   gowner     IN   VARCHAR2  := 'PUBLIC');
```

Parameters

Table 20–136 UNREGISTER_MVIEW_REPGROUP Procedure Parameters

Parameter	Description
gname	Name of the materialized view group to be unregistered.
mviewsite	Global name of the materialized view site.
gowner	Owner of the materialized view group.

VALIDATE Function

This function validates the correctness of key conditions of a multimaster 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" on page 20-114, 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" on page 20-114 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 20–137 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 20–138 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. Oracle Replication expects the two users to be the same. Ensure that the user identification of the replication administration user at the local node and the user identification 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 replication group is replicated at a pre-Oracle8 node. Ensure that all nodes of the master group are running Oracle8 and higher.

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 20–139 WAIT_MASTER_LOG Procedure Parameters

Parameter	Description
gname	Name of the 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 20–140 WAIT_MASTER_LOG Procedure Exceptions

Exception	Description
nonmaster	Invocation site is not a master site.

21

DBMS_REPCAT_ADMIN

DBMS_REPCAT_ADMIN enables you to create users with the privileges needed by the symmetric replication facility.

This chapter discusses the following topics:

- Summary of DBMS_REPCAT_ADMIN Subprograms

Summary of DBMS_REPCAT_ADMIN Subprograms

Table 21–1 DBMS_REPCAT_ADMIN Package Subprograms

Subprogram	Description
GRANT_ADMIN_ANY_SCHEMA Procedure on page 21-3	Grants the necessary privileges to the replication administrator to administer any replication group at the current site.
GRANT_ADMIN_SCHEMA Procedure on page 21-4	Grants the necessary privileges to the replication administrator to administer a schema at the current site.
REGISTER_USER_REPGROUP Procedure on page 5	Assigns proxy materialized view administrator or receiver privileges at the master site or master materialized view site for use with remote sites.
REVOKE_ADMIN_ANY_SCHEMA Procedure on page 21-7	Revokes the privileges and roles from the replication administrator that were granted by GRANT_ADMIN_ANY_SCHEMA.
REVOKE_ADMIN_SCHEMA Procedure on page 21-8	Revokes the privileges and roles from the replication administrator that were granted by GRANT_ADMIN_SCHEMA.
UNREGISTER_USER_REPGROUP Procedure on page 21-9	Revokes the privileges and roles from the proxy materialized view 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 replication groups at the current site.

Syntax

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (
   username  IN   VARCHAR2);
```

Parameters

Table 21–2 GRANT_ADMIN_ANY_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator to whom you want to grant the necessary privileges and roles to administer any replication groups at the current site.

Exceptions

Table 21–3 GRANT_ADMIN_ANY_REPGROUP Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

GRANT_ADMIN_SCHEMA Procedure

This procedure grants the necessary privileges to the replication administrator to administer a schema at the current site. This procedure is most useful if your replication group does not span schemas.

Syntax

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_SCHEMA (
   username IN VARCHAR2);
```

Parameters

Table 21–4 GRANT_ADMIN_REPSCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator. This user is then granted the necessary privileges and roles to administer the schema of the same name within a replication group at the current site.

Exceptions

Table 21–5 GRANT_ADMIN_REPSCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

REGISTER_USER_REPGROUP Procedure

This procedure assigns proxy materialized view administrator or receiver privileges at the master site or master materialized view site for use with remote sites. This procedure grants only the necessary privileges to the proxy materialized view 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 21–6 REGISTER_USER_REPGROUP Procedure Parameters

Parameter	Description
username	Name of the user to whom you are giving either proxy materialized view 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 materialized view administration privileges
list_of_gnames	Comma-separated list of replication groups you want a user registered for receiver privileges. There must be no spaces between entries in the list. If you set list_of_gnames to NULL, then the user is registered for all replication groups, even replication 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 replication group in the list causes registration to fail for the entire list.
table_of_gnames	PL/SQL index-by table of replication groups you want a user registered for receiver privileges. The PL/SQL index-by 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 replication groups. An invalid replication group in the table causes registration to fail for the entire table.

Exceptions

Table 21–7 REGISTER_USER_REPGROUP Procedure Exceptions

Exception	Description
nonmaster	Specified replication group does not exist or the invocation database is not a master site or master materialized view site.
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 21–8 REVOKE_ADMIN_ANY_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator whose privileges you want to revoke.

Exceptions

Table 21–9 REVOKE_ADMIN_ANY_SCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

REVOKE_ADMIN_SCHEMA Procedure

This procedure revokes the privileges and roles from the replication administrator that 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 21–10 REVOKE_ADMIN_SCHEMA Procedure Parameters

Parameter	Description
username	Name of the replication administrator whose privileges you want to revoke.

Exceptions

Table 21–11 REVOKE_ADMIN_SCHEMA Procedure Exceptions

Exception	Description
ORA-01917	User does not exist.

UNREGISTER_USER_REPGROUP Procedure

This procedure revokes the privileges and roles from the proxy materialized view 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 21–12 UNREGISTER_USER_REPGROUP Procedure Parameters

Parameter	Description
username	Name of the user you are unregistering.
privilege_type	Specifies the privilege type you are revoking. Use the following values for to define your privilege_type: ■ receiver for receiver privileges ■ proxy_snapadmin for proxy materialized view administration privileges
list_of_gnames	Comma-separated list of replication groups you want a user unregistered for receiver privileges. There must be no spaces between entries in the list. If you set list_of_gnames to NULL, then the user is unregistered for all replication groups registered. You must use named notation in order to set list_of_gnames to NULL. An invalid replication group in the list causes unregistration to fail for the entire list.
table_of_gnames	PL/SQL index-by table of replication groups you want a user unregistered for receiver privileges. The PL/SQL index-by 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 replication groups registered. An invalid replication group in the table causes unregistration to fail for the entire table.

Exceptions

Table 21–13 UNREGISTER_USER_REPGROUP Procedure Exceptions

Exception	Description
nonmaster	Specified replication group does not exist or the invocation database is not a master site or master materialized view site.
ORA-01917	User does not exist.
typefailure	Incorrect privilege type was specified.

22

DBMS_REPCAT_INSTANTIATE

The DBMS_REPCAT_INSTANTIATE package instantiates deployment templates.

This chapter discusses the following topics:

- Summary of DBMS_REPCAT_INSTANTIATE Subprograms

Summary of DBMS_REPCAT_INSTANTIATE Subprograms

Table 22–1 DBMS_REPCAT_INSTANTIATE Package Subprograms

Subprogram	Description
DROP_SITE_INSTANTIATION Procedure on page 22-3	Public procedure that removes the target site from the DBA_REPCAT_TEMPLATE_SITES view.
INSTANTIATE_OFFLINE Function on page 22-3	Public function that generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while offline.
INSTANTIATE_ONLINE Function on page 22-6	Public function that generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while online.

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 materialized views. 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 22–2 DROP_SITE_INSTANTIATION Procedure Parameters

Parameter	Description
refresh_template_name	The name of the deployment template to be dropped.
site_name	Identifies the master site where you want to drop the specified template instantiation.

INSTANTIATE_OFFLINE Function

This function generates a file at the master site that is used to create the materialized view environment at the remote materialized view site while offline. This generated file is an offline instantiation file and should be used at remote materialized view sites that are not able to remain connected to the master site for an extended amount of time.

This is an ideal solution when the remote materialized view site is a laptop. Use the packaging interface in the Replication Management tool 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.

The script generated by this function is stored in the USER_REPCAT_TEMP_OUTPUT temporary view and is used by several Oracle tools, including the Replication Management tool, 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 materialized view). See these respective packages for more information on their usage.

See Also:

- "Packaging a Deployment Template for Instantiation" on page 4-12
- *Oracle9i Replication*
- The Replication Management tool's online help

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 22–3 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 identification used when creating the runtime parameters (the identification 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 materialized view groups created are owned by the default user `PUBLIC`. If `false`, then any materialized view groups created are owned by the user performing the instantiation.

Exceptions

Table 22–4 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 materialized view site. A deployment template can be instantiated only once at a particular materialized view site.
`not_authorized`	The user attempting to instantiate the deployment template is not authorized to do so.

Returns

Table 22–5 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_ONLINE Function

This function generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while online. This generated script should be used at remote materialized view sites that are able to remain connected to the master site for an extended amount of time, as the instantiation process at the remote materialized view site may be lengthy (depending on the amount of data that is populated to the new materialized views).

The script generated by this function is stored in the `USER_REPCAT_TEMP_OUTPUT` temporary view and is used by several Oracle tools, including the Replication Management tool, 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.

See Also:

- "Packaging a Deployment Template for Instantiation" on page 4-12
- *Oracle9i Replication*
- The Replication Management tool's online help

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 22–6 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 identification used when creating the runtime parameters (the identification 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 materialized view groups created are owned by the default user PUBLIC. If false, then any materialized view groups created are owned by the user performing the instantiation.

Table 22–7 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 materialized view site. A deployment template can be instantiated only once at a particular materialized view site.
not_authorized	The user attempting to instantiate the deployment template is not authorized to do so.

Returns

Table 22–8 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.

23

DBMS_REPCAT_RGT

DBMS_REPCAT_RGT controls the maintenance and definition of refresh group templates.

This chapter discusses the following topics:

- Summary of DBMS_REPCAT_RGT Subprograms

Summary of DBMS_REPCAT_RGT Subprograms

Table 23–1 DBMS_REPCAT_RGT Package Subprograms **(Page 1 of 3)**

Subprogram	Description
ALTER_REFRESH_TEMPLATE Procedure on page 23-5	Allows the DBA to alter existing deployment templates.
ALTER_TEMPLATE_OBJECT Procedure on page 23-7	Alters objects that have been added to a specified deployment template.
ALTER_TEMPLATE_PARM Procedure on page 23-10	Allows the DBA to alter the parameters for a specific deployment template.
ALTER_USER_AUTHORIZATION Procedure on page 23-12	Alters the contents of the DBA_REPCAT_USER_AUTHORIZATIONS view.
ALTER_USER_PARM_VALUE Procedure on page 23-14	Changes existing parameter values that have been defined for a specific user.
COMPARE_TEMPLATES Function on page 23-16	Allows the DBA to compare the contents of two deployment templates.
COPY_TEMPLATE Function on page 23-18	Allows the DBA to copy a deployment template.
CREATE_OBJECT_FROM_EXISTING Function on page 23-20	Creates a template object definition from existing database objects and adds it to a target deployment template.
CREATE_REFRESH_TEMPLATE Function on page 23-22	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 23-24	Adds object definitions to a target deployment template container.
CREATE_TEMPLATE_PARM Function on page 23-27	Creates parameters for a specific deployment template to allow custom data sets to be created at the remote materialized view site.
CREATE_USER_AUTHORIZATION Function on page 23-30	Authorizes specific users to instantiate private deployment templates.
CREATE_USER_PARM_VALUE Function on page 23-31	Predefines deployment template parameter values for specific users.
DELETE_RUNTIME_PARMS Procedure on page 23-33	Deletes a runtime parameter value that you defined using the INSERT_RUNTIME_PARMS procedure.

Table 23–1 DBMS_REPCAT_RGT Package Subprograms (Page 2 of 3)

Subprogram	Description
DROP_ALL_OBJECTS Procedure on page 23-34	Allows the DBA to drop all objects or specific object types from a deployment template.
DROP_ALL_TEMPLATE_PARMS Procedure on page 23-36	Allows the DBA to drop template parameters for a specified deployment template.
DROP_ALL_TEMPLATE_SITES Procedure on page 23-37	Removes all entries from the DBA_REPCAT_TEMPLATE_SITES view.
DROP_ALL_TEMPLATES Procedure on page 23-38	Removes all deployment templates at the site where the procedure is called.
DROP_ALL_USER_AUTHORIZATIONS Procedure on page 23-38	Allows the DBA to drop all user authorizations for a specified deployment template.
DROP_ALL_USER_PARM_VALUES Procedure on page 23-39	Drops user parameter values for a specific deployment template.
DROP_REFRESH_TEMPLATE Procedure on page 23-40	Drops a deployment template.
DROP_SITE_INSTANTIATION Procedure on page 23-41	Removes the target site from the DBA_REPCAT_TEMPLATE_SITES view.
DROP_TEMPLATE_OBJECT Procedure on page 23-42	Removes a template object from a specific deployment template.
DROP_TEMPLATE_PARM Procedure on page 23-44	Removes an existing template parameter from the DBA_REPCAT_TEMPLATE_PARMS view.
DROP_USER_AUTHORIZATION Procedure on page 23-45	Removes a user authorization entry from the DBA_REPCAT_USER_AUTHORIZATIONS view.
DROP_USER_PARM_VALUE Procedure on page 23-46	Removes a predefined user parameter value for a specific deployment template.
GET_RUNTIME_PARM_ID Function on page 23-47	Retrieves an identification to be used when defining a runtime parameter value.
INSERT_RUNTIME_PARMS Procedure on page 23-47	Defines runtime parameter values prior to instantiating a template.
INSTANTIATE_OFFLINE Function on page 23-49	Generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while offline.
INSTANTIATE_ONLINE Function on page 23-52	Generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while online.

Table 23–1 DBMS_REPCAT_RGT Package Subprograms **(Page 3 of 3)**

Subprogram	Description
LOCK_TEMPLATE_EXCLUSIVE Procedure on page 55	Prevents users from reading or instantiating the template when a deployment template is being updated or modified.
LOCK_TEMPLATE_SHARED Procedure on page 23-55	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 23–2 ALTER_REFRESH_TEMPLATE Procedure Parameters

Parameter	Description
refresh_template_name	The name of the deployment template that you want to alter.
new_owner	The name of the new deployment template owner. Do not specify a value to keep the current owner.
new_refresh_group_name	If necessary, use this parameter to specify a new refresh group name 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 23–3 ALTER_REFRESH_TEMPLATE Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
bad_public_template	The public_template parameter is specified incorrectly. The public_template parameter must be specified as a 'Y' for a public template or an 'N' for a private template.
dupl_refresh_template	A template with the specified name already exists. See the ALL_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 re-instantiate the deployment template to apply the changes.

Syntax

```
DBMS_REPCAT_RGT.ALTER_TEMPLATE_OBJECT (
   refresh_template_name       IN   VARCHAR2,
   object_name                 IN   VARCHAR2,
   object_type                 IN   VARCHAR2,
   new_refresh_template_name   IN   VARCHAR2 := '-',
   new_object_name             IN   VARCHAR2 := '-',
   new_object_type             IN   VARCHAR2 := '-',
   new_ddl_text                IN   CLOB  := '-',
   new_master_rollback_seg     IN   VARCHAR2 := '-',
   new_flavor_id               IN   NUMBER := -1e-130);
```

Parameters

Table 23–4 ALTER_TEMPLATE_OBJECT Procedure Parameters

Parameter	Description	
refresh_template_name	Deployment template name that contains the object that you want to alter.	
object_name	Name of the template object that you want to alter.	
object_type	Type of object that you want to alter.	
new_refresh_template_name	Name of the new deployment template 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.	
	Note: Do not set this parameter unless directed to do so by Oracle Support Services.	

Exceptions

Table 23–5 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 Support Services.
bad_object_type	Object type is specified incorrectly. See Table 23–4 for a list of valid object types.
miss_template_object	Template object name specified is invalid or does not exist.
dupl_template_object	New template name specified in the new_refresh_template_name parameter already exists.

Usage Notes

Because the ALTER_TEMPLATE_OBJECT procedure utilizes a CLOB, you 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 MATERIALIZED VIEW mview_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 => 'MVIEW_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 23–6 ALTER_TEMPLATE_PARM Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the parameter that you want to alter.
parameter_name	Name of the parameter that you want to alter.
new_refresh_template_name	Name of the deployment template that the specified parameter should be 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 23–7 ALTER_TEMPLATE_PARM Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_template_parm	Template parameter specified is invalid or does not exist.
dupl_template_parm	Combination of new_refresh_template_name and new_parameter_name already exists.

Usage Notes

Because the ALTER_TEMPLATE_PARM procedure utilizes a CLOB, you 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 materialized view 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 23–8 ALTER_USER_AUTHORIZATION Procedure Parameters

Parameter	Description
user_name	Name of the user whose authorization you want to alter.
refresh_template_name	Name of the deployment template that is currently assigned to the specified user that you want to alter.
new_user_name	Use this parameter to define a new user for this template authorization. Do not specify a value to keep the current user.
new_refresh_template_name	The deployment template that the specified user (either the existing or, if specified, the new user) is authorized to instantiate. Do not specify a value to keep the current deployment template.

Exceptions

Table 23–9 ALTER_USER_AUTHORIZATION Procedure Exceptions

Exception	Description
miss_user_authorization	The combination of user_name and refresh_template_name values specified does not exist in the DBA_REPCAT_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 ALL_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 materialized view environment uses assignment tables. Change a user parameter value to quickly and securely change the data set of a remote materialized view site.

> **See Also:** *Oracle9i 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 23–10 ALTER_USER_PARM_VALUE Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the user parameter value that you want to alter.
parameter_name	Name of the parameter that you want to alter.
user_name	Name of the user whose parameter value you want to alter.
new_refresh_template_name	Name of the deployment template that the specified user parameter value should be 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 23–11 ALTER_USER_PARM_VALUE Procedure Exceptions

Exception	Description
miss_refresh_template	Deployment template name specified is invalid or does not exist.
miss_template_parm	Template parameter specified is invalid or does not exist.
miss_user	User name specified for the user_name or new_user_name parameters is invalid or does not exist.
miss_user_parm_values	User parameter value specified does not exist.
dupl_user_parm_values	New user parameter specified already exists.

Usage Notes

Because the ALTER_USER_PARM_VALUE procedure utilizes a CLOB, you 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 temporary view.

The COMPARE_TEMPLATES function returns a number that you specify in the WHERE clause when querying the USER_REPCAT_TEMP_OUTPUT temporary view. 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 temporary view 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 23–12 COMPARE_TEMPLATES Function Parameters

Parameter	Description
source_template_name	Name of the first deployment template to be compared.
compare_template_name	Name of the second deployment template to be compared.

Exceptions

Table 23–13 COMPARE_TEMPLATES Function Exceptions

Exception	Description
miss_refresh_template	The deployment template name to be compared is invalid or does not exist.

Returns

Table 23–14 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 temporary view to view the discrepancies between the compared templates.

COPY_TEMPLATE Function

This function enables you to copy a deployment template and 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 23–15 COPY_TEMPLATE Function Parameters

Parameter	Description
old_refresh_template_name	Name of the deployment template to be copied.
new_refresh_template_name	Name of the new deployment template.
copy_user_authorizations	Specifies whether the template authorizations for the original template should be copied for the new deployment template. Valid values for this parameter are Y, N, and NULL. **Note**: All users must exist at the target database.
dblink	Optionally defines where the deployment template should be copied from (this is helpful to distribute deployment templates to other master sites). If none is specified, then the deployment template is copied from the local master site.

Exceptions

Table 23–16 COPY_TEMPLATE Function Exceptions

Exception	Description
miss_refresh_template	Deployment template name to be copied is invalid or does not exist.
dupl_refresh_template	Name of the new refresh template specified already exists.
bad_copy_auth	Value specified for the copy_user_authorization parameter is invalid. Valid values are Y, N, and NULL.

Returns

Table 23–17 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 materialized view 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 23–18 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 (enables 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, MATERIALIZED VIEW, and SNAPSHOT are not a valid object types for this function): SEQUENCE PROCEDURE INDEX FUNCTION TABLE PACKAGE VIEW PACKAGE BODY SYNONYM TRIGGER

Exceptions

Table 23–19 CREATE_OBJECT_FROM_EXISTING Function Exceptions

Exception	Description
miss_refresh_template	The specified refresh template name is invalid or missing. Query the DBA_REPCAT_REFRESH_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 23–20 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 enables 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 23–21 CREATE_REFRESH_TEMPLATE Function Parameters

Parameter	Description
owner	User name of the deployment template owner is specified with this parameter. If an owner is not specified, then the name of the user creating the template is automatically used.
refresh_group_name	Name of the refresh group that is created when this template is instantiated. All objects created by this template are assigned to the specified refresh group.
refresh_template_name	Name of the deployment template that you are creating. This name is referenced in all activities that involve this deployment template.
template_comment	User comments defined with this 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 23–22 CREATE_REFRESH_TEMPLATE Function Exceptions

Exception	Description
dupl_refresh_template	A template with the specified name already exists. See the ALL_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 23–23 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 materialized view site. In addition to adding materialized views, 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 23–24 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	TRIGGER
	SEQUENCE	DATABASE LINK
ddl_text	Contains the DDL that creates the object that you are adding to the template. Be sure to end your DDL with a semi-colon. You can use a colon (:) to create a template parameter for your template object. See Chapter 4, "Create a Deployment Template" for more information.	
	When you add a materialized view (snapshot) with a CREATE MATERIALIZED VIEW statement, make sure you specify the schema name of the owner of the master table in the materialized view query.	
master_rollback_seg	Specifies the name of the rollback segment to use when executing the defined object DDL at the remote materialized view site.	
flavor_id	This parameter is for internal use only.	
	Note: Do not set this parameter unless directed to do so by Oracle Support Services.	

Exceptions

Table 23–25 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 23–24 for a list of valid object types.
`dupl_template_object`	An object of the same name and type has already been added to the specified deployment template.

Returns

Table 23–26 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 MATERIALIZED VIEW mview_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 => 'mview_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 materialized view 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 23-10 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 23–27 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 23–28 CREATE_TEMPLATE_PARM Function Exceptions

Exception	Description
`miss_refresh_template`	The specified refresh template name is invalid or missing.
`dupl_template_parm`	A parameter of the same name has already been defined for the specified deployment template.

Returns

Table 23–29 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 23–30 CREATE_USER_AUTHORIZATION Function Parameters

Parameter	Description
user_name	Name of the user that you want to authorize to instantiate the specified template. Specify multiple users by separating user names with a comma (for example, 'john, mike, bob')
refresh_template_name	Name of the template that you want to authorize the specified user to instantiate.

Exceptions

Table 23–31 CREATE_USER_AUTHORIZATION Function Exceptions

Exception	Description
miss_user	User name supplied is invalid or does not exist.
miss_refresh_template	Refresh template name supplied is invalid or does not exist.
dupl_user_authorization	An authorization has already been created for the specified user and deployment template. See the ALL_REPCAT_USER_AUTHORIZATIONS view for a listing of template authorizations.

Returns

Table 23–32 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 23–33 CREATE_USER_PARM_VALUE Function Parameters

Parameter	Description
`refresh_template_name`	Specifies the name of the deployment template that contains the parameter you are creating a user 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 23–34 CREATE_USER_PARM_VALUE Function Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or missing.
dupl_user_parm_values	A parameter value for the specified user, parameter, and deployment template has already been defined. Query the DBA_REPCAT_USER_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 23–35 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 23–36 DELETE_RUNTIME_PARMS Procedure Parameters

Parameter	Description
runtime_parm_id	Specifies the identification 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 view for a list of deployment template parameters).

Exceptions

Table 23–37 DELETE_RUNTIME_PARMS Procedure Exceptions

Exception	Description
miss_template_parm	The specified deployment template parameter name is invalid or missing.

DROP_ALL_OBJECTS Procedure

This procedure allows the DBA to drop all objects or specific object types from a deployment template.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_OBJECTS (
   refresh_template_name   IN   VARCHAR2,
   object_type             IN   VARCHAR2 := NULL);
```

Parameters

Table 23–38 DROP_ALL_OBJECTS Procedure Parameters

Parameter	Description	
refresh_template_name	Name of the deployment template that contains the objects that you want to drop.	
object_type	If NULL, then all objects in the template are dropped. If an object type is specified, then only objects of that type are dropped. Objects of the following type may be specified:	
	SNAPSHOT	PROCEDURE
	INDEX	FUNCTION
	TABLE	PACKAGE
	VIEW	PACKAGE BODY
	SYNONYM	TRIGGER
	SEQUENCE	DATABASE LINK

Exceptions

Table 23–39 DROP_ALL_OBJECTS Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.
bad_object_type	Object type is specified incorrectly. See Table 23–38 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 23–40 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 23–41 DROP_ALL_TEMPLATE_PARMS Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_ALL_TEMPLATE_SITES Procedure

This procedure removes all entries from the DBA_REPCAT_TEMPLATE_SITES view, which keeps a record of sites that have instantiated a particular deployment template.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_TEMPLATE_SITES (
   refresh_template_name   IN   VARCHAR2);
```

Parameters

Table 23–42 DROP_ALL_TEMPLATE_SITES Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template that contains the sites that you want to drop.

Exceptions

Table 23–43 DROP_ALL_TEMPLATE_SITES Procedure Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name is invalid or does not exist.

DROP_ALL_TEMPLATES Procedure

This procedure removes all deployment templates at the site where the procedure is called.

Caution: This is a dangerous procedure that cannot be undone.

Syntax

```
DBMS_REPCAT_RGT.DROP_ALL_TEMPLATES;
```

Parameters

None

DROP_ALL_USER_AUTHORIZATIONS Procedure

This procedure enables 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 23–44 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 23–45 DROP_ALL_USER_AUTHORIZATIONS Procedure Exceptions

Exception	Description
`miss_refresh_template`	Specified deployment template name is invalid or does not exist.

DROP_ALL_USER_PARM_VALUES Procedure

This procedure drops user parameter values for a specific deployment template. This procedure is very flexible and enables you 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` and `user_name`	Drops all of the specified user parameters for the specified deployment template
`refresh_template_name` and `parameter_name`	Drops all user parameter values for the specified deployment template parameter
`refresh_template_name`, `parameter_name`, and `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 23–46 DROP_ALL_USER_PARMS Procedure Parameters

Parameter	Description
`refresh_template_name`	Name of the deployment template that contains the parameter values that you want to drop.
`user_name`	Name of the user whose parameter values you want to drop.
`parameter_name`	Template parameter that contains the values that you want to drop.

Exceptions

Table 23–47 DROP_ALL_USER_PARMS Procedure Exceptions

Exception	Description
`miss_refresh_template`	Deployment template name specified is invalid or does not exist.
`miss_user`	User name specified is invalid or does not exist.
`miss_user_parm_values`	Deployment template, user, and parameter combination does not exist in the `DBA_REPCAT_USER_PARM_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 23–48 DROP_REFRESH_TEMPLATE Procedure Parameters

Parameter	Description
refresh_template_name	Name of the deployment template to be dropped.

Exceptions

Table 23–49 DROP_REFRESH_TEMPLATE Procedure Exceptions

Exception	Description
miss_refresh_template	The deployment template name specified is invalid or does not exist. Query the DBA_REPCAT_REFRESH_TEMPLATES view for a list of deployment templates.

DROP_SITE_INSTANTIATION Procedure

This procedure drops a template instantiation at any target site. This procedure removes all related metadata at the master site and disables the specified site from refreshing its materialized views.

Syntax

```
DBMS_REPCAT_RGT.DROP_SITE_INSTANTIATION (
     refresh_template_name  IN   VARCHAR2,
     user_name              IN   VARCHAR2,
     site_name              IN   VARCHAR2);
```

Table 23–50 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 materialized view site. Query the ALL_REPCAT_TEMPLATE_SITES view to see the users that instantiated templates. See the ALL_REPCAT_TEMPLATE_SITES view on page 25-11 for more information.
site_name	Identifies the master site where you want to drop the specified template instantiation.

Exceptions

Table 23–51 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 materialized view 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 re-instantiate the deployment template to apply the changes.

Syntax

```
DBMS_REPCAT_RGT.DROP_TEMPLATE_OBJECT (
   refresh_template_name  IN   VARCHAR2,
   object_name            IN   VARCHAR2,
   object_type            IN   VARCHAR2);
```

Parameters

Table 23–52 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	TRIGGER
	SEQUENCE	DATABASE LINK

Exceptions

Table 23–53 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 useful 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 23–54 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 23–55 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 23-38

Syntax

```
DBMS_REPCAT_RGT.DROP_USER_AUTHORIZATION (
   refresh_template_name   IN   VARCHAR2,
   user_name               IN   VARCHAR2);
```

Parameters

Table 23–56 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 23–57 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 23–58 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 23–59 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 identification to be used when defining a runtime parameter value. All runtime parameter values are assigned to this identification and are also used during the instantiation process.

Syntax

```
DBMS_REPCAT_RGT.GET_RUNTIME_PARM_ID
  RETURN NUMBER;
```

Parameters

None

Returns

Table 23–60 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 identification to use when inserting a runtime parameter. This identification 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 23–61 INSERT_RUNTIME_PARMS Procedure Parameters

Parameter	Description
runtime_parm_id	The identification retrieved by the GET_RUNTIME_PARM_ID function. This identification is also used when instantiating the deployment template. Be sure to use the same identification 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 23–62 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 materialized view environment at the remote materialized view site while the materialized view site disconnected from the master (that is, while the materialized view site is offline). This generated script should be used at remote materialized view 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 materialized view site may be lengthy (depending on the amount of data that is populated to the new materialized views). 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 temporary view.

Note: This function is used to perform 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 23-49 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 materialized view). 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 23–63 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 identification used when creating the runtime parameters (the identification 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 materialized view groups created are owned by the default user PUBLIC. If false, then any materialized view groups created are owned by the user performing the instantiation.

Exceptions

Table 23–64 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 23–65 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` temporary `view` to retrieve the generated instantiation script.

INSTANTIATE_ONLINE Function

This function generates a script at the master site that is used to create the materialized view environment at the remote materialized view site while the materialized view site is connected to the master (that is, while the materialized view site is online). This generated script should be used at remote materialized view sites that are able to remain connected to the master site for an extended amount of time, as the instantiation process at the remote materialized view site may be lengthy (depending on the amount of data that is populated to the new materialized views). 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` temporary 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_OFFLINE` function, described in "INSTANTIATE_OFFLINE Function" on page 23-49 section.

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 23–66 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 identification used when creating the runtime parameters (the identification 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 materialized view groups created are owned by the default user PUBLIC. If false, then any materialized view groups created are owned by the user performing the instantiation.

Exceptions

Table 23–67 INSTANTIATE_ONLINE Function Exceptions

Exception	Description
miss_refresh_template	Specified deployment template name 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 23–68 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 temporary 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

24

DBMS_REPUTIL

DBMS_REPUTIL contains subprograms to generate shadow tables, triggers, and packages for table replication, as well as subprograms to generate wrappers for replication of standalone procedure invocations and packaged procedure invocations. This package is referenced only by the generated code.

This chapter discusses the following topics:

- Summary of DBMS_REPUTIL Subprograms

Summary of DBMS_REPUTIL Subprograms

Table 24–1 DBMS_REPUTIL Package Subprograms

Subprogram	Description
REPLICATION_OFF Procedure on page 24-3	Modifies tables without replicating the modifications to any other sites in the replication environment, or disables row-level replication when using procedural replication.
REPLICATION_ON Procedure on page 24-3	Re-enables replication of changes after replication has been temporarily suspended.
REPLICATION_IS_ON Function on page 24-4	Determines whether or not replication is running.
FROM_REMOTE Function on page 24-4	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 24-5	Determines the global database name of the local database (the global name is the returned value).
MAKE_INTERNAL_PKG Procedure on page 24-5	Synchronizes internal packages and tables in the replication catalog. **Note:** Do not execute this procedure unless directed to do so by Oracle Support Services.
SYNC_UP_REP Procedure on page 24-6	Synchronizes internal triggers and tables/materialized views in the replication catalog. **Note:** Do not execute this procedure unless directed to do so by Oracle Support Services.

REPLICATION_OFF Procedure

This procedure enables you to modify tables without replicating the modifications to any other sites in the replication environment. It also disables row-level replication when using procedural replication. In general, you should suspend replication activity for all master groups in your replication 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 replication 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 or materialized view in the replication catalog. If the table has replication support, then execute this procedure to create the internal package. If replication support does not exist, then this procedure destroys any related internal package. This procedure does not accept the storage table of a nested table.

Caution: Do not execute this procedure unless directed to do so by Oracle Support Services.

Syntax

```
DBMS_REPUTIL.MAKE_INTERNAL_PKG (
   canon_sname    IN   VARCHAR2,
   canon_oname    IN   VARCHAR2);
```

Parameters

Table 24–2 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 materialized view in the replication catalog. If the table or materialized view has replication support, then execute this procedure to create the internal replication trigger. If replication support does not exist, then this procedure destroys any related internal trigger. This procedure does not accept the storage table of a nested table.

Caution: Do not execute this procedure unless directed to do so by Oracle Support Services.

Syntax

```
DBMS_REPUTIL.SYNC_UP_REP (
   canon_sname    IN   VARCHAR2,
   canon_oname    IN   VARCHAR2);
```

Parameters

Table 24–3 SYNC_UP_REP Procedure Parameters

Parameter	Description
canon_sname	Schema containing the table or materialized view 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 materialized view to be synchronized. This parameter value must be canonically defined (capitalization must match object and must not be enclosed in double quotes).

Part IV

Replication Data Dictionary Reference

Part IV describes data dictionary views that provide information about your replication environment.

Part IV includes the following chapters:

- Chapter 25, "Replication Catalog Views"
- Chapter 26, "Replication Dynamic Performance Views"
- Chapter 27, "Deferred Transaction Views"
- Chapter 28, "Materialized View and Refresh Group Views"

25

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. This chapter contains these topics:

- Summary of Replication Catalog Views

Caution: Do not modify the replication catalog tables directly. Instead, use the procedures provided in the DBMS_REPCAT package.

See Also: Chapter 10, "Monitoring a Replication Environment"

Summary of 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.

As shown in Figure 25–1 on page 25-3, the replication catalog views are used by master sites and materialized view sites to determine such information as what objects are being replicated, where they are being replicated, and if any errors have occurred during replication. Table 25–1 on page 25-4 lists all of the replication catalog views.

Figure 25–1 Replication Catalog Views and Replicated Objects

Master 1 (m1)

employees

employee_id	last_name	department_id	salary
100	King	90	8340
101	Kochhar	90	6650
103	Hunold	60	9725
104	Ernst	60	5890

DBA_REPGROUP

GNAME	MASTER	STATUS	SCHEMA_COMMENT
RS	Y	NORMAL	

DBA_REPSITES

GNAME	DBLINK	MASTERDEF	SNAPMASTER	MASTER_ COMMENT
RS	M1	Y	NULL	
RS	M2	N	NULL	

DBA_REPOBJECT

GNAME	ONAME	TYPE	STATUS	OBJECT_ COMMENT
RS	EMPLOYEES	TABLE	VALID	
RS	EMPLOYEES$RP	PACKAGE	VALID	
RS	EMPLOYEES$RP	PACKAGE BODY	VALID	

Master 2 (m2)

employees

employee_id	last_name	department_id	salary
100	King	90	8340
101	Kochhar	90	6650
103	Hunold	60	9725
104	Ernst	60	5890

DBA_REPGROUP

GNAME	MASTER	STATUS	SCHEMA_COMMENT
RS	Y	NORMAL	

DBA_REPSITES

GNAME	DBLINK	MASTERDEF	SNAPMASTER	MASTER_ COMMENT
RS	M1	Y	NULL	
RS	M2	N	NULL	

DBA_REPOBJECT

GNAME	ONAME	TYPE	STATUS	OBJECT_ COMMENT
RS	EMPLOYEES	TABLE	VALID	
RS	EMPLOYEES$RP	PACKAGE	VALID	
RS	EMPLOYEES$RP	PACKAGE BODY	VALID	

Materialized View 1 (department 90)

employees

employee_id	last_name	department_id	salary
100	King	90	8340
101	Kochhar	90	6650

SELECT employee_id, last_name, department_id,
salary FROM employees@m1 WHERE department_id = 90;

DBA_REPGROUP

GNAME	MASTER	STATUS	SCHEMA_COMMENT
RS	N	NULL	

DBA_REPSITES

GNAME	DBLINK	MASTERDEF	SNAPMASTER	MASTER_ COMMENT
RS	M1	Y	Y	
RS	M2	N	N	

DBA_REPOBJECT

GNAME	ONAME	TYPE	STATUS	OBJECT_ COMMENT
RS	EMPLOYEES	SNAPSHOT	VALID	

Materialized View 2 (department 60)

employees

employee_id	last_name	department_id	salary
103	Hunold	60	9725
104	Ernst	60	5890

SELECT employee_id, last_name, department_id,
salary FROM employees@m2 WHERE department_id = 60;

DBA_REPGROUP

GNAME	MASTER	STATUS	SCHEMA_COMMENT
RS	N	NULL	

DBA_REPSITES

GNAME	DBLINK	MASTERDEF	SNAPMASTER	MASTER_ COMMENT
RS	M1	Y	N	
RS	M2	N	Y	

DBA_REPOBJECT

GNAME	ONAME	TYPE	STATUS	OBJECT_ COMMENT
RS	EMPLOYEES	SNAPSHOT	VALID	

Table 25–1 Replication Catalog Views

ALL_ Views	DBA_ Views	USER_ Views
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_AUTHORIZATION
ALL_REPCAT_USER_PARM_VALUES	DBA_REPCAT_USER_PARM_VALUES	USER_REPCAT_USER_PARM_VALUES
ALL_REPCATLOG	DBA_REPCATLOG	USER_REPCATLOG
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
	DBA_REPEXTENSIONS	
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
	DBA_REPSITES_NEW	

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 or public).

When the DBA adds materialized view definitions to the template container, the DBA references the appropriate REFRESH_TEMPLATE_NAME. Any materialized views 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 (for example, :region). Templates that use parameters allow for greater flexibility during the template instantiation process (that is, in defining data sets specific for a materialized view 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: FUNCTION SNAPSHOT INDEX SYNONYM INDEXTYPE TABLE OPERATOR TRIGGER PACKAGE TYPE PACKAGE BODY TYPE BODY PROCEDURE VIEW

Column	Datatype	NULL	Description
DDL_NUM	NUMBER	NOT NULL	Indicates the order in which to execute the DDL statements stored in the DDL_TEXT column when multiple DDL statements are used to create the 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 column 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, then 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: *Oracle9i Application Developer's Guide - Fundamentals* for more information on using cursors. Also, see *Oracle9i 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 25-6

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 the DEFAULT_PARM_VALUE column 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, then 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: *Oracle9i Application Developer's Guide - Fundamentals* for more information on using cursors. Also, see *Oracle9i 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.

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 materialized view site of the deployment template.
REPAPI_SITE_NAME	VARCHAR2(128)		This column is intended for use in a future release of Oracle.
STATUS	VARCHAR2(10)		Displays the status of the deployment template at the target materialized view site: 0 = Not Installed 1 = Installed -1 = Installed with errors
INSTANTIATION_DATE	DATE		Displays when the template was instantiated. Is NULL if the template has not yet been instantiated.

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 listed 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_AUTHORIZATION 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 columns 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, then 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:** *Oracle9i Application Developer's Guide - Fundamentals* for more information on using cursors. Also, see *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for more information on using the DBMS_LOB package and LOBs in general.

ALL_REPCATLOG

Contains the interim status of any asynchronous administrative requests and any error messages generated at each master site. All messages encountered while executing a request are eventually transferred to the ALL_REPCATLOG view at the master site that originated the request. If an administrative request completes without error, then ultimately all traces of this request are removed from the ALL_REPCATLOG view. This view contains administrative requests and error messages that are accessible to the current user.

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, then indicates which master site 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: FUNCTION SNAPSHOT INDEX SYNONYM INDEXTYPE TABLE OPERATOR TRIGGER PACKAGE TYPE PACKAGE BODY TYPE BODY PROCEDURE VIEW
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 replication group.

ALL_REPCOLUMN

Lists the replicated columns for the tables accessible to the current user.

If the table contains a column object, then this view displays a placeholder for the type and one row for each type attribute. If the table contains a nested table, then then this view displays the storage table for the nested table as an independent table. If a table is an object table, then this view displays the hidden object identifier column.

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, either SNAPSHOT or TABLE.
CNAME	VARCHAR2(4000)		The name of the replicated column.
ID	NUMBER		The ID number of the replicated column.
POS	NUMBER		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(30)		Displays the column type. For user-defined types, displays the user-defined type name.
CTYPE_TOID	RAW(16)		If user-defined type, displays the object identifier (OID) of the type. Otherwise, this field is NULL.

Column	Datatype	NULL	Description
CTYPE_OWNER	VARCHAR2(30)		If user-defined type, displays the owner of a user-defined type. Otherwise, this field is NULL.
CTYPE_HASHCODE	VARCHAR2(34)		If user-defined type, displays the type's hashcode. Otherwise, this field is NULL.
CTYPE_MOD	VARCHAR2(3)		Displays REF for REF columns. Otherwise, this field is NULL.
DATA_LENGTH	VARCHAR2(40)		Displays the length of the column in bytes.
DATA_PRECISION	VARCHAR2(40)		Displays the column precision in terms of decimal digits for NUMBER columns or binary digits for FLOAT columns.
DATA_SCALE	VARCHAR2(40)		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.
TOP	VARCHAR2(30)		Displays the top column for an attribute in a column object. For example, in the oe.customers table, cust_address is a column object and street_address is one of its attributes. For the street_address attribute, cust_address is the TOP column. For built-in datatypes, this field is NULL
CHAR_LENGTH	NUMBER		Displays the length of the column in characters. This value only applies to the following datatypes: ▪ CHAR ▪ VARCHAR2 ▪ NCHAR ▪ NVARCHAR2
CHAR_USED	VARCHAR2(1)		B indicates that the column uses BYTE length semantics. C indicates that the column uses CHAR length semantics. NULL indicates that the datatype is not any of the following: ▪ CHAR ▪ VARCHAR2 ▪ NCHAR ▪ NVARCHAR2

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 site.
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.
DDL_NUM	NUMBER(38)		Indicates the order in which to execute the DDL statements stored in the TEXT column when multiple DDL statements are used.

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, either PACKAGE, PACKAGE BODY, TRIGGER, or INTERNAL PACKAGE.
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 replication groups that are accessible to the current user. The members of each replication group are listed in a different view: ALL_REPOBJECT.

Related Views:

- DBA_REPGROUP describes all of the replication groups in the database that are being replicated.
- USER_REPGROUP describes all of the replication 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 materialized view 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 replication group.
FNAME	VARCHAR2(30)		Flavor name.
RPC_PROCESSING_DISABLED	VARCHAR2(1)		N indicates that this site can receive and apply deferred remote procedure calls (RPCs). Y indicates that this site can not receive and apply deferred remote procedure calls (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 replication groups. Shows only those replication groups accessible to the current user.

Related Views:

- DBA_REPGROUP_PRIVILEGES contains information about the users who are registered for privileges in all the replication groups in the database.
- USER_REPGROUP_PRIVILEGES contains information about the users who are registered for privileges in the replication 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 replication group.
CREATED	DATE	NOT NULL	Displays the date that the replication 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 replication key column(s) accessible to the current user in each table.

The replication key column(s) is an alternate column or group of columns, instead of the primary key, used to determine which columns of a table to compare when using row-level replication. You can set the replication key columns using the SET_COLUMNS procedure in the DBMS_REPCAT package.

The following types of columns cannot be replication key columns:

- LOB or LOB attribute of a column object
- Collection or collection attribute of a column object
- REF
- An entire column object

See Also: "SET_COLUMNS Procedure" on page 20-105

Related Views:

- DBA_REPKEY_COLUMNS describes the replication key column(s) in each table in the database.
- USER_REPKEY_COLUMNS describes the replication 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(4000)		Replication key column(s) in the table.

ALL_REPOBJECT

Contains information about the objects in each replication group accessible to the current user. An object can belong to only one replication group. A replication group can span multiple schemas.

Related Views:

- DBA_REPOBJECT contains information about the objects in each replication group in the database.
- USER_REPOBJECT contains information about the objects owned by the current user in each replication group.

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: FUNCTION SNAPSHOT INDEX SYNONYM INDEXTYPE TABLE OPERATOR TRIGGER PACKAGE TYPE PACKAGE BODY TYPE BODY PROCEDURE VIEW
STATUS	VARCHAR2(10)		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, then Oracle updates its status to DROPPED before deleting the row from the ALL_REPOBJECT view.

Column	Datatype	NULL	Description
GENERATION_STATUS	VARCHAR2(9)		Specifies 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 replication group to which the object belongs.
MIN_COMMUNICATION	VARCHAR2(1)		If Y then use minimum communication for an update. If N then send all old and all new values for an update.
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.
NESTED_TABLE	VARCHAR2(1)		If Y then the replicated object is the storage table of a nested table. If N then the replicated object is not a storage table.

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(4000)		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 replication 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 in 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 is CHAR.
VARCHAR2_VALUE	VARCHAR2(4000)		The value of the priority group member, if DATA_TYPE is VARCHAR2.
NUMBER_VALUE	NUMBER		The value of the priority group member, if DATA_TYPE is NUMBER.
DATE_VALUE	DATE		The value of the priority group member, if DATA_TYPE is DATE.
RAW_VALUE	RAW(2000)		The value of the priority group member, if DATA_TYPE is RAW.
GNAME	VARCHAR2(30)	NOT NULL	The name of the replication group.
NCHAR_VALUE	NCHAR(500)		The value of the priority group member, if DATA_TYPE is NCHAR.
NVARCHAR2_VALUE	VARCHAR2(1000)		The value of the priority group member, if DATA_TYPE is 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 replication group accessible to the current user.

Related Views:

- DBA_REPPRIORITY_GROUP describes the priority group or site priority group defined for each replication group in the database.
- USER_REPPRIORITY_GROUP describes the priority group or site priority group defined for each replication group owned by the current user.

Column	Datatype	NULL	Description
SNAME	VARCHAR2(30)	NOT NULL	The name of the replicated schema. Obsolete in 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 replication 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. These operations may have resulted from a call to a stored procedure or procedure wrapper, or may have been issued against a table directly. This view shows objects accessible to the current user.

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: FUNCTION PROCEDURE INDEXTYPE SNAPSHOT OPERATOR TABLE PACKAGE TYPE PACKAGE BODY TYPE BODY
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 replication group accessible to the current user.

Related Views:

- DBA_REPSITES lists the members of each replication group in the database.
- USER_REPSITES lists the members of each replication group owned by the current user.

Column	Datatype	NULL	Description
GNAME	VARCHAR2(30)	NOT NULL	The name of the replication group.
DBLINK	VARCHAR2(128)	NOT NULL	The database link to a master site for this replication group.
MASTERDEF	VARCHAR2(1)		Indicates which of the DBLINKs is the master definition site.
SNAPMASTER	VARCHAR2(1)		Used by materialized view 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 site.
MY_DBLINK	VARCHAR2(1)		Used to detect problems after import. If Y then the DBLINK is the global name.

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 or 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 25-5.

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 25-6.

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 25-8.

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. Its columns are the same as those in ALL_REPCAT_TEMPLATE_SITES on page 25-11.

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 25-12.

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 25-13.

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 site that originated the request. If an administrative request completes without error, then 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 25-15.

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 25-17.

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 25-19.

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 25-20.

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 25-21.

DBA_REPEXTENSIONS

The DBA_REPEXTENSIONS view contains information about current operations that are adding new master sites to a master group without quiescing the master group.

See Also: "Adding New Master Sites Without Quiescing the Master Group" on page 7-4 for information about the procedure that adds new master sites to a replication environment

Column	Datatype	NULL	Description
EXTENSION_ID	RAW(16)	NOT NULL	The identifier for a current pending request to add master databases to a master group without quiesce.
REQUEST	VARCHAR2(15)		Extension request type. Currently, the only possible value is ADD_NEW_MASTERS, which indicates a request to add new master sites to a master group without quiescing.
MASTERDEF	VARCHAR2(128)		The global name of the master definition site of the master groups to which new master sites are being added.
EXPORT_REQUIRED	VARCHAR2(3)		YES indicates that one or more new master sites will be added using export/import of either the entire database or at the table level. NO indicates that all new master sites will be added using change-based recovery.
REPCATLOG_ID	NUMBER		Identifier of replication catalog records related to a replication extension, on which the master definition site is waiting. This value is only meaningful at the master definition site.

Column	Datatype	NULL	Description
EXTENSION_STATUS	VARCHAR2(13)		Status of each replication extension. This value is only meaningful at the master definition site. The possible values are: READY: The extension request has been created and is ready. STOPPING: The new master sites have been added to the master group and the master definition site is attempting to stop propagation from existing masters to new master sites and to the master definition site. EXPORTING: The propagation of deferred transactions has been stopped from existing master sites to new master sites and to the master definition site. The master definition site is waiting for the export to finish. INSTANTIATING: The DBMS_REPCAT.RESUME_PROPAGATION_TO_MDEF procedure has been invoked (if export was used), and the master definition site is waiting for the new masters to instantiate. ERROR: An error occurred during the execution of this extension request.
FLASHBACK_SCN	NUMBER		The system change number (SCN) that must be used during export or change-based recovery when the new master sites are added. The new master sites must be consistent with the SCN listed.
BREAK_TRANS_TO_MASTERDEF	VARCHAR2(3)		This value is meaningful only if EXPORT_REQUIRED is TRUE. If BREAK_TRANS_TO_MASTERDEF is TRUE, then existing masters may continue to propagate their deferred transactions to the master definition site for replication groups that are not adding master sites. Deferred transactions for replication groups that are adding master sites cannot be propagated until the export completes. If BREAK_TRANS_TO_MASTERDEF is FALSE, then existing masters cannot propagate any deferred transactions to the master definition site.

Column	Datatype	NULL	Description
BREAK_TRANS_TO_NEW_MASTERS	VARCHAR2(3)		If BREAK_TRANS_TO_NEW_MASTERS is TRUE, then existing master sites may continue to propagate deferred transactions to the new master sites for replication groups that are not adding master sites. If BREAK_TRANS_TO_NEW_MASTERS is FALSE, then propagation of deferred transaction queues to the new masters is disabled.
PERCENTAGE_FOR_CATCHUP_MDEF	NUMBER		This value is meaningful only if BREAK_TRANS_TO_MASTERDEF is TRUE. The percentage of propagation resources that should be used for catching up propagation to the master definition site.
CYCLE_SECONDS_MDEF	NUMBER		This value is meaningful when PERCENTAGE_FOR_CATCHUP_MDEF is both meaningful and is a value between 10 and 90, inclusive. In this case, propagation to the master definition site alternates between replication groups that are not being extended and replication groups that are being extended, with one push to each during each cycle. This value indicates the length of the cycle in seconds.
PERCENTAGE_FOR_CATCHUP_NEW	NUMBER		This value is meaningful only if BREAK_TRANS_TO_NEW_MASTERS is TRUE. The percentage of propagation resources that should be used for catching up propagation to new master sites.
CYCLE_SECONDS_NEW	NUMBER		This value is meaningful when PERCENTAGE_FOR_CATCHUP_NEW is both meaningful and is a value between 10 and 90, inclusive. In this case, propagation to a new master alternates between replication groups that are not being extended and replication groups that are being extended, with one push to each during each cycle. This value indicates the length of the cycle in seconds.

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 25-22.

DBA_REPGROUP

The DBA_REPGROUP view describes all of the replication groups in the database. The members of each replication 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 25-23.

DBA_REPGROUP_PRIVILEGES

The DBA_REPGROUP_PRIVILEGES view contains information about the users who are registered for privileges in replication groups. Shows all replication groups in the database. Its columns are the same as those in ALL_REPGROUP_PRIVILEGES on page 25-24.

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 25-25.

DBA_REPKEY_COLUMNS

The DBA_REPKEY_COLUMNS view describes the replication key column(s) in each table in the database. Its columns are the same as those in ALL_REPKEY_COLUMNS on page 25-26.

DBA_REPOBJECT

The DBA_REPOBJECT view contains information about the objects in each replication group in the database. An object can belong to only one replication group. A replication group can span multiple schemas. Its columns are the same as those in ALL_REPOBJECT on page 25-27.

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. Its columns are the same as those in ALL_REPPARAMETER_COLUMN on page 25-29.

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 replication 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 25-30.

DBA_REPPRIORITY_GROUP

The DBA_REPPRIORITY_GROUP view describes the priority group or site priority group defined for each replication group in the database. Its columns are the same as those in ALL_REPPRIORITY_GROUP on page 25-32.

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. These operations may have resulted from a call to a stored procedure or procedure wrapper, or may have been issued against a table directly. This view shows all objects in the database. Its columns are the same as those in ALL_REPPROP on page 25-33.

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 25-34.

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 25-35.

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 25-36.

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 25-37.

DBA_REPSITES

The DBA_REPSITES view lists the members of each replication 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 site.
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 25-38.

DBA_REPSITES_NEW

The DBA_REPSITES_NEW view lists the new replication sites that you plan to add to your replication environment.

See Also: "Adding New Master Sites Without Quiescing the Master Group" on page 7-4 for information about the procedure that adds new master sites to a replication environment

Column	Datatype	NULL	Description
EXTENSION_ID	RAW(16)	NOT NULL	The identifier for a current pending request to add master databases to a master group without quiesce.
GOWNER	VARCHAR2(30)	NOT NULL	The name of the user who owns the master group.
GNAME	VARCHAR2(30)	NOT NULL	The name of the master group.
DBLINK	VARCHAR2(128)	NOT NULL	The database link for a new master site.
FULL_INSTANTIATION	VARCHAR2(1)		Y indicates that the new database in DBLINK is to be added using full database export/import or change-based recovery. N indicates that the new database in DBLINK is to be added using object-level export/import.
MASTER_STATUS	VARCHAR2(13)		The instantiation status of a new master site. This value is only meaningful at the master definition site. The possible values are: READY: The new master site is ready. INSTANTIATING: The new master site is in the process of being instantiated. INSTANTIATED: The new master has been instantiated and is being prepared for replication activity. That is, the DBMS_REPCAT.PREPARE_INSTANTIATED_MASTER procedure has been run. PREPARED: The propagation of deferred transactions is enabled from the new master site to other prepared masters, to existing masters, and to the master definition site. The new master is now prepared to participate in the replication environment.

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 or 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 25-5.

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 25-6.

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 25-8.

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. Its columns are the same as those in ALL_REPCAT_TEMPLATE_SITES on page 25-11.

USER_REPCAT_USER_AUTHORIZATION

The USER_REPCAT_USER_AUTHORIZATION 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 25-12.

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 view 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 25-13.

USER_REPCATLOG

The USER_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 USER_REPCATLOG view at the master site that originated the request. If an administrative request completes without error, then ultimately all traces of this request are removed from the USER_REPCATLOG view.

This view contains asynchronous administrative requests and error messages that are owned by the current user. Its columns are the same as those in ALL_REPCATLOG on page 25-15.

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 25-17.

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 25-19.

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 25-20.

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 25-21.

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 25-22.

USER_REPGROUP

The USER_REPGROUP view describes all of the replication groups owned by the current user. The members of each replication 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 25-23.

USER_REPGROUP_PRIVILEGES

The USER_REPGROUP_PRIVILEGES view contains information about the users who are registered for privileges in replication groups. Shows only those replication groups owned by the current user. Its columns are the same as those in ALL_REPGROUP_PRIVILEGES on page 25-24.

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 25-25.

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 replication key column(s) in each table owned by the current user. Its columns are the same as those in ALL_REPKEY_COLUMNS on page 25-26.

USER_REPOBJECT

The USER_REPOBJECT view contains information about the objects owned by the current user in each replication group. An object can belong to only one replication group. A replication group can span multiple schemas. Its columns are the same as those in ALL_REPOBJECT on page 25-27.

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. Its columns are the same as those in ALL_REPPARAMETER_COLUMN on page 25-29.

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 replication 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 25-30.

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 replication group owned by the current user. Its columns are the same as those in ALL_REPPRIORITY_GROUP on page 25-32.

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. These operations may have resulted from a call to a stored procedure or procedure wrapper, or may have been issued against a table directly. This view shows objects owned by the current user. Its columns are the same as those in ALL_REPPROP on page 25-33.

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 25-34.

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 25-35.

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 25-36.

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 25-37.

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 replication group owned by the current user. Its columns are the same as those in ALL_REPSITES on page 25-38.

26

Replication Dynamic Performance Views

All Oracle installations include the dynamic performance views, often referred to as V$ views, described in this chapter. These views are used by master sites and materialized view sites to determine such information as which materialized views are being refreshed currently and statistics about the deferred transaction queue.

This chapter describes the following views:

- V$MVREFRESH
- V$REPLPROP
- V$REPLQUEUE

See Also: Chapter 10, "Monitoring a Replication Environment"

V$MVREFRESH

Contains information about the materialized views currently being refreshed.

Column	Datatype	Description
SID	NUMBER	Session identifier.
SERIAL#	NUMBER	Session serial number, which is used to identify uniquely a session's objects. Guarantees that session-level commands are applied to the correct session objects if the session ends and another session begins with the same session ID.
CURRMVOWNER	VARCHAR2(31)	Owner of the materialized view currently being refreshed. The materialized view resides in this user's schema.
CURRMVNAME	VARCHAR2(31)	Name of the materialized view currently being refreshed.

V$REPLPROP

Contains information about the parallel propagation currently in progress at the replication site. Use this view to determine which transactions are currently being propagated, the number of calls propagated in each transaction, and the current activity of the parallel propagation slave processes or parallel propagation coordinator process.

Note: This view only contains data when deferred transactions are being pushed using parallel propagation at the current site. The `parallelism` parameter must be set to `1` or higher in the `DBMS_DEFER_SYS.PUSH` function for a push to use parallel propagation. Otherwise, the push uses serial propagation, and no data appears in this view during the push.

Column	Datatype	Description
SID	NUMBER	Session identifier.
SERIAL#	NUMBER	Session serial number. Used to identify uniquely a session's objects. Guarantees that session-level commands are applied to the correct session objects if the session ends and another session begins with the same session ID.
NAME	VARCHAR2(71)	Replication Parallel Prop Slave *n* indicates that the slave process is active, either waiting, pushing deferred transactions, purging metadata, or creating an error transaction. Replication Parallel Prop Coordinator indicates that the coordinator process is active, either waiting, sleeping, or scheduling slaves to perform operations. The Replication Parallel Prop Coordinator reads transactions from the deferred transaction queue and assigns them to the Replication Parallel Prop Slaves. Then, the slaves propagate the transactions to the destination site. When the slaves push transactions in a push session, the slaves remain active until the push session completes, even if there are no more transactions to push.
DBLINK	VARCHAR2(128)	Database link on which this replication session is propagating.
STATE	VARCHAR2(12)	WAIT indicates that either the slave or coordinator process is waiting for an event (that is, a message). SLEEP indicates that the coordinator process is sleeping for the duration of the delay_seconds setting. You set delay_seconds with the SCHEDULE_PUSH procedure in the DBMS_DEFER_SYS package. PUSH indicates that the slave process is pushing transactions from the deferred transaction queue to the remote site. PURGE indicates that the slave process is purging metadata related to successfully applied transactions from the remote site. CREATE ERROR indicates that the slave process is creating an error transaction. In this case, an error or a conflict occurred while the slave was pushing deferred transactions to the remote site. SCHEDULE TXN indicates that the coordinator process is determining the order that transactions are applied and assigning slave processes to execute the transactions.
XID	VARCHAR2(22)	If the session is a slave session, then indicates the transaction id of the transaction that the slave is currently propagating.
SEQUENCE	NUMBER	If the process is a slave process, then the sequence number of the calls propagated in the current operation, if relevant. Each transaction must process one or more calls, and the value of SEQUENCE starts at zero and increases as each call is processed. So, the SEQUENCE value shows the call that is currently being processed in each transaction. This value increases until the slave has processed all of the calls in a transaction.

V$REPLQUEUE

Contains statistics about the replication deferred transactions queue. All values are stored since the start of the current database instance.

Column	Datatype	Description
TXNS_ENQUEUED	NUMBER	Number of transactions enqueued in the deferred transactions queue.
CALLS_ENQUEUED	NUMBER	Number of calls enqueued into the deferred transactions queue.
TXNS_PURGED	NUMBER	Number of transactions purged from the deferred transactions queue.
LAST_ENQUEUE_TIME	DATE	Date when the last transaction was enqueued into the deferred transaction queue. NULL if no transactions have been enqueued into the deferred transaction queue since the instance started.
LAST_PURGE_TIME	DATE	Date when the last transaction was purged from the deferred transaction queue. NULL if no transactions have been purged from the deferred transaction queue since the instance started.

27

Deferred Transaction Views

Oracle provides several views for you to use when 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.

This chapter describes the following views:

- DEFCALL
- DEFCALLDEST
- DEFDEFAULTDEST
- DEFERRCOUNT
- 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.

See Also: Chapter 10, "Monitoring a Replication Environment"

DEFCALL

Records all deferred remote procedure calls.

For calls placed in the queue using asynchronous replication, Oracle uses null compression for column objects and object tables that contain three or more consecutive nulls. Therefore, this view may show fewer attributes than the total number of attributes in a column object and fewer columns than the total number for an object table. For example, null compression may cause a column object with eight attributes to show only five attributes.

Null compression does not apply to error transactions.

Column	Datatype	NULL	Description
CALLNO	NUMBER		The 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, then 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 remote procedure calls (RPCs).

Column	Datatype	NULL	Description
ID	RAW(16)	NOT NULL	Identifier of the LOB parameter.
DEFERRED_TRAN_ID	VARCHAR2(22)		Transaction ID for deferred remote procedure calls (RPCs) with this LOB parameter.
BLOB_COL	BLOB(4000)		The binary LOB parameter.
CLOB_COL	CLOB(4000)		The character LOB parameter.
NCLOB_COL	NCLOB(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 and also includes propagation statistics. The propagation statistics are for propagation of deferred transactions from the current site to the site specified in the DBLINK column.

To clear the propagation statistics for a remote site and start fresh, use the CLEAR_PROP_STATISTICS procedure in the DBMS_DEFER_SYS package.

See Also: "CLEAR_PROP_STATISTICS Procedure" on page 14-5

Column	Datatype	NULL	Description
DBLINK	VARCHAR2(128)	NOT NULL	Fully qualified path name to the master 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.

Column	Datatype	NULL	Description
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.
CATCHUP	RAW(16)	NOT NULL	The extension identifier associated with a new master site that is being added to a master group without quiescing the master group. If there is no extension identifier for a master site, then the value is 00.
TOTAL_TXN_COUNT	NUMBER		Total combined number of successful transactions and error transactions.
AVG_THROUGHPUT	NUMBER		The average number of transactions per second that are propagated using parallel propagation. The transactions include both successfully applied transactions and error transactions created on the remote site. Time that has elapsed when the propagation coordinator is inactive (sleeping) is included in the calculation.
AVG_LATENCY	NUMBER		If the transaction is successfully applied at the remote site, then the average number of seconds between the first call of a transaction on the current site and the confirmation that the transaction was applied at the remote site. The first call begins when the user makes the first data manipulation language (DML) change, not when the transaction is committed. If the transaction is an error transaction, then the average number of seconds between the first call of a transaction on the current site and the confirmation that the error transaction is committed on the remote site.
TOTAL_BYTES_SENT	NUMBER		Total number of bytes sent, including replicated data and metadata.
TOTAL_BYTES_RECEIVED	NUMBER		Total number of bytes received in propagation confirmation messages.
TOTAL_ROUND_TRIPS	NUMBER		Total number of network round trips completed to replicate data. A round trip is one or more consecutively sent messages followed by one or more consecutively received messages. So, if site A sends 20 messages to site B and then site B sends one message to site A, then that is that one round trip.

Column	Datatype	NULL	Description
TOTAL_ADMIN_COUNT	NUMBER		Total number of administrative requests sent to maintain information about transactions applied at the receiving site. The receiving site is the site specified in the DBLINK column. This special administration is only required for parallel propagation.
TOTAL_ERROR_COUNT	NUMBER		Total number of unresolved conflicts for which a remote error was created.
TOTAL_SLEEP_TIME	NUMBER		Total number of seconds the propagation coordinator was inactive (sleeping). You control the amount of time that the propagation coordinator sleeps using the delay_seconds parameter in the DBMS_DEFER_SYS.PUSH function.
DISABLED_INTERNALLY_SET	VARCHAR2(1)		This value is relevant only if DISABLED is Y. If DISABLED_INTERNALLY_SET is Y then propagation to destination was set to disabled internally by Oracle for propagation synchronization when adding a new master site to a master group without quiescing the master group. Oracle will enable propagation automatically at a later time. If DISABLED_INTERNALLY_SET is N then propagation was not disabled internally.

DEFTRAN

Records all deferred transactions in the deferred transactions queue at the current site.

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 change number (SCN) 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 each deferred transaction in the deferred transactions queue at the current site.

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 change number (SCN) of the originating transaction.
DBLINK	VARCHAR2(128)	NOT NULL	The fully qualified database name of the destination database.

28

Materialized View and Refresh Group Views

This chapter describes the following data dictionary views, which provide information about materialized views and materialized view refresh groups.

ALL_ Views	DBA_ Views	USER_ Views
ALL_BASE_TABLE_MVIEWS	DBA_BASE_TABLE_MVIEWS	USER_BASE_TABLE_MVIEWS
	DBA_MVIEW_LOG_FILTER_COLS	
ALL_MVIEW_LOGS	DBA_MVIEW_LOGS	USER_MVIEW_LOGS
ALL_MVIEW_REFRESH_TIMES	DBA_MVIEW_REFRESH_TIMES	USER_MVIEW_REFRESH_TIMES
ALL_MVIEWS	DBA_MVIEWS	USER_MVIEWS
ALL_REFRESH	DBA_REFRESH	USER_REFRESH
ALL_REFRESH_CHILDREN	DBA_REFRESH_CHILDREN	USER_REFRESH_CHILDREN
	DBA_REGISTERED_MVIEW_GROUPS	
ALL_REGISTERED_MVIEWS	DBA_REGISTERED_MVIEWS	USER_REGISTERED_MVIEWS

See Also: Chapter 10, "Monitoring a Replication Environment"

ALL_BASE_TABLE_MVIEWS

ALL_BASE_TABLE_MVIEWS lists information about all materialized views using materialized view logs accessible to the current user. A materialized view log can be created for a master or base table or a master materialized view. Query this view at the master site or master materialized view site to show one row for each materialized view using a materialized view log.

Related Views:

- DBA_BASE_TABLE_MVIEWS describes all materialized views using materialized view logs in the database.
- USER_BASE_TABLE_MVIEWS describes all materialized views using materialized view logs owned by the current user.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Schema in which the master table or master materialized view was created
MASTER	VARCHAR2(30)	NOT NULL	Name of the master table or master materialized view
MVIEW_LAST_REFRESH_TIME	DATE	NOT NULL	The date when the materialized view based on the master was last refreshed
MVIEW_ID	NUMBER(38)		Unique identifier of the materialized view that is based on the master

ALL_MVIEW_LOGS

ALL_MVIEW_LOGS describes all materialized view logs accessible to the current user.

Related Views:

- DBA_MVIEW_LOGS describes all materialized view logs in the database.
- USER_MVIEW_LOGS describes all materialized view logs owned by the current user.

Column	Datatype	NULL	Description
LOG_OWNER	VARCHAR2(30)	NOT NULL	Owner of the materialized view log
MASTER	VARCHAR2(30)	NOT NULL	Name of the master table or master materialized view whose changes are logged
LOG_TABLE	VARCHAR2(30)	NOT NULL	Name of the table where the changes to the master table or master materialized view are logged
LOG_TRIGGER	VARCHAR2(30)		Obsolete with the release of Oracle8*i* and higher. 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
OBJECT_ID	VARCHAR2(3)		If YES, records object identifier information in an object table
FILTER_COLUMNS	VARCHAR2(3)		If YES, records filter columns
SEQUENCE	VARCHAR2(3)		If YES, records the sequence value, which provides additional ordering information
INCLUDE_NEW_VALUES	VARCHAR2(3)		If YES, records both old and new values. If NO, records old values, but does not record new values.

ALL_MVIEW_REFRESH_TIMES

ALL_MVIEW_REFRESH_TIMES describes refresh times of materialized views accessible to the current owner.

Related Views:

- DBA_MVIEW_REFRESH_TIMES describes refresh times of all materialized views in the database.
- USER_MVIEW_REFRESH_TIMES describes refresh times of all materialized views owned by the current user.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Schema in which the materialized view was created
NAME	VARCHAR2(30)	NOT NULL	Name of the materialized view
MASTER_OWNER	VARCHAR2(30)		Owner of the materialized view's master table or master materialized view
MASTER	VARCHAR2(30)		Name of the materialized view's master table or master materialized view
LAST_REFRESH	DATE		The date of the last refresh

ALL_MVIEWS

ALL_MVIEWS describes all materialized views accessible to the current user.

Related views:

- DBA_MVIEWS describes all materialized views in the database.
- USER_MVIEWS describes all materialized views owned by the current user.

See Also:

- *Oracle9i Replication* for more information on materialized views to support replication
- *Oracle9i Data Warehousing Guide* for more information on materialized views to support data warehousing

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Schema in which the materialized view was created
MVIEW_NAME	VARCHAR2(30)	NOT NULL	Name of the materialized view
CONTAINER_NAME	VARCHAR2(30)	NOT NULL	Name of the container in which the materialized view's data is stored. Normally, this is the same as MVIEW_NAME. For materialized views created in releases before 8.1 (or under pre-8.1 compatibility mode), Oracle attaches the 6-byte prefix SNAP$_. If MVIEW_NAME has more than 19 bytes, Oracle truncates the name to 19 bytes and may add a 4-byte sequence number as a suffix to produce a non ambiguous CONTAINER_NAME.
QUERY	LONG		The query that defines the materialized view
QUERY_LEN	NUMBER(38)		Length in bytes of the defining query
UPDATABLE	VARCHAR2(1)		Whether the materialized view is updatable (Y | N)
UPDATE_LOG	VARCHAR2(30)		For updatable materialized views, the filename of the update log
MASTER_ROLLBACK_SEG	VARCHAR2(30)		Rollback segment for the master site or master materialized view site
MASTER_LINK	VARCHAR2(128)		Database link for the master site or master materialized view site
REWRITE_ENABLED	VARCHAR2(1)		Whether rewrite is enabled (Y | N)

Column	Datatype	NULL	Description
REWRITE_ CAPABILITY	VARCHAR2(9)		Whether the materialized view is eligible for rewrite, and if so, what rules must be followed: ■ NONE: The materialized view cannot be used for rewrite, because rewrite is disallowed or prevented. ■ TEXTMATCH: The defining query of the materialized view contained restrictions on the use of query rewrite. ■ GENERAL: The defining query of the materialized view contained no restrictions on the use of query rewrite, so Oracle can apply any rewrite rule that is supported.
REFRESH_MODE	VARCHAR2(6)		Refresh mode of the materialized view: ■ DEMAND: Oracle refreshes this materialized view whenever an appropriate refresh procedure is called. ■ COMMIT: Oracle refreshes this materialized view when a transaction on one of the materialized view's masters commits. ■ NEVER: Oracle never refreshes this materialized view.
REFRESH_METHOD	VARCHAR2(8)		Default method used to refresh the materialized view. (Can be overridden through the API.) ■ COMPLETE (C): The materialized view is completely refreshed from the masters. ■ FORCE (?): Oracle performs a fast refresh if possible, otherwise a complete refresh. ■ FAST (F): Oracle performs an incremental refresh applying changes that correspond to changes in the masters since the last refresh. ■ NEVER (N): The user specified that Oracle should not refresh this materialized view.
BUILD_MODE	VARCHAR2(9)		How the materialized view was populated during creation: ■ IMMEDIATE: Populated from the masters during creation. ■ DEFERRED: Not populated during creation. Must be explicitly populated later by the user. ■ PREBUILT: Populated with an existing table during creation. The relationship of the contents of this prebuilt table to the materialized view's masters is unknown to Oracle.

Column	Datatype	NULL	Description
FAST_REFRESHABLE	VARCHAR2(18)		Whether or not the materialized view is eligible for incremental (fast) refresh. Oracle calculates this value statically, based on the materialized view definition query. ■ NO: The materialized view is not fast refreshable, and hence is complex. ■ DIRLOAD: Fast refresh is supported only for direct loads. ■ DML: Fast refresh is supported only for DML operations. ■ DIRLOAD_DML: Fast refresh is supported for both direct loads and DML operations. ■ DIRLOAD_LIMITEDDML: Fast refresh is supported for direct loads and a subset of DML operations.
LAST_REFRESH_ TYPE	VARCHAR2(8)		The method used for the most recent refresh: ■ COMPLETE: The most recent refresh was complete. ■ FAST: The most recent refresh was fast (incremental). ■ NA: the materialized view has not yet been refreshed (for example, if it was created DEFERRED).
LAST_REFRESH_ DATE	DATE		Date on which the materialized view was most recently refreshed. Blank if not yet populated.
STALENESS	VARCHAR2(9)		The relationship between the contents of the materialized view and the contents of the materialized view's masters. ■ FRESH: The materialized view is a read-consistent view of the current state of its masters. ■ STALE: The materialized view is out of date because one or more of its masters has changed. If the materialized view was FRESH before it became STALE, then it is a read-consistent view of a former state of its masters. ■ UNUSABLE: The materialized view is not a read-consistent view of its masters from any point in time. ■ UNKNOWN: Oracle does not know whether the materialized view is in a read-consistent view of its masters from any point in time. (This is the case for materialized views created on prebuilt tables.) ■ UNDEFINED: The materialized view has remote masters. The concept of staleness is not defined for such materialized views.
AFTER_FAST_ REFRESH	VARCHAR2(9)		Specifies the staleness value that will occur if a fast refresh is applied to this materialized view. Its values are the same as for the STALENESS column, plus the value NA, which is used when fast refresh is not applicable to this materialized view.

Column	Datatype	NULL	Description
COMPILE_STATE	VARCHAR2(13)		Validity of the materialized view with respect to the object(s) upon which it depends: ■ VALID: The materialized view has been validated without error, and no object upon which it depends has changed since the last validation. ■ NEEDS_COMPILE: Some object upon which the materialized view depends has changed (other than normal DML changes). An ALTER MATERIALIZED VIEW...COMPILE statement is required to validate this materialized view. ■ ERROR: The materialized view has been validated with one or more errors.
USE_NO_INDEX	VARCHAR2(1)		Y if the materialized view was created using the USING NO INDEX clause, which suppresses the creation of the default index. N if the materialized view was created with the default index.

ALL_REFRESH

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 removed
PUSH_DEFERRED_RPC	VARCHAR2(1)		Y or N; if Y, then push changes from materialized view to masters before refresh
REFRESH_AFTER_ERRORS	VARCHAR2(1)		Y or N; if Y, then proceed with refresh despite error when pushing deferred remote procedure calls (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 the 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; if Y, then the job is broken and will never run
PURGE_OPTION	NUMBER(38)		The method for purging the transaction queue after each push. 1 is quick purge option. 2 is precise purge option.
PARALLELISM	NUMBER(38)		The level of parallelism for transaction propagation. Zero indicates serial propagation. 1 or higher indicates parallel propagation.
HEAP_SIZE	NUMBER(38)		The size of the heap

ALL_REFRESH_CHILDREN

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 the refresh group
IMPLICIT_DESTROY	VARCHAR2(1)		Y or N; if Y, then destroy the refresh group when its last item is removed
PUSH_DEFERRED_RPC	VARCHAR2(1)		Y or N; if Y, then push changes from materialized view to masters before refresh
REFRESH_AFTER _ERRORS	VARCHAR2(1)		Y or N; if Y, then proceed with refresh despite error when pushing deferred remote procedure calls (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 the 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; if Y, then the job is broken and will never run
PURGE_OPTION	NUMBER(38)		The method for purging the transaction queue after each push. 1 is quick purge option. 2 is precise purge option.
PARALLELISM	NUMBER(38)		The level of parallelism for transaction propagation. Zero indicates serial propagation. 1 or higher indicates parallel propagation.
HEAP_SIZE	NUMBER(38)		The size of the heap

ALL_REGISTERED_MVIEWS

ALL_REGISTERED_MVIEWS describes all materialized views registered at a master site or master materialized view site that are accessible to the current user.

Related Views:

- DBA_REGISTERED_MVIEWS describes all registered materialized views in the database.
- USER_REGISTERED_MVIEWS describes all registered materialized views owned by the current user.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the materialized view
NAME	VARCHAR2(30)	NOT NULL	Name of the materialized view
MVIEW_SITE	VARCHAR2(128)	NOT NULL	Global name of the materialized view site
CAN_USE_LOG	VARCHAR2(3)		YES if the materialized view can use a materialized view log, NO if the materialized view is too complex to use a log.
UPDATABLE	VARCHAR2(3)		Whether the materialized view is updatable. YES if it is, and NO if it is not. If NO, then the materialized view is read only.
REFRESH_METHOD	VARCHAR2(11)		Whether the materialized view uses primary key, rowids, or object identifiers for fast refresh
MVIEW_ID	NUMBER(38)		Identifier for the materialized view used by the masters for fast refresh
VERSION	VARCHAR2(26)		Oracle version of materialized view **Note:** Oracle9*i* materialized views show Oracle8.
QUERY_TXT	LONG		The query that defines the materialized view

DBA_BASE_TABLE_MVIEWS

DBA_BASE_TABLE_MVIEWS lists information about all materialized views using materialized view logs in the database. A materialized view log can be created for a master or base table or a master materialized view. Query this view at the master site or master materialized view site to show one row for each materialized view using a materialized view log. Its columns are the same as those in ALL_BASE_TABLE_MVIEWS on page 28-2.

DBA_MVIEW_LOG_FILTER_COLS

ALL_MVIEW_LOG_FILTER_COLS lists all columns (excluding primary key columns) being logged in the materialized view logs.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(30)	NOT NULL	Owner of the master being logged
NAME	VARCHAR2(30)	NOT NULL	Name of the master being logged
COLUMN_NAME	VARCHAR2(30)	NOT NULL	Column being logged

DBA_MVIEW_LOGS

ALL_MVIEW_LOGS describes all materialized view logs in the database. Its columns are the same as those in ALL_MVIEW_LOGS on page 28-3.

DBA_MVIEW_REFRESH_TIMES

DBA_MVIEW_REFRESH_TIMES lists refresh times of all materialized views in the database. Its columns are the same as those in ALL_MVIEW_REFRESH_TIMES on page 28-4.

DBA_MVIEWS

DBA_MVIEWS describes all materialized views in the database. Its columns are the same as those in ALL_MVIEWS on page 28-5.

DBA_REFRESH

DBA_REFRESH describes all refresh groups in the database. Its columns are the same as those in ALL_REFRESH on page 28-9.

DBA_REFRESH_CHILDREN

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 28-10.

DBA_REGISTERED_MVIEW_GROUPS

DBA_REGISTERED_MVIEW_GROUPS lists all the registered materialized view groups at the master site or master materialized view site.

Column	Datatype	NULL	Description
NAME	VARCHAR2(30)		Name of the materialized view replication group
MVIEW_SITE	VARCHAR2(128)		Site of the materialized view replication group
GROUP_COMMENT	VARCHAR2(80)		Description of the materialized view replication group
VERSION	VARCHAR2(8)		Oracle version of the materialized view replication group **Note:** Oracle9*i* materialized view groups show Oracle8.
FNAME	VARCHAR2(30)		Name of the flavor of the materialized view group
OWNER	VARCHAR2(30)		Owner of the materialized view replication group

DBA_REGISTERED_MVIEWS

DBA_REGISTERED_MVIEWS describes all registered materialized views in the database. Its columns are the same as those in ALL_REGISTERED_MVIEWS on page 28-11.

USER_BASE_TABLE_MVIEWS

USER_BASE_TABLE_MVIEWS lists information about all materialized views using materialized view logs owned by the current user. A materialized view log can be created for a master or base table or a master materialized view. Query this view at the master site or master materialized view site to show one row for each materialized view using a materialized view log. Its columns are the same as those in ALL_BASE_TABLE_MVIEWS on page 28-2.

USER_MVIEW_LOGS

USER_MVIEW_LOGS lists all materialized view logs owned by the current user. Its columns are the same as those in ALL_MVIEW_LOGS on page 28-3.

USER_MVIEW_REFRESH_TIMES

USER_MVIEW_REFRESH_TIMES describes refresh times of materialized views owned by the current user. Its columns are the same as those in ALL_MVIEW_REFRESH_TIMES on page 28-4.

USER_MVIEWS

USER_MVIEWS describes all materialized views owned by the current user. Its columns are the same as those in ALL_MVIEWS on page 28-5.

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 28-9.

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 28-10.

USER_REGISTERED_MVIEWS

USER_REGISTERED_MVIEWS describes all registered materialized views owned by the current user. Its columns are the same as those in ALL_REGISTERED_MVIEWS on page 28-11.

Part V

Appendixes

Part V includes the following appendixes:

- Appendix A, "Security Options"
- Appendix B, "User-Defined Conflict Resolution Methods"

A

Security Options

This appendix contains these topics:

- Security Setup for Multimaster Replication
- Security Setup for Materialized View 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, in 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.

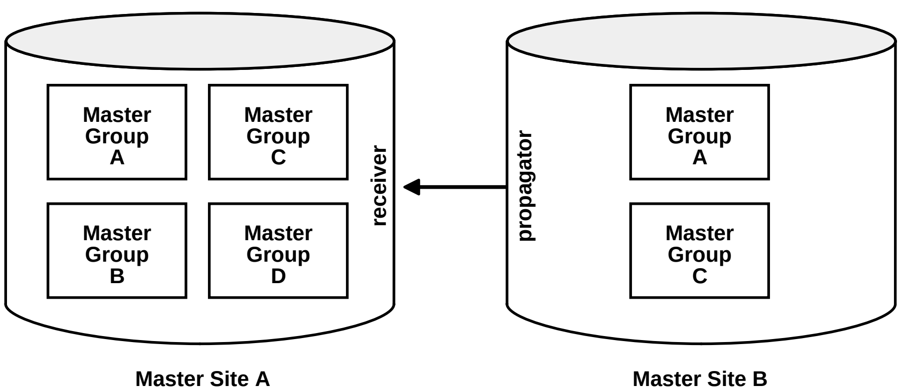
Trusted Compared With Untrusted Security

In addition to the different types of 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.

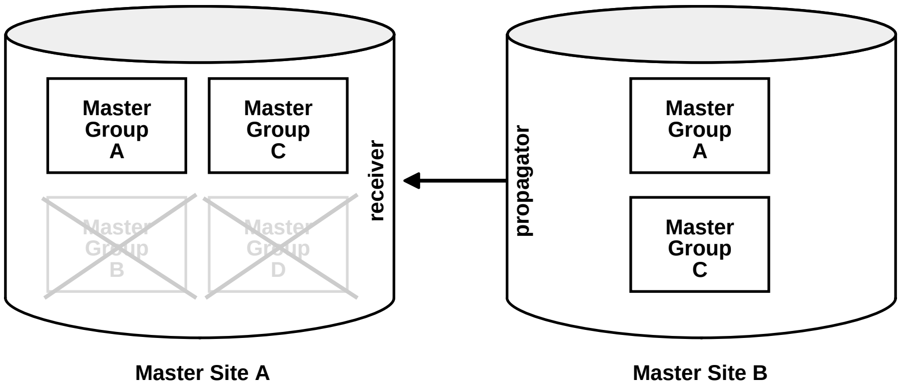
Regardless of the security model used, Oracle automatically grants the appropriate privileges for objects as they are added to or removed from a replication environment.

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, then 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 "REGISTER_USER_REPGROUP Procedure" on page 21-5 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 Materialized View Replication

Nearly all users should find it easiest to use the Replication Manager Setup Wizard when configuring materialized view 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 materialized view 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 or master materialized view site. You may need to create additional users, or assign additional privileges to users at the associated master site or master materialized view site.

In addition to end users who will be accessing replicated objects, there are three special categories of "users" at a materialized view 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 materialized views from the associated master tables or master materialized views.

Typically, a single user performs each of these functions. However, there may be situations where you need different users performing these functions. For example, materialized views may be created by a materialized view site administrator and refreshed by another end user.

Table A–2 describes the privileges needed to create and maintain a materialized view site.

Table A–2 Required Materialized View Site User Accounts

User	Privileges
Materialized view site replication administrator	DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA
Propagator	DBMS_DEFER_SYS.REGISTER_PROPAGATOR
Refresher	CREATE ANY MATERIALIZED VIEW ALTER ANY MATERIALIZED VIEW

In addition to creating the appropriate users at the materialized view site, you may need to create additional users at the associated master site or master materialized view site, as well. Table A–3 on on page A-11 describes the privileges need by master site or master materialized view site users to support a new materialized view site.

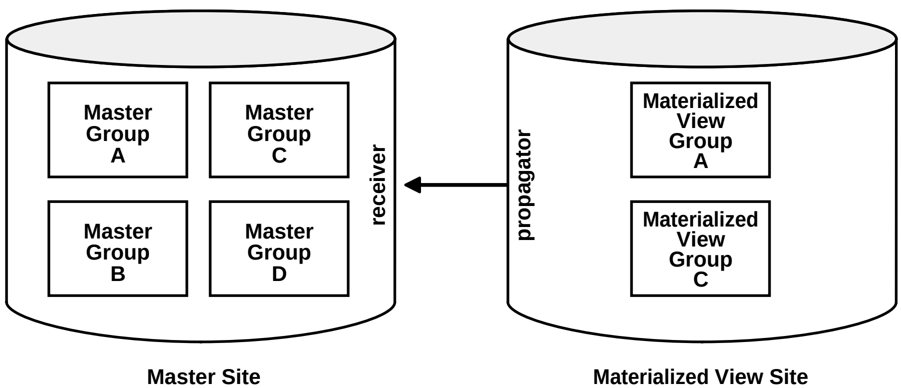
Trusted Compared With Untrusted Security

In addition to the different users at the master site or master materialized view 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 materialized view administrator have access to all local replication groups. The receiver and proxy materialized view administrator perform database activities at the local master site or master materialized view site on behalf of the propagator and materialized view administrator, respectively, at the remote materialized view site. Therefore, the propagator and materialized view administrator at the remote materialized view site also have access to all replication groups at the master site or master materialized view site. Remember that a single receiver is used for all incoming transactions.

For example, consider the scenario in Figure A–3. Even though Materialized View Groups A and C exist at the materialized view site (based on Master Groups A and C at the Master Site), the propagator and materialized view 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.

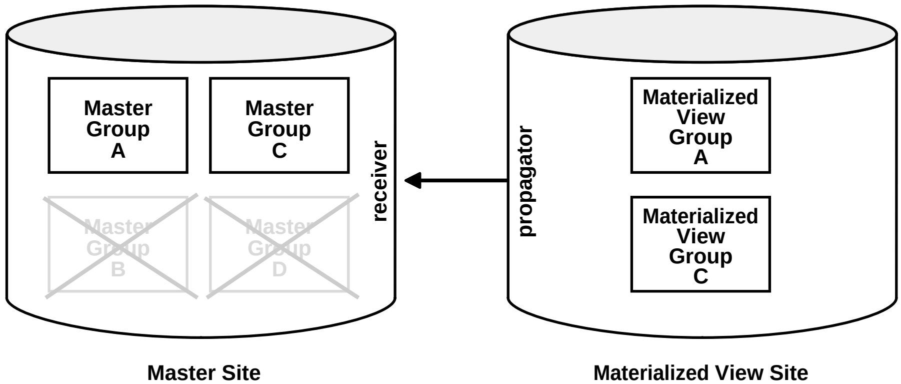
Regardless of the security model used, Oracle automatically grants the appropriate privileges for objects as they are added to or removed from a replication environment.

Figure A–3 Trusted Security: Materialized View Replication



Untrusted security assigns only the privileges to the proxy materialized view administrator and receiver that are required to work with specified replication groups. The propagator and materialized view administrator, therefore, can only access these specified replication groups at the Master Site. Figure A–4 illustrates an untrusted security model with materialized view replication. Because the Materialized View Site contains Materialized View Groups A and C, access to only Master Groups A and C are required. Using untrusted security does not allow the propagator or the materialized view administrator at the Materialized View Site to access Master Groups B and D at the Master Site.

Figure A–4 Untrusted Security: Materialized View Replication



Typically, materialized view 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 materialized view site and it is recommended that you use the untrusted security model with materialized view sites.

One reason you might choose to use a trusted security model is if your materialized view site is considered a master site in every way (security, constant network connectivity, resources) but is a materialized view only because of data subsetting requirements. Remember that row and column subsetting are not supported in a multimaster configuration.

Use the appropriate API calls listed for the proxy materialized view administrator and receiver in Table A–3 to assign the different users the appropriate privileges.

Table A–3 Required Master Site or Master Materialized View Site User Accounts

User	Privileges

proxy materialized view site administrator	See "REGISTER_USER_REPGROUP Procedure" on page 21-5 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 "REGISTER_USER_REPGROUP Procedure" on page 21-5 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 or master materialized views and materialized view logs

After creating the accounts at both the materialized view and associated master sites or master materialized view sites, you need to create the following private database links, including username and password, from the materialized view site to the master site or master materialized view site:

- From the materialized view replication administrator to the proxy materialized view replication administrator.
- From the propagator to the receiver.
- From the refresher to the proxy refresher.
- From the materialized view owner to the master site or master materialized view site for refreshes.

Assuming you have designated a single user account to act as materialized view administrator, propagator, and refresher, you must create one link for each materialized view site for those functions. You do not need a link from the master site or master materialized view site to the materialized view site.

After creating these links, you must call `DBMS_DEFER_SYS.SCHEDULE_PUSH` and `DBMS_DEFER_SYS.SCHEDULE_PURGE` at the materialized view site to define how frequently you want to propagate your deferred transaction queue to the associated master site or master materialized view site, and how frequently you wish to purge this queue. You must also call `DMBS_REFRESH.REFRESH` at the materialized view site to schedule how frequently to pull changes from the associated master site or master materialized view site.

B

User-Defined Conflict Resolution Methods

This appendix describes how to build user-defined conflict resolution methods and user-defined conflict notification methods. This appendix contains these topics:

- User-Defined Conflict Resolution Methods
- User-Defined Conflict Notification Methods
- Viewing Conflict Resolution Information

User-Defined Conflict Resolution Methods

Oracle enables 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, then 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, then Oracle stops evaluation of the method, and, if any other methods were provided to resolve the conflict with a later sequence number, then 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.
- Attributes of column objects cannot be defined as column parameters for user-defined conflict resolution methods.

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, then 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`, then 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`, then 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, then 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, then 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, then 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: "Creating Conflict Avoidance Methods for Delete Conflicts" on page 6-31

Multitier Materialized Views and User-Defined Conflict Resolution Methods

When you use user-defined conflict resolution methods with multitier materialized views, the information about these methods is pulled down to the master materialized view sites automatically. This information is stored in the data dictionary at the master materialized view site. However, the user-defined conflict resolution methods themselves cannot be pulled down from the master site. Therefore, you must re-create these methods at the master materialized view site.

See Also:

- "Viewing Conflict Resolution Information" on page B-12 for information about the data dictionary views that store information about user-defined conflict resolution methods
- *Oracle9i Replication* for more information about conflict resolution and multitier materialized views

Restrictions for User-Defined Conflict Resolution Methods

The following sections describe restrictions for user-defined conflict resolution methods.

SQL Statement Restrictions for User-Defined Conflict Resolution Methods

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`)

Column Subsetting Restrictions for User-Defined Conflict Resolution Methods

Avoid subsetting the columns in a column group when you create updatable multitier materialized views. Column subsetting excludes columns that are in master tables or master materialized views from a materialized view based on these masters. You do this by specifying certain select columns in the `SELECT` statement during materialized view creation.

When you use conflict resolution with multitier materialized views, you cannot define the conflict resolution methods at the materialized view site. Conflict resolution methods are always pulled down from the master site. Therefore, if you subset the columns in a column group that has a user-defined conflict resolution applied to it, the conflict resolution method will not be able to find all of the columns in the column group at a master materialized view site. When this happens, the conflict resolution method returns the following error:

```
ORA-23460 missing value for column in resolution method
```

For example, consider a case where the `job_id`, `salary`, and `commission_pct` columns in the `hr.employees` table are part of a column group name `employees_cg1` that has a user-defined conflict resolution method applied to it at the master site `hq.world`. To protect the privacy of your sales staff, you create a level 1 updatable materialized view that uses column subsetting to exclude the `salary` and `commission_pct` columns at the `ca.us` office. When you create this materialized view at the `ca.us` office, the conflict resolution method is pulled down from `hq.world`. You then create an updatable multitier materialized view at the `sf.ca` office based on the level 1 materialized view at the `ca.us` office.

Given this replication environment, if a conflict arises for a `job_id` value at the level 1 materialized view at the `ca.us` office, then the conflict resolution method fails to find the `salary` and `commission_pct` columns and returns the `ORA-23460` error mentioned previously.

> **See Also:** *Oracle9i Replication* for more information about column subsetting

Examples of 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, then use new values.
-- If new is null or new < curr, then use current values.
-- If both are null, then 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, then old = 0.
-- If new is null, then new = 0.
-- If curr is null, then 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 "EXECUTE_DDL Procedure" on page 20-80. 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, then Oracle rolls back the entire transaction, including any updates to a notification table. If notification is necessary independent of transactions, then you can design a notification mechanism to use the Oracle `DBMS_PIPES` package.

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 index-by 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 replication environment. Each view has three versions: USER_*, ALL_*, SYS.DBA_*. The available views 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 25, "Replication Catalog Views"

Index

A

C

D

E

F

G

I

J

L

M

N

O

V

W