Oracle8*i*

Replication

Release 8.1.5

February 1999

A67791-01

Oracle8i Replication, Release 8.1.5

A67791-01

Primary Author: William Creekbaum

Contributors: Alan Downing, Harry Sun, Al Demers, Maria Pratt, Gordon Smith, Jim Stamos, Curt Elsbernd, Pat McElroy, Ruth Baylis, and others.

Graphic Designer: Valarie Moore.

Contents

Send Us Your Comments

Oracle8i Replication, Release 8.1.5

A67791-01

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

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

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

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

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

If you have problems with the software, please contact your local Oracle World Wide Support Center.

Preface

This Preface contains the following topics:

- Overview of the Oracle8i Replication Manual.
- Audience.
- How The Oracle8i Replication Manual Is Organized.
- Conventions Used in This Manual.
- Your Comments Are Welcome.

Oracle8*i* Replication contains information relating to both Oracle8*i* Server and Oracle8*i* Server Enterprise Edition. Some features documented in this manual are available only with the Oracle8*i* Server Enterprise Edition. Furthermore, some of these features are only available if you have purchased a particular option, such as the Objects Option.

For information about the differences between Oracle8*i* Server and the Oracle8*i* Server Enterprise Edition, please refer to *Getting to Know Oracle8i.* This text describes features common to both products, and features that are only available with the Oracle8*i* Server Enterprise Edition or a particular option.

Overview of the Oracle8*i* Replication Manual

This manual describes Oracle8*i* Server replication capabilities. To use the synchronous replication facility, you must have installed Oracle's advanced replication option. Basic replication (read-only and updateable snapshots) is standard Oracle distributed functionality. Procedural replication requires PL/SQL and the advanced replication option.

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

- Read-only snapshots.
- Advanced replication facility.
  - Updateable snapshots.
  - Multi-master replication.
  - Snapshot site replication.
  - Conflict resolution.
- Administering a replicated environment.
- Advanced techniques.
- Troubleshooting.
- Using job queues.
- Deferred transactions.

Audience

This manual is written for application developers and database administrators who develop and maintain advanced Oracle8*i* distributed systems.

Knowledge Assumed of the Reader

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

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

- *Oracle8i Concepts*.
- *Oracle8i Administrator's Guide*.
- *Oracle8i Distributed Database Systems*.
- *PL/SQL User's Guide and Reference* (if you plan to use procedural replication).

How The Oracle8*i* Replication Manual Is Organized

This manual contains nine chapters and two appendices as described below.

Chapter 1, "Understanding Replication"
Introduces the concepts and terminology of Oracle replication.

Chapter 2, "Using Multimaster Replication"
Describes how to create and maintain a multi-master replicated environment using the Oracle advanced replication facility.

Chapter 3, "Snapshot Concepts & Architecture"
Describes the concepts and architecture of snapshot replication. This chapter will additionally discuss the prerequisites of building a snapshot environment.

Chapter 4, "Creating Snapshots with Deployment Templates"
Describes how to use Oracle Deployment Templates to mass deploy a snapshot environment to remote snapshot sites.

Chapter 5, "Directly Create Snapshot Environment"
Chapter 5 shows you how to build a snapshot environment will directly connected to a remote snapshot site. In addition to creating a new environment, you will also learn how to manage the snapshot site.

Chapter 6, "Conflict Resolution"
Describes how to use Oracle-supplied conflict resolution methods to resolve conflicts resulting from dynamic or shared ownership of data in replicated environments.

Chapter 7, "Administering a Replicated Environment"
Describes how to detect and resolve unresolved replication errors as well as how to monitor successful conflict resolution.

Chapter 8, "Advanced Techniques"
Describes advanced replication techniques, including how to do the following: create your own conflict resolution routines, implement fail-over sites, implement token passing, use procedural replication, and manage deletes.

Chapter 9, "Using Deferred Transactions"
Describes how to build transactions for deferred execution at remote locations.

Appendix A, "New Features"
Briefly describes the new features of this release and refers to sections of this document having more information.

Appendix B, "Migration and Compatibility"
Discusses the compatibility issues between release 8 of the advanced replication option and previous releases.

Conventions Used in This Manual

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

Special Notes

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

Note: Indicates special or auxiliary information.

Attention: Indicates items of particular importance about matters requiring special attention or caution.

Additional Information: Indicates where to get more information.

Text of the Manual

The following sections describe conventions used this manual.

UPPERCASE Characters

Uppercase text is used to call attention to command keywords, object names, parameters, filenames, and so on.

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

Italicized Characters

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

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

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

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

Code Examples

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

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

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

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

Uppercase words in example statements indicate the keywords within Oracle SQL. When issuing statements, however, keywords are not case sensitive.

Lowercase words in example statements indicate words supplied only for the context of the example. For example, lowercase words may indicate the name of a table, column, or file.

Your Comments Are Welcome

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

Oracle8*i* Server Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065

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

infodev@us.oracle.com

1 Understanding Replication

This chapter explains the basic concepts and terminology for the Oracle replication features.

Note: If you are using Trusted Oracle, see your Trusted Oracle documentation for information about using replication in that environment.

What Is Replication?

Replication is the process of copying and maintaining database objects in multiple databases that make up a distributed database system. Changes applied at one site are captured and stored locally before being forwarded and applied at each of the remote locations. Replication provides user with fast, local access to shared data, and protects availability of applications because alternate data access options exist. Even if one site becomes unavailable, users can continue to query or even update the remaining locations.

Replication Objects, Groups, and Sites

The following sections explain the basic components of a replication system, including replication sites, replication groups, and replication objects.

Replication Objects

A *replication object* is a database object existing on multiple servers in a distributed database system. Oracle's replication facility enables you to replicate tables and supporting objects such as views, database triggers, packages, indexes, and synonyms. `SCOTT.EMP` and `SCOTT.BONUS` illustrated in Figure 1–1 are examples of replication objects.

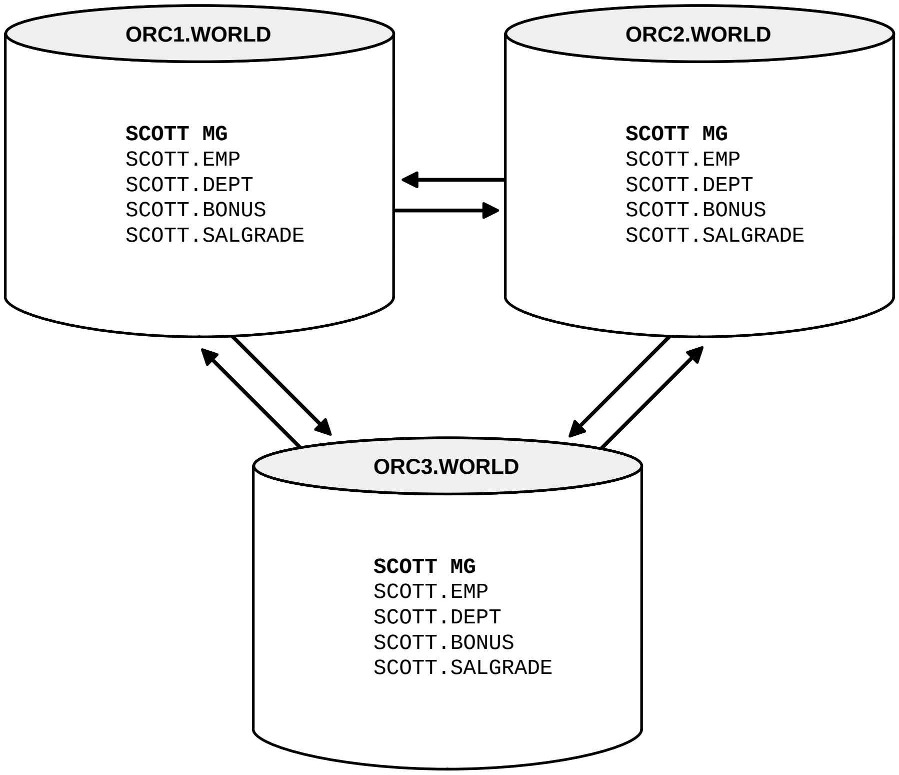
Replication Groups

In a replication environment, Oracle manages replication objects using *replication groups*. By organizing related database objects within a replication group, it is easier to administer many objects together. Typically, you create and use a replication group to organize the schema objects necessary to support a particular database application. That is not to say that replication groups and schemas must correspond with one another. Objects in a replication group can originate from several database schemas and a schema can contain objects that are members of different replication groups. The restriction is that a replication object can be a member of only one group.

Note: Read-only snapshots are not required to belong to a snapshot group, nor are they required to be based on a master table that is part of a master group.

In a multimaster replication environment, the replication groups are called *master groups*. Corresponding master groups at different sites must contain the same set of replication objects (see "Replication Objects" on page 1-2). Figure 1–1 illustrates that master group "SCOTT_MG" contains an exact replica of the replicated objects at each master site.

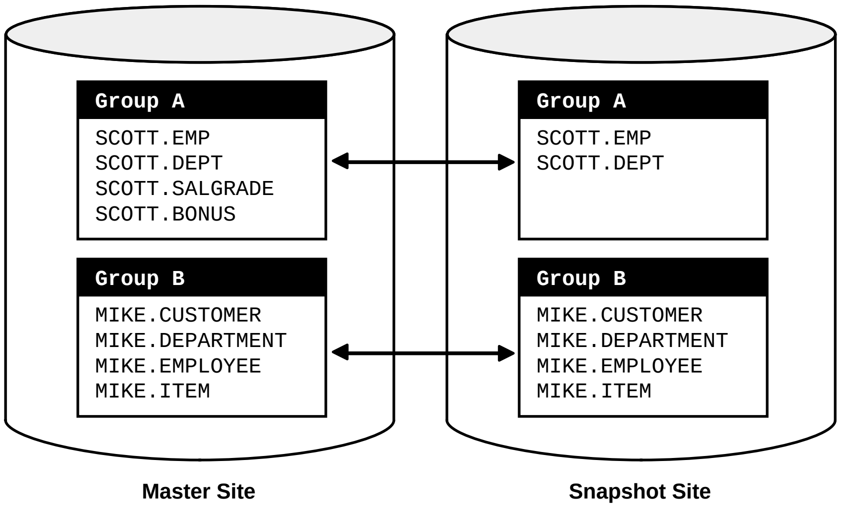
Figure 1–1 Master Group SCOTT_MG contains same replication objects at all sites.



At a snapshot site, organization is maintained using a snapshot group. A *snapshot group* maintains a partial or complete copy of the objects at the target master group. Figure 1–2 illustrates that snapshot group "Group A" at the snapshot site maintains only a partial replica of master group "Group A" at the master site, while the "Group B" snapshot and master groups maintain a complete replica.

Additionally, Figure 1–2 illustrates that each site may contain multiple replication groups.

Figure 1–2 Snapshot Groups Correspond with Master Groups

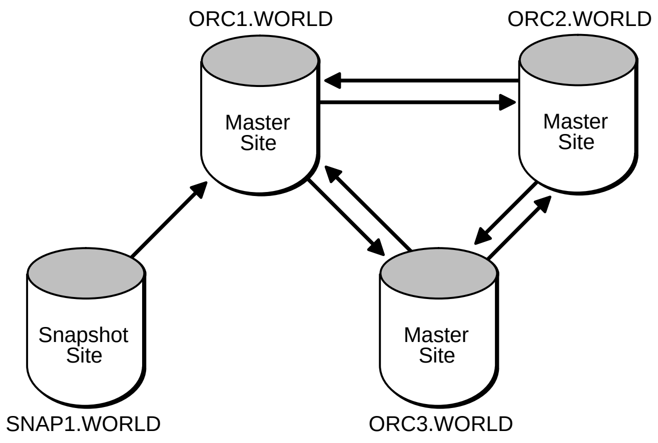


Replication Sites

A replication group can exist at multiple *replication sites*. Replication environments support two basic types of sites: master sites and snapshot sites.

- A *master site* maintains a complete copy of all objects in a replication group. All master sites in a multimaster replication environment communicate directly with one another to propagate data and schema changes in the replication group. A replication group at a master site is more specifically referred to as a *master group*. Additionally, every master group has one and only one *master definition site* (for example, ORC1.WORLD in Figure 1–3 might be the master definition site). A replication group's master definition site is a master site serving as the control point for managing the replication group and objects in the group.
- A *snapshot site* supports read-only and updateable snapshots of the table data at an associated master site. A snapshot site's table snapshots can contain all or a subset of the table data within a replication group. However, these must be simple snapshots with a one-to-one correspondence to tables at the master site. For example, a snapshot site may contain snapshots for only selected tables in a replication group. And a particular snapshot might be just a selected portion of a certain replicated table. A replication group at a snapshot site is more specifically referred to as a *snapshot group*. A snapshot group can also contain other replication objects.

Figure 1–3 Three Master Sites and One Snapshot Site



Multimaster Replication

Oracle’s *multimaster replication* allows multiple sites, acting as equal peers, to manage groups of replicated database objects. Applications can update any replicated table at any site in a multimaster configuration. Figure 1–4 illustrates a multimaster replication system.

Oracle database servers operating as master sites in a multimaster environment automatically work to converge the data of all table replicas, and ensure global transaction consistency and data integrity.

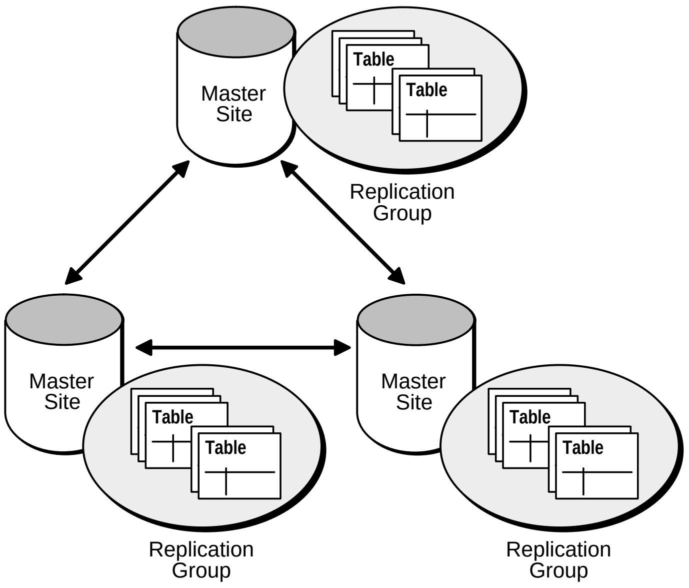
Uses for Multimaster Replication

Multimaster replication is useful for many types of application systems with special requirements. The following scenarios describe some of the uses for multimaster replication:

Failover Site

Multimaster replication can be useful to protect the availability of a mission critical database. For example, a multimaster replication environment can replicate all of the data in your database to establish a failover site should the primary site become unavailable due to system or network outages. In contrast with Oracle's standby database feature, such a failover site can also serve as a fully functional database to support application access when the primary site is concurrently operational.

Figure 1–4 Multimaster Replication System

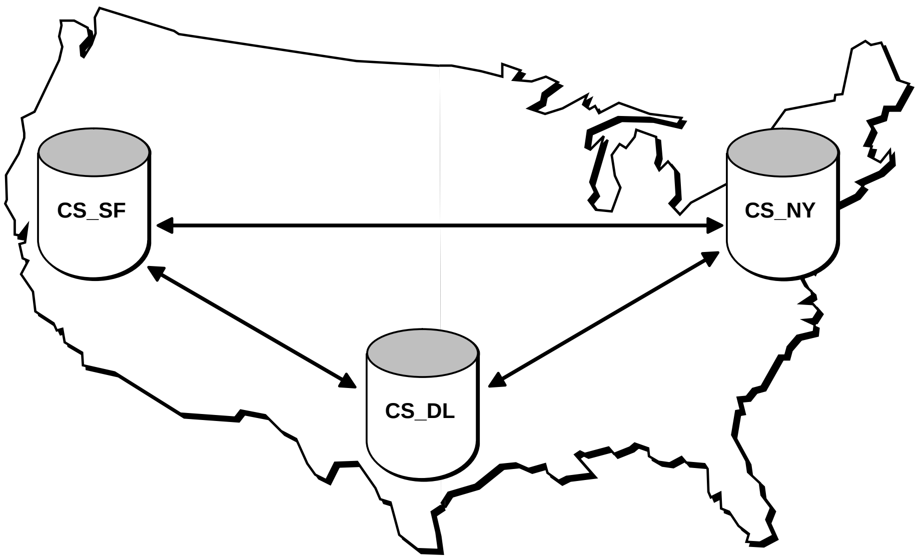


Distributing Application Loads

Multimaster replication is useful for transaction processing applications that require multiple points of access to database information for the purposes of distributing a heavy application load, ensuring continuous availability, or providing more localized data access.

Applications that have application load distribution requirements commonly include customer service oriented applications. (Application load distribution can also be achieved by using updateable snapshots - see "Snapshot Replication" on page 1-7 for more information.)

Figure 1–5 Multimaster Replication Supporting Multiple Points of Update Access



Snapshot Replication

A snapshot contains a complete or partial replica of a target master table from a single point-in-time. A snapshot may be read-only or updateable.

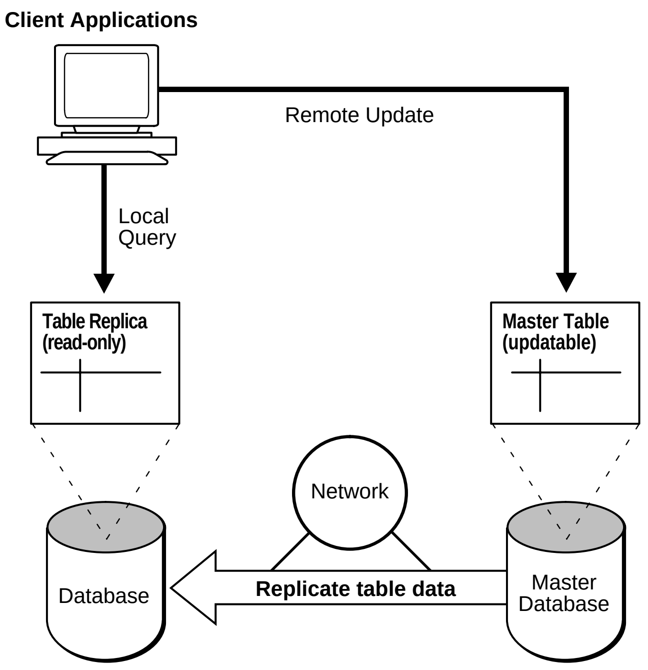
Read-only Snapshots

In a basic configuration, snapshots may provide read-only access to the table data that originates from a primary or "master" site. Applications can query data from local data replicas to avoid network access regardless of network availability. However, applications throughout the system must access data at the primary site when updates are necessary. Figure 1–6 illustrates basic, read-only replication.

The following is a list of benefits of read-only snapshots:

- Master tables do not need to belong to a master group.
- Can support complex snapshots (snapshot may be based on one or more tables and may contain aggregates, joins, set operations, or a CONNECT BY clause).
- Provide local access to provide improved response times and availability.
- Offload queries from master site.

Figure 1–6 Read-Only Snapshot Replication

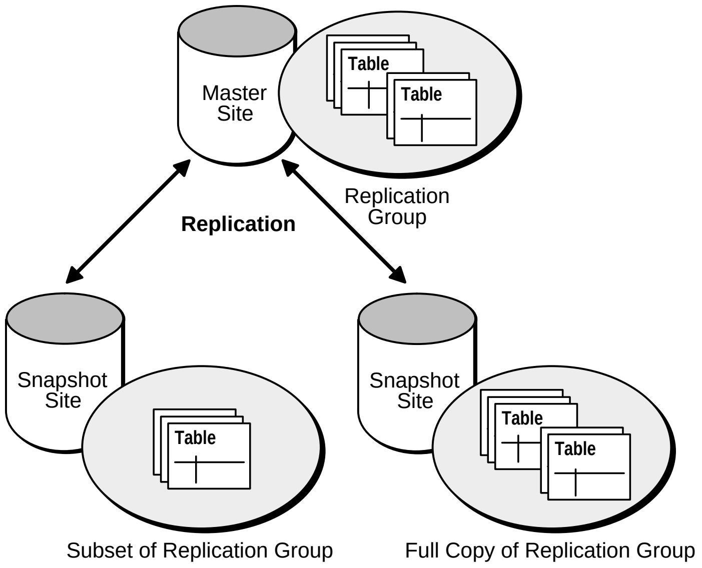


Updateable Snapshots

In a more advanced configuration, you can create an *updateable snapshot* that allows users to insert, update, and delete rows of the target master table. An updateable snapshot may also contain only a subset of the target master table's data set. Figure 1–7 illustrates a replication environment using updateable snapshots.

Updateable snapshots are based on tables at a master site that has been setup to support multimaster replication. In fact, updateable snapshots must be part of a snapshot group that is based on a master group at a master site (see "Snapshot Groups" on page 3-18 for more information).

Figure 1–7 Updateable Snapshot Replication



Updateable snapshots have the following properties.

- Updateable snapshots are always based on a single table and can be incrementally (or "fast") refreshed.
- Oracle propagates the changes made through an updateable snapshot to the snapshot's remote master table. If necessary, the updates then cascade to all other master sites.
- Oracle refreshes an updateable snapshot as part of a refresh group identical to read-only snapshots. (A *refresh group* is an organizational mechanisms that maintains transactional consistency; see "Refresh Groups" on page 3-21 for details).

Updateable snapshots have the following benefits:

- Allowing users to query and update a local replicated data set even when disconnected from the master site.
- Increased data security achieved by replicating only a selected subset of the target master table's data set.
- Smaller footprint than multimaster replication.

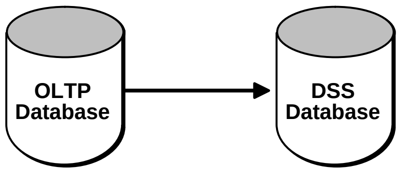
Uses of Snapshot Replication

Snapshot replication is useful for several types of applications. The following sections describe some of the typical uses for snapshot replication:

Information Off-Loading

Read-only snapshot replication is useful as a way to replicate entire databases or off-load information. For example, when the performance of high-volume transaction processing systems is critical, it can be advantageous to maintain a duplicate database to isolate the demanding queries of decision support applications.

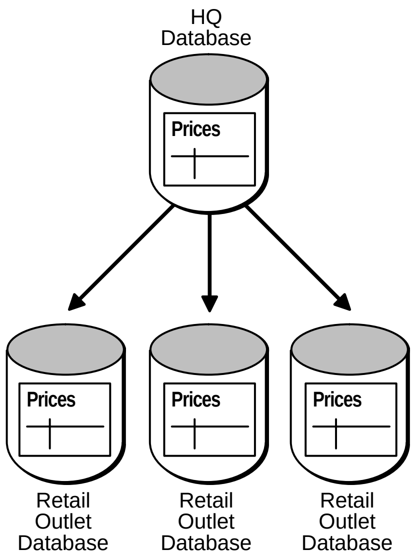
Figure 1–8 Information Off-Loading



Information Distribution

Read-only snapshot replication is useful for information distribution. For example, consider the operations of a large consumer department store chain. In this case, it is critical to ensure that product price information is always available and relatively current and consistent at retail outlets. To achieve these goals, each retail store can have its own copy of product price data that it refreshes nightly from a primary price table.

Figure 1–9 Information Distribution



Information Transport

Read-only and updateable snapshot replication can be useful as an information transport mechanism. For example, read-only snapshot replication can periodically move data from a production transaction processing database to a data warehouse.

Disconnected Environments

Updateable snapshot replication is useful for the deployment of transaction processing applications that operate using disconnected components. For example, consider the typical sales force automation system for a life insurance company. Each salesperson must visit customers regularly with a laptop computer and record orders in a personal database while disconnected from the corporate computer network and centralized database system. Upon returning to the office, each salesperson must forward all orders to a centralized, corporate database.

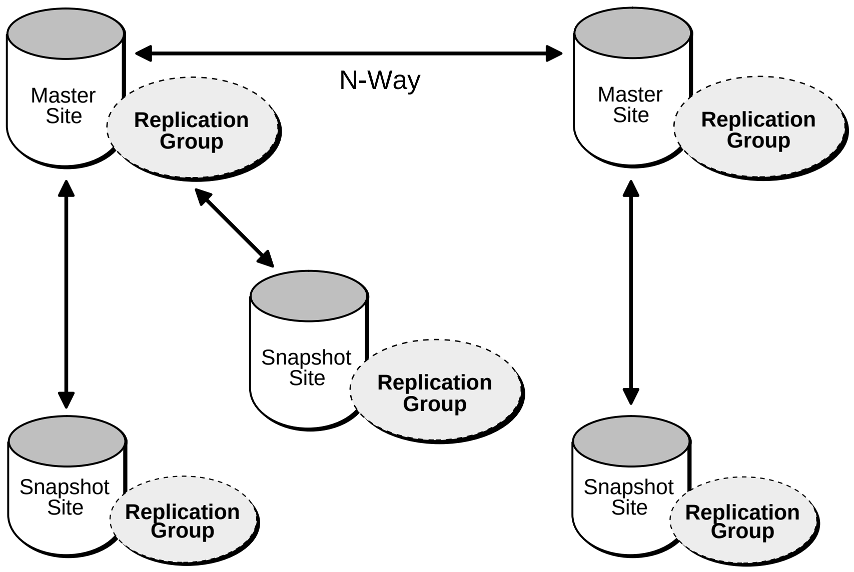
To help deploy a snapshot environment to, for example, a sales force, *deployment templates* allow the database administrator to pre-create a snapshot environment at the master site for an easy, custom, and secure distribution and installation of a snapshot environment. Deployment templates allow the DBA to create a snapshot environment once and deploy as often as necessary to the target snapshot sites.

Multimaster and Snapshot Hybrid Configurations

Multimaster replication and snapshots can be combined in *hybrid* or "mixed" configurations to meet different application requirements. Mixed configurations can have any number of master sites and multiple snapshot sites for each master.

For example, as shown in Figure 1–10, *n*-way (or multimaster) replication between two masters can support full-table replication between the databases that support two geographic regions. Snapshots can be defined on the masters to replicate full tables or table subsets to sites within each region.

Figure 1–10 Hybrid Configuration



Key differences between snapshots and replicated masters include the following:

- Replicated masters must contain data for the full table being replicated, whereas snapshots can replicate subsets of master table data.
- Multimaster replication allows you to replicate changes for each transaction as the changes occur. Snapshot refreshes are set oriented, propagating changes from multiple transactions in a more efficient, batch-oriented operation, but at less frequent intervals.
- If conflicts occur from changes made to multiple copies of the same data, master sites detect and resolve the conflicts.

Administering a Replicated Environment

There are several tools that are available to help you administer and monitor your replication environment. Oracle's Replication Manager provides a powerful GUI interface to help you manage your environment, while the Replication Management API provides you with the familiar application programming interface (API) to build customized scripts for replication administration. Additionally, the replication catalog keeps you informed about your replicated environment.

Replication Catalog

Every master and snapshot site in a replication environment has a *replication catalog*. A site's replication catalog is a distinct set of data dictionary tables and views that maintain administrative information about replication objects and replication groups at the site. Every server participating in a replication environment can automate the replication of objects in replication groups using the information in its replication catalog.

Replication Management API and Administration Requests

To configure and manage a replication environment, each participating server uses Oracle's replication application programming interface (API). A server's *replication management API* is a set of PL/SQL packages encapsulating procedures and functions administrators can use to configure Oracle's replication features. Oracle Replication Manager also uses the procedures and functions of each site's replication management API to perform work.

An *administration request* is a call to a procedure or function in Oracle's replication management API. For example, when you use Replication Manager to create a new master group, Replication Manager completes the task by making a call to the DBMS_REPCAT.CREATE_MASTER_REPGROUP procedure. Some administration requests generate additional replication management API calls to complete the request.

Oracle Replication Manager

Replication environments supporting both a multimaster and snapshot replication environment can be challenging to configure and manage. To help administer these replication environments, Oracle provides a sophisticated management tool, *Oracle Replication Manager*. Other sections in this book include information and examples for using Replication Manager.

Replication Conflicts

Asynchronous multimaster and updateable snapshot replication environments must address the possibility of replication conflicts that may occur when, for example, two transactions originating from different sites update the same row at nearly the same time.

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

In addition to logging any conflicts that may occur in your replicated environment, Oracle replication offers a variety of conflict resolution methods that will allow you to define a conflict resolution system for your database that will resolve conflicts in accordance with your business rules. If you have a unique situation that Oracle’s pre-built conflict resolution methods cannot resolve, you have the option of building and using your own conflict routines.

Chapter 6, "Conflict Resolution" discusses how to design your database to avoid data conflicts and how to build conflict resolution routines that resolve such conflicts when they occur. Chapter 6, "Conflict Resolution" of the *Oracle8i Replication API Reference* describes how to build conflict resolution routines using the Replication Management API.

Specialized Replication Options

Some applications have special requirements of a replication system. The following sections explain the Oracle unique replication options, including:

- Procedural Replication.
- Synchronous (Real-Time) Data Propagation.

Procedural Replication

Batch processing applications can change large amounts of data within a single transaction. In such cases, typical row-level replication could load a network with a large quantity of data changes. To avoid such problems, a batch processing application operating in a replication environment can use Oracle's *procedural replication* to replicate simple stored procedure calls to converge data replicas. Procedural replication replicates only the call to a stored procedure that an application uses to update a table. Procedural replication does not replicate data modifications.

To use procedural replication, you must replicate the packages that modify data in the system to all sites. After replicating a package, you must generate a *wrapper* for this package at each site. When an application calls a packaged procedure at the local site to modify data, the wrapper ensures that the call is ultimately made to the same packaged procedure at all other sites in the replicated environment. Procedural replication can occur asynchronously or synchronously.

Conflict Detection and Procedural Replication When a replication system replicates data using procedural replication, the procedures that replicate data are responsible for ensuring the integrity of the replicated data. That is, you must design such procedures either to avoid or to detect replication conflicts and resolve them appropriately. Consequently, procedural replication is most typically used when databases are available only for the processing of large batch operations. In such situations, replication conflicts are unlikely because numerous transactions are not contending for the same data. See "Using Procedural Replication" on page 8-2.

Synchronous (Real-Time) Data Propagation

Asynchronous data propagation is the normal configuration for replication environments. However, Oracle also supports synchronous data propagation for applications with special requirements. *Synchronous data propagation* occurs when an application updates a local replica of a table, and within the same transaction, also updates all other replicas of the same table. Consequently, synchronous data replication is also called *real-time data replication*. Use synchronous replication only when applications require that replicated sites remain continuously synchronized.

You can create a replicated environment with some sites propagating changes synchronously while others use asynchronous propagation (deferred transactions).

Note: A replication system using real-time propagation of replication data is highly dependent on system and network availability because it can function only when all system sites are concurrently available.

Replication Conflicts and Synchronous Data Replication When a shared ownership system replicates all changes synchronously (real-time replication), replication conflicts cannot occur. With real-time replication, applications use distributed transactions to update all replicas of a table at the same time. As is the case in nondistributed database environments, Oracle automatically locks rows on behalf of each distributed transaction to prevent all types of destructive interference among transactions. Real-time replication systems can prevent replication conflicts.

2 Using Multimaster Replication

This chapter explains how to configure and manage an advanced replication system that uses multimaster replication. Advanced replication is only available with the Oracle8*i* Server Enterprise Edition. To learn more about the differences between Oracle* products and the Oracle8*i* Server Enterprise Edition, please refer to the book *Getting to Know Oracle8i*.

This chapter covers the following topics.

- Oracle's Multimaster Replication Architecture.
- Quick Start: Building a Multimaster Replication Environment.
- Preparing for Multimaster Replication.
- Managing Scheduled Links.
- Purging a Site's Deferred Transaction Queue.
- Managing Master Groups.
- Advanced Multimaster Replication Options.

Note: This chapter explains how to manage a multimaster replication system that uses the default replication architecture—row-level replication using asynchronous propagation. For information about configuring procedural replication and synchronous data propagation, see Chapter 8, "Advanced Techniques". Also, examples appear throughout this chapter of how to use the Oracle Replication Manager tool to manage a multimaster replication system. Each section lists equivalent replication management API procedures for your reference. For complete information about Oracle's replication management API, see *Oracle8i Replication API Reference*.

Oracle's Multimaster Replication Architecture

Oracle converges data from typical advanced replication configurations using row-level replication with asynchronous data propagation. The following sections explain how these mechanisms function.

Note: Oracle offers other advanced replication features such as procedural replication and synchronous data propagation for unique application requirements. To learn more about these special configurations, read "Specialized Replication Options" on page 1-14.

Row-Level Replication

Typical transaction processing applications modify small numbers of rows per transaction. Such applications at work in an advanced replication environment will usually depend on Oracle's row-level replication mechanism. With *row-level replication*, applications use standard DML statements to modify the data of local data replicas. When transactions change local data, the server automatically captures information about the modifications and queues corresponding deferred transactions to forward local changes to remote sites.

Generated Replication Objects

To support the replication of transactions in an advanced replication environment, one or more internal system objects are generated to support each replicated table, package or procedure.

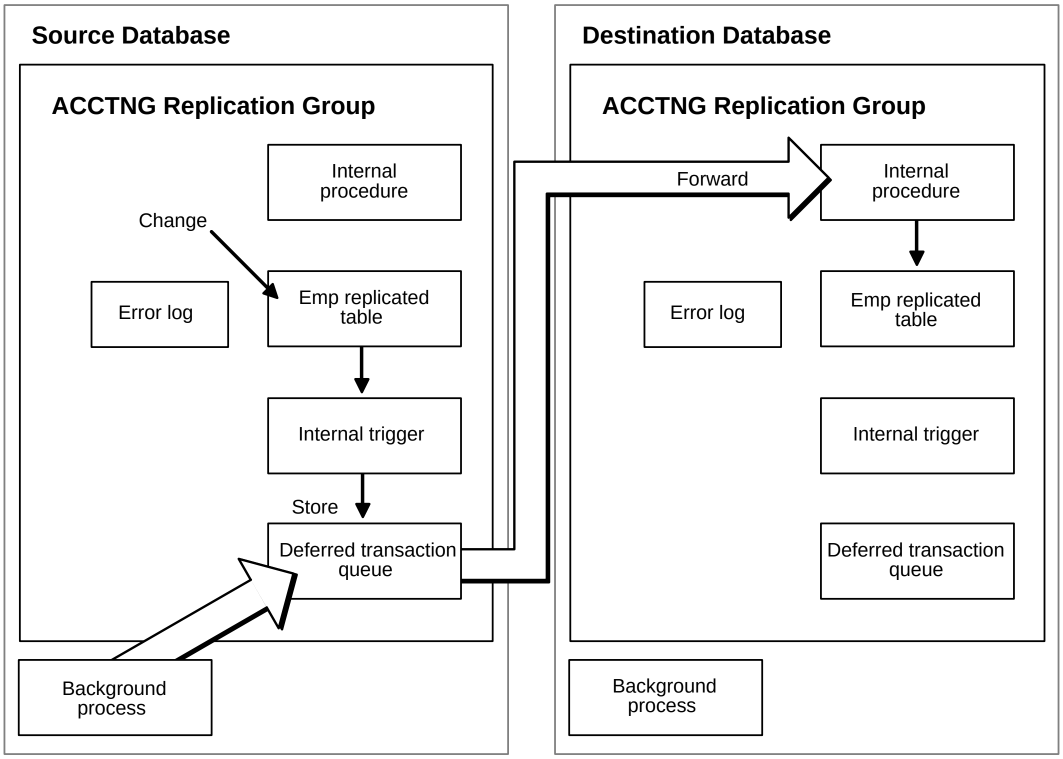
- When you replicate a package specification and package body to support procedural replication, you can generate a corresponding *wrapper* package specification and package body. By default, Oracle names the wrapper for a package specification and package body using the name of the object with the prefix "defer_".

Note: With versions of Oracle prior to 8.0, the server also generated PL/SQL triggers for a replicated table. With version 8.1, the server "activates" internal triggers and packages when you generate replication support for a replicated table.

Asynchronous (Store-and-Forward) Data Propagation

Typical advanced replication configurations that rely on row-level replication propagate data level changes using *asynchronous data replication*. Asynchronous data replication occurs when an application updates a local replica of a table, stores replication information in a local queue, and then forwards the replication information to other replication sites at a later time. Consequently, asynchronous data replication is also called *store-and-forward data replication*.

Figure 2–1 Asynchronous Data Replication Mechanisms



As Figure 2–1 shows, Oracle uses its internal system of triggers, deferred transactions, deferred transaction queues, and job queues to propagate data-level changes asynchronously among master sites in an advanced replication system, as well as from an updateable snapshot to its master table.

- When applications work in an advanced replication environment, Oracle uses internal triggers to capture and store information about updates to replicated data. Internal triggers build *remote procedure calls (RPCs)* to reproduce data changes made at the local site to remote replication sites. The internal triggers supporting data replication are essentially components within the Oracle Server executable. Therefore, Oracle can capture and store updates to replicated data very quickly with minimal use of system resources.
- Oracle stores RPCs produced by the internal triggers in a site’s *deferred transaction queue* for later propagation. Oracle also records information about initiating transactions so that all RPCs from a transaction can also be propagated and applied remotely as a transaction. Oracle’s advanced replication facility implements the deferred transaction queue using Oracle’s advanced queueing mechanism.
- Oracle manages the propagation process using Oracle's *job queue mechanism* and *deferred transactions*. Each server participating in an advanced replication system has a local job queue. A server’s job queue is a database table storing information about local jobs such as the PL/SQL call to execute for a job, when to run a job, and so on. Typical jobs in an advanced replication environment include jobs to push deferred transactions to remote master sites, jobs to purge applied transactions from the deferred transaction queue, and jobs to refresh snapshot refresh groups.
- Oracle forwards data replication information by executing RPCs as part of deferred transactions. Oracle uses distributed transaction protocols to protect global database integrity automatically and ensure data survivability.

Serial Propagation

With *serial propagation*, Oracle asynchronously propagates replicated transactions, one at a time, in the same order of commit as on the originating site.

Parallel Propagation

With *parallel propagation*, Oracle asynchronously propagates replicated transactions using multiple, parallel transit streams for higher throughput. When necessary, Oracle orders the execution of dependent transactions to ensure global database integrity.

Parallel propagation uses the same execution mechanism Oracle uses for parallel query, load, recovery, and other parallel operations. Each server process propagates transactions through a single stream. A parallel coordinator process controls these server processes. The coordinator tracks transaction dependencies, allocates work to the server processes, and tracks their progress.

Purging of the Deferred Transaction Queue

After a site pushes a deferred transaction to its destination, the transaction remains in the deferred transaction queue until another job purges the applied transaction from the queue.

Replication Administrators, Propagators, and Receivers

An Oracle advanced replication environment requires several unique database user accounts to function properly, including replication administrators, propagators, and receivers.

- Every site in an Oracle advanced replication system requires at least one *replication administrator,* a user responsible for configuring and maintaining replicated database objects.
- Each replication site in an Oracle advanced replication system requires special user accounts to propagate and apply changes to replicated data.

Configuration Options

In most advanced replication configurations, just one account is used for all purposes: as a replication administrator, a replication propagator, and a replication receiver. However, Oracle also supports distinct accounts for unique configurations.

Quick Start: Building a Multimaster Replication Environment

To create a multimaster advanced replication environment, you must complete the following steps at a minimum:

1. Design the advanced replication environment. Decide what tables and supporting objects to replicate to multiple databases, and organize replication objects in suitable master groups.
2. Use Replication Manager's setup wizard to configure a number of databases to support a multimaster replication environment. The Replication Manager setup wizard quickly configures all components necessary to support a multimaster replication system.
3. Using the Replication Manager database connection to the master definition site, create one or more master groups to replicate tables and related objects to multiple master sites.
4. Grant privileges necessary for application users to access data at each site.

For detailed information about each step and other optional configuration steps, see the later sections of this chapter.

A Simple Example

The following simple example demonstrates the steps necessary to build a multimaster replication environment.

Step 1: Design the Environment

The first step is to design the basic replication environment. This example demonstrates how to replicate the tables SCOTT.EMP and SCOTT.DEPT tables at the master sites DBS1 and DBS2. DBS1 is designated as the master definition site for the system.

Note: The primary key of SCOTT.EMP is the EMPNO column, and the primary key of SCOTT.DEPT is the DEPTNO column.

Step 2: Use the Replication Manager Setup Wizard

The Replication Manager setup wizard helps you configure the supporting accounts, links, schemas, and scheduling at all master sites in a multimaster replication system. For this example, use the setup wizard to:

- Identify the master sites DBS1 and DBS2.
- Create the default REPADMIN account at each master site to serve as global replication administrator, propagator, and receiver account.
- Create the schema SCOTT at each master site to support the proposed replication objects throughout the multimaster system.
- Configure the default scheduling of data propagation and purging at each master site (it is not necessary to customize each master site's scheduling information for the purposes of this brief example).

Step 3: Create a Master Group

In a multimaster replication environment, Oracle replicates tables and related replication objects as part of a master group. Using the database connection to the master definition site DBS1, open Replication Manager's Create Master Group property sheet to create a new master group called EMPLOYEE. Use the property sheet's pages to identify the replication objects for the group, SCOTT.EMP and SCOTT.DEPT, as well as the other master site, DBS2. By default, Replication Manager generates replication support for all objects in the group and then resumes replication activity for the group.

Step 4: Grant Access to Replication Objects

After configuring a multimaster replication environment, grant access to the various replication objects so that users that connect to each site can use them.

```
GRANT SELECT ON scott.emp TO ... ;
```

Other Steps

This simple example does not mention several optional steps that might be necessary to configure certain multimaster replication systems. For example, when an advanced replication system uses a shared ownership data model, you'll want to configure conflict resolution for all replicated tables before resuming replication activity for a master group. Refer to the remainder of this chapter for more detailed information about configuring multimaster replication systems.

Preparing for Multimaster Replication

Before starting to build a multimaster advanced replication environment, you must prepare each participating database with the following:

- A replication administrator.
- A replication propagator.
- A replication receiver.
- Database links to provide for interdatabase communication.

The Replication Setup Wizard

Preparing all sites for a default multimaster replication configuration is a simple process using Replication Manager's replication setup wizard. At each master site that you specify, this wizard performs the following steps:

- The wizard creates a database account to serve as a replication administrator. By default, the wizard creates this account to serve also as the replication propagator and receiver.
- The wizard grants the necessary privileges to the replication administrator / propagator / receiver account.
- The wizard creates Replication Manager database connections to correspond to new replication administrator accounts.
- The wizard creates scheduled links to all other master sites in the environment.
- The wizard schedules purging of the deferred transaction queue for all master sites in the system.

To start the Replication Manager setup wizard:

1. Select **Setup Wizard** from the **File** menu.

The following sections explain how to use the Replication Manager setup wizard to prepare the master sites in a multimaster replication system.

Create Master Sites

The initial page of the replication setup wizard prompts you to indicate what type of replication environment setup that you want to perform.

1. Select **Setup Master Sites**.
2. Click **Next**.

The next page of the wizard lets you create a list of the master sites in the new multimaster replication system. At this point, it is likely that you will not have any Replication Manager database connections available to use for the setup wizard. When this is the case, perform the following steps

1. Press the **Add** button.
2. In the **Add Site** dialog that appears, enter the global database name of a master site in the proposed system, as well as the password for the SYSTEM account at the site. (The setup wizard uses the SYSTEM account to perform subsequent configuration tasks.)
3. When you finish, press the **Down Arrow** button to add the site to the list of master sites in the setup wizard.
4. Repeat Step 2 for each master site. After you enter the final site's information, press the **OK** button to add the site and dismiss the dialog.
5. After reviewing the list of master sites, click **Next** to continue.

Create Replication Administrator, Propagator, and Receiver Accounts

The next page of the wizard lets you specify information for the database accounts that will function as each master site's replication administrator, propagator, and receiver. The wizard creates accounts with the same name and password at all master sites in the system.

The setup wizard supports two different types of master site account setups.

- By default, the wizard creates a single account that is capable of performing administrator, propagator, and receiver functions. Simply enter an account name and password and then press **Next** to continue.

- Alternatively, when you want to create a separate account for replication administration apart from the propagator/receiver account, enable the **Different Propagator/Receiver** checkbox and enter different account information for each user account. Then press **Next** to continue.

Create Schemas to Organize Replication Objects

The next page of the setup wizard lets you indicate what schemas to create as schemas that will contain replication objects. The wizard creates schemas with the same name and password at all master sites in the system.

To add new schemas to the list

1. Press the **Add** button.
2. In the **Create Schema** dialog that appears, enter the name of a schema that you want to use to contain replication objects, as well as the password for the schema. When you finish, click **OK** to add the schema to the list of schemas in the setup wizard.
3. Repeat Steps 1 and 2 to add additional schema.
4. When you have finished adding schema, press the **Next** button to continue.

Create Scheduled Links

The next page of the setup wizard lets you indicate default propagation characteristics for all master sites in the system. The setup wizard uses this information to create corresponding scheduled links from each master site to all other master sites. For explanations of each setting in this page of the wizard, see "Creating a Scheduled Link".

After reviewing the default scheduling settings or making any necessary changes, press the **Next** button to continue.

Specify Purge Scheduling

The next page of the setup wizard lets you configure the default purge schedule for the deferred transaction queue at each master site in the system. For explanations of each setting in this page of the wizard, see "Purging a Site’s Deferred Transaction Queue".

After reviewing the default purge settings or making any necessary changes, press the **Next** button to continue.

Customize Each Master Site

The next page of the setup wizard lets you customize settings for individual master sites in the system. If you choose not to customize master sites in the system, each site will have matching:

- Replication administrators, propagators, and receivers.
- Database link specifications.
- Scheduled propagation and purge settings.

To customize a master site's settings:

1. Select the target master site to customize.
2. Press the **Customize** button.

Use the pages of the **Customize Master Site** property sheet to customize the target master site's:

- Replication administrator and / or propagator accounts.
- Purge scheduling settings.

After reviewing the customized settings for a master site, press the **OK** button. To customize another master site's settings, repeat the process above. When you are finished customizing all master sites, press the **Next** button to continue.

Reviewing and Building the Configuration

The next page of the setup wizard asks if you are ready to complete the configuration of the multimaster advanced replication system. When you are ready, click **Finish** to continue. Replication Manager then presents an informational dialog that lets you quickly review your settings.

- If everything looks good, click **OK** to build the multimaster replication environment.
- If you find a setting that is not correct, click **Cancel** to return to the **Finish** page. Then click **Back** and **Next** to navigate among the pages of the wizard and change any setting. Return to the final page of the wizard when you are ready to build the environment.

After you click **Finish**, Replication Manager builds the multimaster replication environment.

Note: If you want to record a script of the API procedures that are executed during the setup process, click **Record a script** before building the system. Additionally, Replication Manager records information in the file Repsetup.log in the current working directory.

What's Next?

After using the Replication Manager setup wizard, you should continue configuration by completing the following steps.

1. Create the master groups at the site that serves as each group's master definition site. See "Managing Master Groups" for more information about creating and managing master groups.
2. Create or identify the replication objects for each master group at its master definition site. See "Adding Objects to a Master Group" for more information about creating and managing replicated objects.
3. Configure conflict resolution for all replicated tables in each master group. See Chapter 6, "Conflict Resolution" for more information about configuring conflict resolution for replication objects.
4. Resume replication activity for each master group. See "Resuming Replication Activity for a Master Group" for more information about managing the replication activity for master groups.
5. Monitor the multimaster environment to ensure that the system is operating properly. See "Monitoring an Advanced Replication System" for more information about monitoring the activity of master groups.

Starting SNP Background Processes

To simplify administration, most advanced replication environments configure data propagation to occur automatically. Accordingly, each master site in an advanced replication environment must start one or more SNP background processes. The following initialization parameters control the SNP background process setting for each server.

- JOB_QUEUE_PROCESSES specifies the number of SNP*n* background processes per server, where *n* is 0 to 9 followed by A to Z. Set this parameter to a value of one or higher. Two or three background processes will usually be sufficient unless you have a large system.

- JOB_QUEUE_INTERVAL specifies the interval, in seconds, between wake-ups for the SNP background processes of a server. The default of 60 seconds is adequate for typical replication environments.

Managing Scheduled Links

Scheduled links are necessary to propagate replicated data from one replication site to another. In a multimaster replication environment, each master site requires a scheduled link to move data to every other master site. Additionally, a snapshot site with updateable snapshots requires a scheduled link to move data to its corresponding master site.

Among other things, Replication Manager's setup wizards prepare each multimaster or snapshot site environment with the necessary scheduled links. Replication Manager also has features that allow you to manage scheduled links. The following sections explain more about managing scheduled links.

Creating a Scheduled Link

To create a new scheduled link:

1. Select the database in the left pane of the Replication Manager user interface where you want to create the scheduled link.
2. Select **Create** from the **File** menu
3. Select **Scheduled Link** from the sub-menu.

   You can optionally press the **Create Scheduled Link** button on the toolbar.

Use the **Create New Scheduled Link** property sheet to create the new link. The following sections explain the settings that are available for the **General** and **Options** pages of this property sheet. Press the **Create** button when you have completed creating your scheduled link.

Link The database link to use for the new scheduled link. Click **Browse** to display the **Set Scheduled Link** dialog and select a database link. The database link must already exist.

Next Date The initial time to push changes to the new destination. Click **Edit** to display the **Set Date** dialog and set a time for the **Next Date** field.

Interval The automatic interval to push changes to the new destination. Click **Edit** to display the **Set Interval** dialog and set a time for the **Interval** field.

Delay Seconds The amount of time to continue polling the queue, even if the queue is empty. See "Guidelines for Scheduled Links" for more information about this setting.

Enabled Check to immediately enable the new scheduled link and push changes to the new destination.

Note: If the target destination is unavailable when creating the link, consider disabling the new scheduled link. This way, Oracle does not try to push changes to the unavailable destination.

Stop on Error How to react after an error occurs while pushing the local deferred transaction queue. The default, unchecked, indicates that propagation of the local deferred transaction queue should continue. When checked, Oracle stops execution of deferred transactions.

Parallel Propagation Whether to use parallel propagation (or serial propagation) for the scheduled link. When checked, you can set parallel propagation settings for the link. When unchecked, the new scheduled link uses serial propagation. See "Planning for Parallel Propagation" for more information.

Processes The number of background processes that the scheduled link uses for parallel propagation of information to the destination. The default value, 0, is an alternate way to indicate serial propagation for the link. A value n that is greater than 0 indicates parallel propagation with *n* background processes.

Batch Size (Oracle7 Database Only) It determines how often to commit transactions when pushing the local deferred transaction queue. The default, 0, indicates that you want to commit each transaction as it pushes to the remote destination. When using serial propagation for the scheduled link, setting Batch Size to a higher value can commit several deferred transactions in one operation and reduce the overhead from many transaction commits.

API Equivalents: DBMS_DEFER_SYS.SCHEDULE_PUSH, DBMS_DEFER_SYS.SCHEDULE_EXECUTION (Oracle7 Database Only)

Guidelines for Scheduled Links

A scheduled link determines how a master site propagates its deferred transaction queue to another master site (or from a snapshot site to its master site). When you create a scheduled link, Oracle creates a job in the local job queue to push the deferred transaction queue to another site in the system. When Oracle propagates deferred transactions to a remote master site, it does so within the security context of the replication propagator. Additionally, you can configure a scheduled link to push information using serial or parallel propagation. Before creating the scheduled links for an advanced replication system, carefully consider how you want replication to occur globally throughout the system.

For example, to simulate near real-time replication, you might want to have each scheduled link constantly push a master site's deferred transaction queue to its destination. Alternatively, when you choose to propagate deferred transactions at regular intervals, you must decide how often and when to schedule pushes. You might want schedule links at a time of the day when connectivity is guaranteed or when communications costs are lowest, such as during evening hours. Furthermore, you might want to stagger the scheduling for links among all master sites to distribute the load that replication places on network resources.

Scheduling Continuous Pushes Even when using Oracle's asynchronous replication mechanisms, you can configure a scheduled link to simulate continuous, real-time replication. When configuring a scheduled link in the Replication Manager setup wizard, the **Create Scheduled Link** property sheet, or the **Edit Scheduled Link** property sheet, set **Delay Seconds** on the **Option** page to 500,000.

Scheduling Periodic Pushes Alternatively, you can schedule periodic pushes of a site's deferred transaction queue to a remote destination. When configuring a scheduled link in the Replication Manager setup wizard, the **Create Scheduled Link** property sheet, or the **Edit Scheduled Link** property sheet, set **Delay Seconds** on the **Option** page to the default value, 0. Then configure the interval to push the deferred transaction queue using the **Next Date** and **Interval** settings on the **General** page.

Deciding between Serial and Parallel Propagation A scheduled link can push a site's deferred transaction queue using either serial or parallel propagation. For more information about issues related to serial and parallel propagation, see "Planning for Parallel Propagation" on page 2-40.

Editing a Scheduled Link

To edit the refresh interval or propagation characteristics for a scheduled link, or disable a scheduled link

1. Expand the database node that contains the scheduled link that you want to edit.
2. Expand the **Scheduling** node.
3. Expand the **Scheduled Links** node.
4. Select the scheduled link that you want to edit.

   The current properties of the selected scheduled link will appear in the right pane of the Replication Manager user interface.

Use the **Scheduled Link** property sheet to modify the properties of the scheduled link and apply your changes. Press the **Apply** button when you have completed your modifications. See "Creating a Scheduled Link" on page 2-12 for more information about the properties that you can adjust for a scheduled link.

API Equivalents: DBMS_DEFER_SYS.SCHEDULE_PUSH, DBMS_DEFER_SYS. SET_DISABLED, DBMS_DEFER_SYS.SCHEDULE_EXECUTION (Oracle7 only)

Viewing the Status of a Scheduled Link

To list the status of all scheduled links for a site, use Replication Manager.

1. Expand the database node that contains the scheduled link(s) that you want to view.
2. Expand the **Scheduling** node.
3. Select the **Scheduled Links** node.

   All of the scheduled links in the selected database will be displayed in the right pane of the Replication Manager user interface. Link name, next date, interval, and last date properties will be displayed for each scheduled link.

API Equivalent: DBMS_DEFER_SYS.DISABLED

Deleting a Scheduled Link

To delete a scheduled link

1. Expand the database node that contains the scheduled link that you want to delete.
2. Expand the **Scheduling** node.
3. Expand the **Scheduled Links** node.
4. Right-click on the scheduled link that you want to delete and select **Remove**.
5. Press the **Yes** button to confirm that you want to delete the selected scheduled link. Press the **No** button to leave the scheduled link intact.

API Equivalent: DBMS_DEFER_SYS.UNSCHEDULE_PUSH

Purging a Site’s Deferred Transaction Queue

After successfully pushing a deferred transaction to its destination master site, the transaction does not have to remain in the site’s deferred transaction queue. Regular purging of applied deferred transactions from a site’s deferred transaction queue keeps the size of the queue manageable. When you use the Replication Manager setup wizard to configure an advanced replication system, the wizard configures purging for all master and snapshot sites in the system. The settings for a site’s purge schedule include:

Next Date The next time to purge applied transactions from the local deferred transaction queue. Click **Edit** to display the **Set Date** dialog and set a time for the **Next Date** field.

Interval Expression The automatic interval to purge applied transactions from the local deferred transaction queue. Click **Edit** to display the **Set Interval** dialog and set a time for the **Interval** field.

Rollback Segment The rollback segment to target when performing a purge of the local deferred transaction queue. Click **Browse** to display the **Select a Rollback Segment** dialog and pick a rollback segment in the database. A null value for this setting allows Oracle to pick the rollback segment when purging the deferred transaction queue.

Note: When you expect a purge of the local deferred transaction queue to generate a large amount of rollback data, target a sufficiently large rollback segment.

Delay Seconds The amount of time to continue polling the queue, even if the queue is empty. Useful for reducing overhead when scheduled purges happen frequently.

API Equivalents: DBMS_DEFER_SYS.SCHEDULE_PURGE, DBMS_DEFER_ SYS.SCHEDULE_EXECUTION (Oracle7 only)

Guidelines for Scheduled Purges of a Deferred Transaction Queue

A scheduled purge determines how a master or snapshot site purges applied transactions from its deferred transaction queue. When you use Replication Manager’s setup wizards to create a master or snapshot site, Oracle creates a job in each site’s local job queue to purge the local deferred transaction queue on a regular basis. Carefully consider how you want purging to occur before configuring the sites in an advanced replication system. For example:

- You can synchronize the pushing (scheduled links) and purging of a site’s deferred transaction queue. For example, you can configure continuous pushing and purging of the transaction queue. This type of configuration might offer performance advantages because it’s likely that information about recently pushed transactions is already in the server’s buffer cache for the corresponding purge operation.
- When a server is not CPU bound, you can schedule continuous purging of the deferred transaction queue to keep the size of the queue as small as possible.
- For servers that experience a high-volume of transaction throughput during normal business hours, you can schedule purges to occur during off-peak hours.

Scheduling Continuous Purges To configure continuous purging of a site’s deferred transaction queue when using a Replication Manager setup wizard, or the **Purge Scheduling** page of the **Edit DB Connection** property sheet, set **Delay Seconds** to 500,000.

Scheduling Periodic Purges Alternatively, you can schedule periodic purges of a site’s deferred transaction queue. When configuring a site’s scheduled purge using a Replication Manager setup wizard, or the **Purge Scheduling** page of the **Edit DB Connection** property sheet, set **Delay Seconds** to the default value, 0. Then configure the interval to purge the deferred transaction queue using the **Next Date** and **Interval** settings.

Specifying a Site’s Purge Schedule

If you manually configured a master or snapshot site or want to modify a site’s purge schedule, use the **Database Information** property sheet. To edit the purge schedule for a site:

1. Expand the database node that contains the purge schedule that you want to edit.
2. Select the **Database Information** node.

   The current properties of the selected scheduled link will appear in the right pane of the Replication Manager user interface.
3. Press the **Purge Job** page.

Use the **Purge Job** page of the **Database Information** property sheet to modify the purge schedule for the site and apply your changes. Press the **Apply** button when you have completed your modifications.

API Equivalents: DBMS_DEFER_SYS.SCHEDULE_PURGE, DBMS_DEFER_ SYS.SCHEDULE_EXECUTION (Oracle7 only)

Manually Purging a Site’s Deferred Transaction Queue

To manually purge a master or snapshot site’s deferred transaction queue, use the **Database Information** property sheet. To edit the purge schedule for a site:

1. Expand the database node that contains the deferred transactions that you want to purge.
2. Select the **Database Information** node.

   The current properties of the selected scheduled link will appear in the right pane of the Replication Manager user interface.
3. Press the **Purge Job** page.
4. Press the **Purge** button.

API Equivalents: DBMS_DEFER_SYS.PURGE

Managing Master Groups

Each master site in an advanced replication system maintains a complete copy of all objects in a replication group. A replication group at a master site is more specifically referred to as a master group. Replication Manager has many features that let you create and manage master groups.

The following sections explain more about managing master groups.

Creating a Master Group

To create a new master group in an advanced replication environment, use the **Create Master Group** property sheet of Replication Manager. To create a new master group

1. Select the database in the left pane of the Replication Manager user interface that you want to be the master definition site for the new master group.
2. Select **Create** from the **File** menu
3. Select **Master Group** from the sub-menu.

   You can optionally press the **Create Master Group** button on the toolbar.

The **Create Master Group** property sheet has three pages: **General**, **Objects**, and **Destinations**. The settings of the **Objects** and **Destinations** pages are optional; if used, they enable Replication Manager to complete more configuration steps when creating a master group.

- To create a master group without any replication objects for the group and without registering other master sites that replicate the master group, complete the settings of the **General** page only and ignore the optional **Objects** and **Destinations** pages.
- Use the **Objects** page only when you want to add objects to the new master group during creation. You should consider several issues when adding objects to a master group. See "Adding Objects to a Master Group" for more information about adding replication objects to a master group.
- Use the **Destinations** page only when you want to add destination master sites to the new master group during creation. You should consider several issues when adding master sites to a master group. See "Adding a Master Site to a Master Group" for more information about adding master sites to a replication group.

Note: During the creation of a new master group, Replication Manager might prompt for supplemental information to create the group and the replication objects that you identify. For example, when you create a new master group along with a replicated table that does not have a primary key, Replication Manager displays the **Set Alternate Key Columns** dialog so that you can identify an alternate identity column or set of columns for the replicated table. Replication Manager also prompts whether or not to enable replication activity for the group after creation.

Warning: If you decide to add one or more tables to a master group during creation of the group, make sure not to resume replication activity. First consider the possibility of replication conflicts, and configure conflict resolution for the replicated tables in the group. See Chapter 6, "Conflict Resolution" for more information about configuring conflict resolution for master group objects.

API Equivalents: DBMS_REPCAT.CREATE_MASTER_REPGROUP, DBMS_
REPCAT.SET_COLUMNS

Using Link Qualifiers for a Master Group

Connection qualifiers allow several database links pointing to the same remote database to establish connections using different paths. For example, a database can have two public database links DBS1 that connect to the remote database using different paths.

- DBS1@ETHERNET, a link that connects to DBS1 using an ethernet link
- DBS1@MODEM, another link that connects to DBS1 using a modem link

Additional Information: See Chapter 2 of *Oracle8i Distributed Database Systems* to learn about defining connection qualifiers for a database link.

When you create a new master group, you can indicate that you want to use a connection qualifier for all scheduled links that correspond to the group. However, when you use connection qualifiers for a master group, Oracle propagates information only after you have created database links with connection qualifiers at every master site.

For example, consider a multimaster configuration with two master sites, DBS1 and DBS2, and two master groups, MG1 and MG2. You want the group MG1 to use the connection qualifier ETHERNET and the group MG2 to use the connection qualifier MODEM. To accomplish this configuration:

- The DBS1 database must have a database link DBS2@ETHERNET and DBS2@MODEM.
- The DBS2 database must have a database link DBS1@ETHERNET and DBS1@MODEM.
- When you create the group MG1, indicate that you want to use the connection qualifier ETHERNET.
- When you create the group MG2, indicate that you want to use the connection qualifier MODEM.

Caution: To preserve transaction integrity in a multimaster environment that uses connection qualified links and multiple master groups, a transaction cannot manipulate replication objects in multiple groups.

Attention: If you plan to use connection qualifiers, you will probably need to increase the value of the initialization parameter OPEN_LINKS at all master sites. The default is four open links per process. You will need to estimate the required value based on your usage. See the *Oracle8i Reference* for more information about the parameter OPEN_LINKS.

Deleting a Master Group

To remove a master group from all master sites in an advanced replication environment:

1. Expand the database node that contains the master group that you want to delete.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Right-click on the target master group and select **Remove**.

API Equivalent: DBMS_REPCAT.DROP_MASTER_REPGROUP

Suspending Replication Activity for a Master Group

Before completing most administrative operations for a master group or any of its replication objects, Oracle requires that you suspend replication activity for the master group at all master sites. Suspending replication activity is also called *quiescing* the master group.

Oracle requires that you suspend replication activity before completing the following administration tasks:

- Adding an object to a master group.
- Altering the definition of an object in a master group.
- Regenerating replication support for a replication object.
- Adding or modifying conflict resolution for a replicated table.
- Adding a master site destination to a master group.
- Changing the propagation method from one site to another.

You may find it necessary to suspend replication activity for a group in other situations as well. For example, administrators may wish to suspend activity and perform queries and updates manually on master group table replicas to restore equivalence if an unexpected conflict is detected that was not resolved.

Warning: Before performing any administration task that requires you to suspend replication activity of a group, wait until the status of the group is "quiescing" at the master definition site. If the presence of a nonempty deferred transaction queue or replication trigger at a site could cause a problem, you should wait until the status of the group is "quiesced" before proceeding.

To suspend replication activity for a master group:

1. Expand the database node that contains the master group that you want to quiesce.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Select the target master group that you want to quiesce.

   The property page for the selected master group will appear in the right pane of the Replication Manager user interface.
5. Press the **Suspend** button.

After suspending replication activity for a master group, monitor the status of the master group at all master sites before completing any administrative operation at the master definition site.

Note: When you request Oracle to suspend replication activity for a master group, Oracle first pushes the deferred transaction queue at all master sites before "quiescing" the group. During the process, Replication Manager displays the status of the group "Await Callback." Once the process completes at all sites, Replication Manager displays the status of the group "Quiesced".

API Equivalent: DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY

Resuming Replication Activity for a Master Group

After completing administrative operations for a master group or any of its replication objects, you can resume replication activity for the master group at all master sites.

Note: Before resuming replication activity for a master group, ensure that there are no unexpected errors by checking the status of the group's administration requests.

To resume replication activity for a master group:

1. Expand the database node that contains the master group that you want to resume replication on.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Select the target master group that you want to resume replication on.

   The property page for the selected master group will appear in the right pane of the Replication Manager user interface.
5. Press the **Resume** button.

After resuming replication activity for a master group, monitor the status of the master group to ensure that replication activity resumes without errors.

API Equivalent: DBMS_REPCAT.RESUME_MASTER_ACTIVITY

Adding Objects to a Master Group

After suspending replication activity of a master group, you can identify new replication objects for the group. Oracle lets you replicate tables, views, synonyms, indexes, triggers, procedures, functions, and packages as part of a master group. To add one or more objects to a master group:

1. Expand the database node that contains the master group that you want to add an object to.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Select the target master group that you want to add an object to.

   The property page for the selected master group will appear in the right pane of the Replication Manager user interface.
5. Press the **Objects** page.

   You will see a list of all objects that are currently contained in the selected master group (the schema, name, and type properties will be displayed for each object).
6. Press the **Add** button to add an object or objects to the selected master group.

   The **Add object(s) to group** dialog box will appear.
7. Select the schema that contains the object that you want to add to the selected master group.
8. Enable the checkbox that correspond with the type(s) of objects that you want to add in the **Objects to display** object group.

   A list of available objects will be displayed in the **Available Objects** list.
9. Select the object(s) that you want to add to your master group (press the <SHIFT> key to select a range of objects or press the <CTRL> key to individually select multiple objects).
10. Press the **Down Arrow** to add the selected objects to the **Selected Objects** list.
11. When you have completed adding new objects, press the **OK** button.

You will see the new objects appear in the **Objects** page of the Master Group property sheet.

12. Press the **Apply** button.

Warnings: To avoid name conflicts for generated objects, the name of a replicated table should not exceed 27 bytes. Also, do not explicitly replicate indexes that correspond to PRIMARY KEY and UNIQUE constraints for replicated tables in a master groups. Oracle automatically replicates all table constraint definitions, which in turn replicates indexes that are necessary to enforce constraints.

When adding an object to a master group, you must also consider the following administrative operations:

- How to replicate the object definition (and data) at all master sites. See "Replicating Object Definitions to Master Sites" for more information about this topic.
- When adding a table, how to replicate table data at all master sites. See "Replicating Table Data to Master Sites" for more information about this topic.
- When adding a table, how to configure conflict resolution for the table. See Chapter 6, "Conflict Resolution" for more information about this topic.

API Equivalent: DBMS_REPCAT.CREATE_MASTER_REPOBJECT

Designating an Alternate Key for a Replicated Table

Oracle must be able to uniquely identify and match corresponding rows at different sites during data replication. Typically, Oracle's advanced replication facility uses the primary key of a table to uniquely identify rows in the table. When a table does not have a primary key, you must designate an alternate key—a column or set of columns that Oracle can use to identify rows in the table during data replication**.**

Warning: Applications should not be allowed to update the identity columns of a table to ensure that Oracle can identify rows and preserve the integrity of replicated data.

When you create a new master group along with a table that does not have a primary key, or attempt to add to a master group a table that does not have a primary key, Replication Manager automatically displays the **Set Alternate Key Columns** dialog so that you can identify an alternate identity column or set of columns for the replicated table.

API Equivalent: DBMS_REPCAT.SET_COLUMNS

Datatype Considerations for Replicated Tables

Multimaster replication supports the replication of tables with columns that use the following datatypes: NUMBER, DATE, VARCHAR2, CHAR, NVARCHAR2, NCHAR, RAW, ROWID.

Oracle also supports the replication of tables with columns that use the following large object types: binary LOBs (BLOBs), character LOBs (CLOBs), and national character LOBs (NCLOBs). The deferred and synchronous remote procedure call mechanism used for multiple master replication propagates only the piece-wise changes to the supported LOB datatypes when piece-wise updates and appends are applied to these LOB columns.

Note: Oracle8*i* does not support replication of LOB datatypes in replication environments where some sites are running Oracle7 release 7.3.

Oracle does not support the replication of columns that use the LONG and LONG RAW datatypes. Oracle simply omits columns containing these datatypes from replicated tables.

Oracle also does not support user-defined object types and external or file-based LOBs (BFILEs). Attempts to configure tables containing columns of these datatypes as masters will return an error message.

Replicating Object Definitions to Master Sites

When you add an object to a master group, Replication Manager prompts you whether to "use existing object if present."

Allowing Oracle to Create Objects By default, when you add an object to a group at the master definition site, Oracle can use the definition of the object to create the same object at all master sites. This option requires less administrative work but creates network traffic due to initial object creation.

Note: When you add a partitioned table (or index) to a master group, Oracle also replicates the table’s partitions to all other master sites. When a master site does not have tablespaces with the same names as those in the master definition site, Oracle creates the replicated table’s partitions at the master site using the default tablespace of its schema.

Manually Creating Objects Before adding an object to a group at the master definition site, you can manually create an identical object definition at each master site. Later, when you add the object to the group, Oracle can use the existing objects and forego creating the object at each master site.

Manual creation of replication objects helps to minimize network traffic when you are configuring large replication environments. You might also have to consider this option when a master group contains tables with circular dependencies or a specific table contains a self-referential constraint.

When you choose to precreate replication objects, consider the following issues:

- When you choose to precreate a replicated table yourself, you are responsible for ensuring that the "shape of the table" (number, names, and datatypes of all table columns) is identical at all sites. Oracle returns an error when a table at a master site is not the same shape as the table at the master definition site.
- When you choose to precreate a nontable replication object yourself, you are responsible for ensuring that the SQL definition of the object is identical at all sites. Oracle returns an error when an object at the master site does not have the same SQL definition as the object at the master definition site.

Replicating Table Data to Master Sites

When you add a table to a master group, Replication Manager prompts you whether to "copy row data".

Allowing Oracle to Replicate Table Data By default, when you add a table to a group at the master definition site, Oracle can replicate the data of the master definition site table to the table at all master sites. This option requires less administrative work but creates network traffic due to initial object creation.

Manually Loading Table Data Before adding a table to a group at the master definition site, you can precreate an identical table structure at each master site and then manually load identical data into each table replica. Later, when you add the object to the group at the master definition site, Oracle can use the existing table replicas and forego creating and replicating table data at each master site. This option is appropriate when you are configuring large tables and want to minimize the network traffic due to initial object creation and data replication.

When you choose to populate a replicated table at a master site yourself, you are responsible for ensuring that the table data is consistent among all replicas in the system. For example, when manually populating replicated tables with data, do so before adding the table to its master group. Furthermore, prevent applications from accessing the replicated table until the table is added to a master group and replication activity is resumed; otherwise, the table might become inconsistent at the various master sites.

Offline Instantiation If you are currently replicating a large amount of data and want to add a new site to the system, you should consider offline instantiation. For complete information about offline instantiation, see "Snapshot Cloning and Offline Instantiation".

Altering Objects in a Master Group

To alter the definition of a replication object in a master group, you should *always* use Replication Manager (or an equivalent API call). Use of Enterprise Manager or a SQL DDL command (for example, ALTER TABLE) to directly alter an object in a replicated environment does not necessarily propagate DDL changes to the object at all sites in the system.

Note: Local customization of individual replicas at snapshot or master sites is outside the scope of Oracle's advanced replication facility. As a replication administrator, you must ensure that local customizations do not interfere with any global customizations done with Replication Manager.

After successfully suspending replication activity for a master group, alter the definition of an object in the group as follows:

1. Expand the database node that contains the object that you want to alter.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Expand the master group that contains the object that you want to alter.
5. Expand the **Replicated Objects** node.
6. Expand the object type node that matches the type of object that you want to alter.
7. Expand the schema that contains the object that you want to alter.
8. Select the object that you want to alter.

The property sheet for the selected replicated object will appear in the right pane of the Replication Manager user interface.

9. Press the **Alter Object** page.
10. Enter the DDL to alter your replicated object in the **Enter DDL Text** box.
11. Press the **Apply** button when you have completed defining your DDL to alter the selected object.

API Equivalent: DBMS_REPCAT.ALTER_MASTER_REPOBJECT

Identifying Subset Columns

In order to support column subsetting with deployment templates, you need to identify which columns of your replicated table will not be checked for conflicts during update and/or delete operations on the target table.

Complete the following steps to mark the necessary columns:

1. Expand the **Configuration** node.
2. Expand the **Master Groups** node.
3. Expand the master group that contains the target replicated table.
4. Expand the **Replicated Object** node.
5. Expand the **Tables** node.
6. Expand the schema that contains the target replicated table.
7. Select the target replicated table.

   The replicated object property sheet will appear in the right pane of the Replication Manager user interface.
8. Press the **Column Subsetting** tab.
9. Select the column group that contains the column that you want to mark.

   If you have not explicitly defined any column groups, select the "Shadow Group."

10. Disable the **Delete** and/or **Update** checkbox next to the column(s) that you do not want checked for conflicts.

    You will not be able to disable the **Delete** or **Update** checkboxes for primary key columns.

11. Press the **Apply** button.

    You will now be able to exclude the columns identified in step 10 above from being replicated to updateable snapshots when using a deployment template).

Considerations

Consider the following issues before and after altering an object in a master group:

- Before you alter a replicated table that has dependent updateable snapshots, be sure to push all changes from the snapshot site.
- After altering a replication object, you might need to regenerate replication support for the object.
- After altering a replication object, check the administration requests at the master definition site to be sure that the object was successfully modified at each master site. The DDL changes to the object and any supporting objects are asynchronously applied at each master site.
- After altering a replicated table that has dependent snapshots, you must drop and re-create the snapshots. All other objects at a snapshot site will be automatically re-created the next time that a dependent snapshot is refreshed.

Removing Objects from a Master Group

To remove objects from a master group:

1. Expand the database node that contains the master group that you want to remove an object from.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Select the target master group that you want to remove an object from.

   The property page for the selected master group will appear in the right pane of the Replication Manager user interface.
5. Press the **Objects** page.

You will see a list of all objects that are currently contained in the selected master group (the schema, name, and type properties will be displayed for each object).

6. Select the object or objects that you want to remove (press the <SHIFT> key to select a range of objects or press the <CTRL> key to individually select multiple objects).
7. Press the **Remove** button to remove selected objects from the master group.

   A **Drop Object** confirmation dialog box will allow you to confirm that you want to remove the selected objects.
8. Press the **Apply** button.

Note: Before dropping an object from a master group, ensure that no snapshots depend on the object.

Note: When you drop a replication object from a master group, Replication Manager automatically removes all corresponding system-generated objects that were necessary to support the replication object.

API Equivalent: DBMS_REPCAT.DROP_MASTER_REPOBJECT

Adding a Master Site to a Master Group

Before adding a new master site to a master group, you must:

- Prepare the new master site with the necessary replication administrator, propagator, receiver, schemas, and links to existing sites.
- Modify the existing sites so that they are aware of the new site.

To prepare a multimaster replication system for the addition of a new master site, use the Replication Manager setup wizard. When using the setup wizard, consider the following issues:

- You must specify all master sites in the system, including all new master sites as well as the sites that the system already supports. To specify master sites that already exist in the system, use the **Browse** button on the **Create Master Sites** page of the wizard. To specify new master sites, use the **New** button of the same page.

- The wizard will ask you to enter the passwords for the SYSTEM accounts and existing propagators in the system.
- Optionally, you can choose to change system settings in the multimaster environment, including the administrator and/or propagator/receiver accounts, and default and site-specific propagation settings.

Note: See "The Replication Setup Wizard" for more information about using the setup wizard for multimaster configuration.

After you use the setup wizard to prepare a multimaster replication system for the addition of a new master site, you are ready to add the new master site to the group. After suspending replication activity of a master group, add a new destination to a master group:

1. Expand the database node that contains the master group that you want to add a destination to.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Select the target master group that you want to add a destination to.

   The property page for the selected master group will appear in the right pane of the Replication Manager user interface.
5. Press the **Master Sites** page.

   You will see a list of all destinations that are currently defined for the selected master group.
6. Press the **Add** button to add a destination to the selected master group.

   The **Add database destinations to the group** dialog box will appear.
7. Select the radio-button that corresponds with the type of database link that you want to use to define the new destination.
8. Select the database link(s) that you want to use from the **Available Links** list (press the <SHIFT> key to select a range of database links or press the <CTRL> key to individually select multiple database links).

9. Press the **OK** button when you have completed selecting the database links.

   The **Add Destination to Group** dialog box will appear.

10. Modify your destination properties as needed and press the **OK** button (press the **OK All** button to accept the default settings for all new destinations).

    When you return to the **Master Sites** page of the Master Group property sheet, you will see the new destinations listed in the **Master Sites** list.

11. Press the **Apply** button.

Note: When adding a master site to a master group that contains tables with circular dependencies or a specific table that contains a self-referential constraint, you must precreate the tables at the master site and manually load data at the new site. See "Replicating Object Definitions to Master Sites" for more information.

API Equivalent: DBMS_REPCAT.ADD_MASTER_DATABASE

Removing a Master Site from a Master Group

After suspending replication activity of a master group, you can remove destinations (master sites) from the group. To remove a master site destination from a master group:

1. Expand the database node that contains the master group that you want to remove a destination from.

2. Expand the **Configuration** node.

3. Expand the **Master Groups** node.

4. Select the target master group that you want to remove a destination from.

   The property page for the selected master group will appear in the right pane of the Replication Manager user interface.

5. Press the **Master Sites** page.

   You will see a list of all destinations that are currently defined for the selected master group in the **Master Sites** list.

6. Select the destination or destinations that you want to remove from the master group (press the <SHIFT> button to select a range of destinations or press the <CTRL> key to individually select multiple destinations).
7. Press the **Remove** button to remove a destination from the selected master group.
8. Press the **Apply** button.

API Equivalent: DBMS_REPCAT.REMOVE_MASTER_DATABASES

Special Circumstances

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 will not be able to suspend replication activity for the master group. 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 in the following order as soon as possible after the unavailable master sites are removed:

1. Suspend replication activity for the master group.
2. Remove all deferred transactions from each master site where the destination for the transaction is a removed master site.
3. Delete all deferred transactions from removed master sites.
4. Re-execute or delete all error transactions at each remaining master site.
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. Use the DBMS_RECTIFIER_DIFF package to determine and fix differences.
7. Resume replication activity for the master group.

Note: After dropping an unavailable master site from a master group, you should also remove the master group from the site to finish cleanup.

Generating Replication Support for Master Group Objects

After performing administrative operations for a master group, Oracle must generate replication support for your changes before you can resume replication activity for the group. For example, after you add a table to a master group, Oracle must activate internal triggers and packages before it can support the replicated table. When you later add conflict resolution to the table, you must regenerate replication support for the table so that all master sites use the same conflict resolution methods for the table.

Note: To display the status of a replication object, click on the master group that contains the object. The **Status** field displays the status of each replication object in the group. When an object's status is "Valid", no action is necessary; however, when an object's status is "Needs Gen," you should generate replication support for the object.

Oracle generates replication support for an object using two phases:

1. When you generate replication support for an object, Oracle begins Phase One by synchronously broadcasting the request to all sites to create the necessary generated packages. These packages are created asynchronously. For procedural replication, Phase One generates the package specification.
2. Phase Two does not begin until each site indicates to the master definition site that it has generated the packages necessary to support replication. Oracle then begins Phase Two by synchronously broadcasting the request to activate the necessary internal triggers at each site. (For Oracle7 release 7.3 sites, the broadcast request generates the necessary PL/SQL triggers and their associated packages.) Once again, these objects are created asynchronously. For procedural replication, Phase Two generates the package body.

Note: Oracle is optimized to allow additional generation requests and to allow the creation of a master group to proceed after Oracle has broadcast the request to create the packages at each site. It is not necessary to wait until all packages have actually been created at all of the sites to begin processing these types of requests. New administration requests do not execute until after Oracle completes the second phase for generating replication support.

Generating Replication Support for Individual Objects

To generate replication support for an individual object in a master group:

1. Expand the database node of the master definition site that contains the object you want to generate replication support for.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Expand the master group that contains the object that you want to generate replication support for.
5. Expand the **Replicated Objects** node.
6. Expand the object type node that matches the target object type.
7. Expand the schema that contains the target object.
8. Select the object that you want to generate replication support for.

   The property sheet for the selected replicated object will appear in the right pane of the Replication Manager user interface.
9. Press the **Generate** button.

Note: After generating replication support for one or more objects, you can ensure that the operation was successful by checking the status of the object using Replication Manager.

Generating Replication Support for All Master Group Tables

To generate replication support for all tables in a master group:

1. Expand the database node of the master definition site that contains the master group that you want to generate replication support for.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Select the target master group that you want to generate replication support for.

   The property page for the selected master group will appear in the right pane of the Replication Manager user interface.
5. Press the **Operations** page.
6. Press the **Generate** button.

API Equivalents: DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT

Note: After generating replication support for one or more objects at the master definition site, you can ensure that the operation was successful by checking the status of the object using Replication Manager.

Minimizing Data Propagation

The **Min(imize) Communications** setting of the **Edit Replication Object** property sheet determines how much data sites must transfer to perform conflict detection for a table. This setting is valid only for Oracle8 and greater databases and is available only when using the database connection to the group's master definition site.

Note: If any master sites in your replicated environment are running Oracle7 release 7.3, this setting must be disabled. When disabled, Oracle propagates the old and new values of all columns in a row when any column in the row is updated. This is the behavior expected by Oracle7 release 7.3.

When **Min(imize) Communications** is enabled, the default, Oracle propagates only the new values for updated columns plus the old values of the primary key and the columns in each updated column group.

Additional Information: To learn about additional techniques that minimize data propagation, see "Minimizing Data Propagation for Update Conflict Resolution" on page 6-42.

Viewing Information About Master Groups

Replication Manager can display information about the master groups in an advanced replication system.

Listing Master Groups

To display a list of all master groups at a site:

1. Expand the database node that contains the master groups that you want to view.
2. Expand the **Configuration** node.
3. Select the **Master Groups** node.

All master groups of the currently selected database will be displayed in the right pane of the Replication Manager user interface. The master group name, master definition site status (is the selected site the master definition site), master group status, link qualifier, and any remarks will be displayed for each listed master group.

Listing Objects for a Master Group

To display a list of all objects in a master group at a site:

1. Expand the database node that contains the master group that you want to view.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Expand the master group that contains the objects that you want to view.
5. Select the **Replicated Objects** node.

For each object in the target master group, the detail panel lists the name of the object, the schema that contains the object, the type (table, index, procedure, and so on) of the object, and the status (for example, valid or needs generation of replication support).

Displaying a Destination Map for a Master Group

Replication Manager uses a destination map to represent visually the configuration of a master group in an advanced replication environment. To display the destination map for a master group at a master site:

1. Expand the database node that contains the master group that you want to view.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Expand the master group that to view a destination map for.
5. Select the **Destination Map** node.

A destination map for a master group provides the following visual information about the master group:

- The master definition and master sites of the master group.
- The propagation methods between each master site.

A destination map also lets you edit the properties for the scheduled links that appear between master sites. To edit a link in a destination map, use the Edit Database Destination property sheet of Replication Manager. To access the dialog, right-click on the scheduled link and select Edit Destination.

Use the Edit Database Destination property sheet to:

- Change the propagation mode for the scheduled link.
- Reschedule the link’s propagation interval.

Displaying Generated Objects Associated with a Master Group

To display the generated objects associated with the replication objects in a master group at a master site:

1. Expand the database node that contains the generated objects that you want to view.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Expand the master group that contains the generated objects that you want to view.
5. Select the **Generated Objects** node.

All generated objects contained in the selected master group will appear in the right pane of the Replication Manager user interface. Object name, schema, object type, status, and any remarks will be displayed for each generated object.

Data Dictionary Views

In addition to using Replication Manager to view information about an advanced replication environment, you can also use the following data dictionary views.

- REPGROUP
- REPSITES
- REPOBJECT
- REPCATLOG

Other Master Site Administration Issues

The preceding sections of this chapter explained the most commonly performed administrative procedures that involve master groups. For additional information on less commonly performed administrative procedures for master groups, see "Advanced Management of Master and Snapshot Groups" on page 7-2.

Advanced Multimaster Replication Options

The following sections explain some additional topics to consider when building and managing a multimaster replication system:

- Planning for Parallel Propagation.
- Understanding Replication Protection Mechanisms.

Planning for Parallel Propagation

When you create the scheduled links for an advanced replication environment, each link can asynchronously propagate changes to a destination using either serial or parallel propagation.

- With serial propagation, Oracle propagates replicated transactions one at a time in the same order that they are committed on the source system. To configure a schedule link with serial propagation, use the **Create Scheduled Link** or **Edit Scheduled Link** property sheet, and disable the **Parallel Propagation** setting of the **Options** page.

- With parallel propagation, Oracle propagates replicated transactions using multiple parallel streams for higher throughput. When necessary, Oracle orders the execution of dependent transactions to preserve data integrity. To configure a scheduled link with parallel propagation, use the **Create Scheduled Link** or **Edit Scheduled Link** property sheet, and enable the **Parallel Propagation** setting of the **Options** page. Set **Processes** to the desired degree of parallelism (1 or greater).

Parallel propagation uses the pool of available parallel server processes. This is the same facility Oracle uses for other parallel operations such as parallel query, parallel load, and parallel recovery. Each server process propagates transactions through a single stream. A parallel coordinator process controls these server processes. The coordinator tracks transactions dependencies, allocates work to the server processes, and tracks their progress.

Parallel server processes remain associated with a parallel operation throughout the execution of that operation. When the operation is complete, those server processes become available to process other parallel operations. For example, when Oracle performs a parallel push of the deferred transaction queue to its destination, all parallel server processes used to push the queue remain dedicated to the operation until it completes.

To configure a pool of parallel server processes for a server properly, you must consider several issues related to the configuration of an advanced replication system.

- When you configure all scheduled links to use serial propagation, the replication system does not use parallel server processes. Therefore, you do not have to adjust the size of any server's pool of parallel servers to account for replication.
- When you configure one or more scheduled links to use parallel propagation, you must consider the number of parallel server processes that each link will use to push changes to its destination. Furthermore, you should also consider how long each push will hold parallel servers from being used by other operations. For example, when you configure a scheduled link for continuous propagation (with a large value for **Delay Seconds**), Oracle holds on to the parallel server processes used to push transactions to its destination. Therefore, you should increase the number of parallel server processes for the corresponding database server to ensure that there is a sufficient number of processes for other parallel operations on the server.

To configure a database server's pool of parallel query processes, use the following initialization parameters:

- PARALLEL_MAX_SERVERS.
- PARALLEL_MIN_SERVERS.
- PARALLEL_SERVER_IDLE_TIME.

Additional Information: See the book *Oracle8i Concepts*.

Understanding Replication Protection Mechanisms

Oracle ensures that transactions propagated to remote sites are never lost and never propagated more than once, even when failures occur.

- Multiple procedure calls submitted within a single local transaction are executed within a transaction remotely.
- If the network or remote database fails during propagation, the transaction is rolled back at the remote site and the transaction remains in the local queue until the remote database becomes accessible again and can be successfully propagated.
- A transaction is not removed from the queue at the local site until it is successfully propagated and applied to all of its destination sites.

Note: Successful propagation does not necessarily imply successful application of the transaction at the remote site. Errors such as unresolvable conflicts or running out of storage space can cause the transaction to result in an error, which the remote site keeps track of. See "Displaying Error Transactions" for more information about viewing and managing error transactions.

Protection against failures is provided for both serial and parallel propagation.

- In the case of serial propagation, the advanced replication facility uses two-phase commit.
- In the case of parallel propagation, the advanced replication facility uses a special-purpose distributed transaction protocol optimized for parallel operations. The remote site keeps track of the transactions that have been propagated successfully and periodically sends this information back to the

local site. The local site records this information and purges the entries in its local queue that have been propagated to all destination sites. In case of failures, the local site asks the remote site for information on the transactions that have been propagated successfully so that propagation can continue at the appropriate point.

Data Propagation Dependency Maintenance

Oracle maintains dependency ordering when propagating replicated transactions to remote systems. For example,

1. Transaction A cancels an order.
2. Transaction B sees the cancellation and processes a refund.

Transaction B is dependent on Transaction A because Transaction B *sees the committed update* cancelling the order (Transaction A) on the local system.

Oracle propagates Transaction B (the refund) *after* it successfully propagates Transaction A (the order cancellation). Oracle applies the updates that process the refund *after* it applies the cancellation.

Parallel Propagation Dependency Tracking When Oracle on the local system executes a new transaction,

1. Oracle records the system commit number of the most recent transaction that updated data seen by the new transaction as the *dependent system commit number.*
2. Oracle ensures that transactions with system commit numbers less than or equal to the *dependent system commit number* propagate successfully to the remote system.
3. Oracle propagates the awaiting, dependent transaction.

Note: When there are no possible dependencies between transactions, Oracle propagates transactions in parallel.

Parallel propagation maintains data integrity in a manner different from that of serial propagation. With serial propagation, Oracle applies all transaction in the same order that they commit on the local system to maintain any dependencies. With parallel propagation, Oracle tracks dependencies and executes them in commit order when dependencies can exist; in parallel when dependencies cannot exist. With both serial and parallel propagation, Oracle preserves the order of execution within a transaction. The deferred transaction executes every remote procedure call at each system in the same order as it was executed within the local transaction.

Note: A single coordinator process exists for each database link to a remote site. Each database link to the same remote site requires a different connection qualifier.

Additional Information: See "Using Link Qualifiers for a Master Group".

Minimizing Transaction Dependencies to Improve Parallelism Certain application conditions can establish dependencies among transactions that force Oracle to serialize the propagation of deferred transactions. When several unrelated transactions modify the same data block in a replicated table, Oracle serializes the propagation of the corresponding transactions to remote destinations.

To minimize transaction dependencies created at the data block level, you should try to avoid situations that concentrate data block modifications into one or a small number of data blocks. For example:

- When a replicated table experiences a high degree of INSERT activity, you can distribute the storage of new rows into multiple data blocks by creating multiple free lists for the table.
- If possible, avoid situations where many transactions all update the same small table. For example, a poorly designed application might employ a small table that transactions read and update to simulate sequence number generation for a primary key. This design forces all transactions to update the same data block. A better solution is to create a sequence and cache sequence numbers to optimize primary key generation and improve parallel propagation performance.

3 Snapshot Concepts & Architecture

This chapter explains the concepts and architecture of Oracle Snapshots. This chapter covers the following topics:

Snapshot Concepts

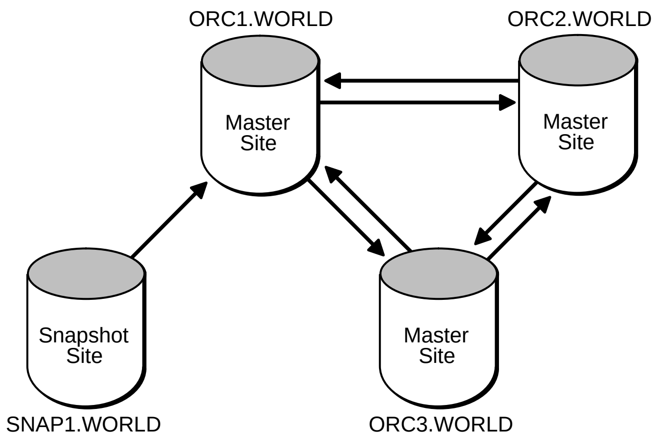
Oracle uses *snapshots*, sometimes referred to as materialized views, to meet the requirements of delivering data to non-master sites in a replicated environment and caching "expensive" queries from a data warehouse. This chapter, and the *Oracle8i Replication* manual in general, will discuss snapshots for use in a replicated environment.

To learn more about materialized views for data warehousing, see the *Oracle8i Tuning* book.

What is a Snapshot?

A snapshot is a replica of a target master table from a single point-in-time. Whereas in multimaster replication tables are continuously being updated by other master sites, snapshots are updated by one or more master tables via individual batch updates, known as a *refresh*, from a single master site (Figure 3–1).

Figure 3–1 Snapshot Connected to a Single Master Site in a Replicated Environment



Snapshots also have the option of containing a WHERE clause so that snapshot sites can contain custom data sets, which can be very helpful for regional offices or sales forces that don't require the complete corporate data set.

Why use Snapshots?

Oracle offers a variety of snapshots to meet the needs of many different replication (and non-replication) situations; each of these snapshots will be discussed in detail in following sections. You might use a snapshot to achieve one or more of the following:

- Ease Network Loads
- Mass Deployment
- Data Subsetting
- Disconnected Computing

Ease Network Loads

If one of your goals is to reduce network loads, you can use snapshots to distribute your corporate database to regional sites; instead of the entire company accessing a single database server, user load is distributed across multiple database servers.

While multimaster replication also distributes a corporate database to multiple sites, the networking requirements are greater than replicating with snapshots because of the transaction by transaction nature of multimaster replication. Since multimaster replication can provide real-time or near real-time results, network traffic is much greater, resulting in the need for a dedicated network link.

Snapshots are updated via an efficient batch process from a single master site and have less network requirements and dependency than multimaster replication because of the point-in-time nature of snapshot replication. In addition to not requiring a dedicated network connection, replicating data with snapshots increases data availability by providing local access to the target data. These benefits, combined with mass deployment and data subsetting (both of which also reduce network loads), will greatly enhance the performance and reliability of your replicated database.

Mass Deployment

Deployment templates allow you to precreate a snapshot environment locally. Deployment templates allow you to quickly and easily deploy snapshot environments to support sales force automation and other mass deployed environments. Parameters allow you to create custom data sets for individual users without changing the deployment template. This technology allows you to rollout a database infrastructure to hundreds or thousands of users.

Data Subsetting

Snapshots allow you to replicate data based on column and/or row-level subsetting (remember that multimaster replication requires replication of the entire table). Data subsetting allows you to replicate information that only pertains to a particular site. For example, if you have a regional sales office, you might replicate only the data that is needed in that region, thereby cutting down on unnecessary network traffic.

Disconnected Computing

Unlike multimaster replication, snapshots do not require a dedicated network link. Though you have the option of automating the refresh process by scheduling a job, you can manually refresh your snapshot on-demand. This is an ideal solution for sales applications running on a laptop. For example, a developer can integrate the Oracle Replication Management API to refresh on-demand into the sales application. When the sales person has completed the day's order, they simply dial-up the network and use the integrated mechanism to refresh the database, thus transferring the orders to the main office.

Available Snapshots

As previously mentioned, there are several types of snapshots available to meet a variety of distributed database needs. The following sections describe each snapshot and also describe some environments for which they are best suited.

Primary Key

Primary key snapshots are considered the normal (default) type of snapshot. Primary key snapshots are updateable if the snapshot has been created as part of a snapshot group (see "Snapshot Groups" on page 3-18) and "FOR UPDATE" was specified when defining the snapshot. Changes are propagated according to the row changes as identified by the primary key value of the row (vs. the ROWID). The SQL command for creating an updateable, primary key snapshot might look like:

```
CREATE SNAPSHOT sales.customers FOR UPDATE AS
 SELECT * FROM sales.customers@dbs1.acme.com;
```

Primary key snapshots may contain a subquery so that you can create a horizontally partitioned subset of data at the remote snapshot site. This subquery may be as simple as a basic WHERE clause or as complex as a multilevel WHERE EXISTS clause. Primary key snapshots that contain a selected class of subqueries can still be incrementally or *fast* refreshed (see "Snapshots with Subqueries" on page 3-8 for more information). The following is a subquery snapshot with a WHERE clause containing a subquery:

```
CREATE SNAPSHOT sales.orders REFRESH FAST AS
 SELECT * FROM sales.orders@dbs1.acme.com o
 WHERE EXISTS
   (SELECT 1 FROM sales.customers@dbs1.acme.com c
    WHERE o.c_id = c.c_id AND zip = 19555);
```

ROWID

For backwards compatibility, Oracle supports ROWID snapshots in addition to the default, primary key snapshots. A ROWID snapshot is based on the physical row identifiers (ROWIDs) of the rows in a master table. ROWID snapshots should be used only for snapshots based on master tables from an Oracle7 database, and should not be used when creating new snapshots based on master tables from Oracle8 or greater databases (see "Snapshot Log" on page 3-15 for more information on the differences between a ROWID and Primary Key snapshot).

```
CREATE SNAPSHOT sales.customers REFRESH WITH ROWID AS
 SELECT * FROM sales.customers@dbs1.acme.com;
```

Complex

If your snapshot does not need to be fast refreshable, then you can create a complex snapshot that allows for a defining SELECT statement that might contain an aggregate or a set operation. Specifically, a snapshot is considered complex when the defining query of a snapshot contains:

- A distinct or aggregate function
- A CONNECT BY clause
- Snapshots with subqueries that don't comply with the requirements detailed in Table 3–1, "Restrictions for Snapshots with Subqueries"
- A set operation (UNION, INTERSECT, or MINUS)

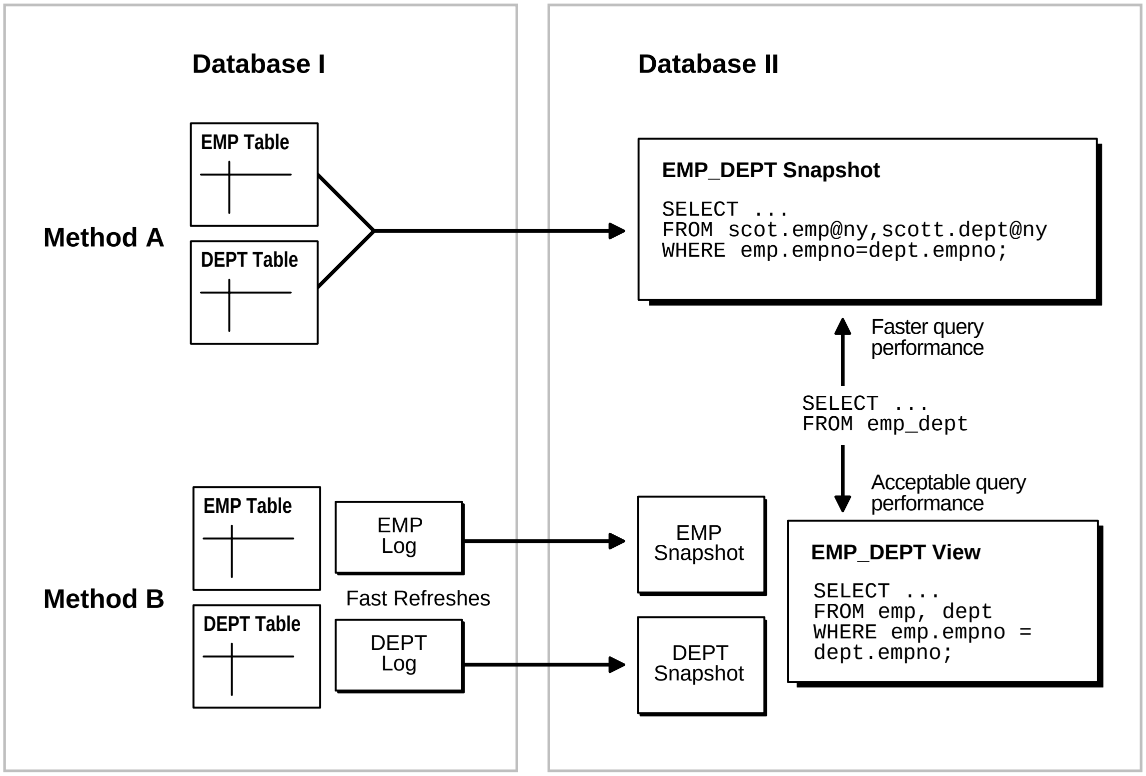
Note: In most cases, you should avoid using complex snapshots since they cannot be fast refreshed, which may degrade network performance (see "Refresh Process" on page 3-22 for information).

A sample complex snapshot CREATE statement might look like the following:

```
CREATE SNAPSHOT scott.snap_employees AS
 SELECT emp.empno, emp.ename FROM scott.emp@dbs1.acme.com
  UNION ALL
 SELECT new_emp.empno, new_emp.ename FROM scott.new_emp@dbs1.acme.com;
```

A Comparison of Simple and Complex Snapshots For certain applications, you might want to consider the use of a complex snapshot. Figure 3–2 and the following text discuss some issues that you should consider.

Figure 3–2 Comparison of Simple and Complex Snapshots



- **Complex Snapshots**: Method A in Figure 3–2 shows a complex snapshot. The snapshot in Database II exhibits efficient query performance because the join operation has already been completed during the snapshot’s refresh. However, complete refreshes must be performed, because it is a complex snapshot.

- **Simple Snapshot with a Joined View**: Method B in Figure 3–2 shows two simple snapshots in Database II, as well as a view that performs the join in the snapshot’s database. Query performance against the view would not be as good as the query performance against the complex snapshot in Method A. However, the simple snapshots can be more efficiently refreshed using snapshot logs.

In summary, to decide which method to use:

- If you refresh rarely and want faster query performance, use Method A.
- If you refresh regularly and can sacrifice query performance, use Method B.

Read-only Snapshots

Any of the previously described types of snapshots can be made read-only by omitting the FOR UPDATE clause (or disabling the equivalent checkbox in the Replication Manager GUI interface). Read-only snapshots use many of the same mechanisms as updateable snapshots, except that they do not need to belong to a snapshot group (see "Snapshot Groups" on page 3-18 for more information).

In addition to not needing to belong to a snapshot group, using read-only snapshots eliminates introducing data conflicts originating from a remote snapshot site, though this convenience means that updates cannot be made at the remote snapshot site. You might define a read-only snapshot as:

```
CREATE SNAPSHOT sales.customers AS
  SELECT * FROM sales.customers@hq.acme.com
```

Note: In all cases, the defining query of the snapshot must reference all of the primary key columns in the master table.

Data Subsetting with Snapshots

In certain situations, you will want your snapshot to reflect a horizontally or vertically partitioned segment of the master table's data. If you use deployment templates to build your snapshots, you can define vertical data subsets to replicate data along column boundaries (for additional information on vertical partitioning, see "Vertical Partitioning" on page 4-20). Some reasons to consider partitioning data are:

- **Reduce Network Traffic**: Only changes that satisfy the snapshot’s WHERE clause of the defining query will be propagated to the snapshot, thereby reducing the amount of data transferred, reducing network traffic.

- **Keep Sensitive Data Secure**: Users will only be able to view data that satisfies the defining query for the snapshot.
- **Reduce Resource Requirements**: If the snapshot will be located on a laptop, hard disks will generally be significantly smaller than those on a corporate server. Partitioned snapshots may require significantly less storage space.
- **Faster Refresh Times**: Since less data will be propagated to the snapshot, the refresh process will be faster. This is essential for those who need to refresh snapshots using a dial-up network connection from a laptop.

In many instances, the above objectives can be met by using a simple WHERE clause. For example, the following DDL creates a snapshot that contains information about customers who are in the 19555 zip code:

```
CREATE SNAPSHOT sales.customers AS
  SELECT * FROM sales.customers@dbs1.acme.com
  WHERE zip = 19555;
```

Snapshots with Subqueries

The above example works very well for individual snapshots that don't have any referential constraints to other snapshots. But, if you want more than just the customer information, maintaining and defining these snapshots could be difficult.

Consider the scenario where you have three tables (`Customer`, `Order`, and `Order_line`) and you want to create three corresponding snapshots that maintain the referential integrity of these master tables. If a salesperson wants to retrieve the customer information, pending orders, and the associated order lines for all customers in the 19555 zip code, the most efficient method is to create snapshots with one or more subqueries in the defining query of the snapshot.

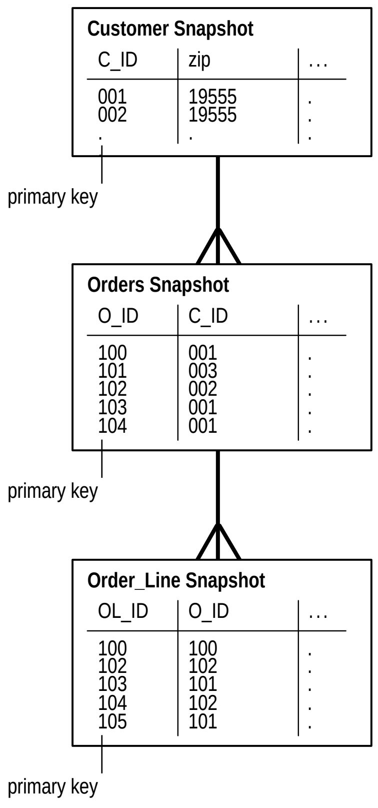
The `Customer` snapshot has a very simple defining query since the `Customer` master table is at the top of the hierarchy:

```
CREATE SNAPSHOT sales.customers AS
  SELECT * FROM sales.customers@dbs1.acme.com
  WHERE zip = 19555;
```

When you create the `Orders` snapshot, you want to retrieve all of the orders for the customers located in the 19555 zip code. If you look at the relationships in Figure 3–3, you will notice that the `Customer` and `Orders` table are related via the `C_ID` column. The following DDL will create the `Orders` snapshot with the appropriate data set:

```
CREATE SNAPSHOT sales.orders AS
  SELECT * FROM sales.orders@dbs1.acme.com o
  WHERE EXISTS
    (SELECT c_id FROM sales.customers@dbs1.acme.com c
     WHERE o.c_id = c.c_id AND c.zip = 19555);
```

Figure 3–3 Advanced subsetting with a subquery.



Creating the `Order_line` snapshot uses the same approach as the `Orders` snapshot, except that you have one additional subquery. Notice in Figure 3–3 that the `Order_line` and the `Order` tables are related via the `O_ID` row. The following DDL will create the `Order_line` snapshot with the appropriate data set:

```
CREATE SNAPSHOT sales.lines AS
  SELECT * FROM sales.order_line@dbs1.acme.com ol
  WHERE EXISTS
    (SELECT o_id FROM sales.orders@dbs1.acme.com o
     WHERE ol.o_id = o.o_id AND EXISTS
      (SELECT c_id FROM sales.customers@dbs1.acme.com c
       WHERE o.c_id = c.c_id AND c.zip = 19555));
```

The snapshots created by these three DDL statements are each fast refreshable. If new customers are identified in the target zip code, the new data will be propagated to the snapshot site during the subsequent refresh process. Likewise, if a customer is removed from the target zip code, the appropriate data will also be removed from the snapshot during the subsequent refresh process.

The subqueries in these snapshot examples walk-up the many-to-one references from the child to the parent tables. The snapshots will be populated with data that satisfies the defining query for each of these snapshots and will be refreshed only with data that satisfies these defining queries.

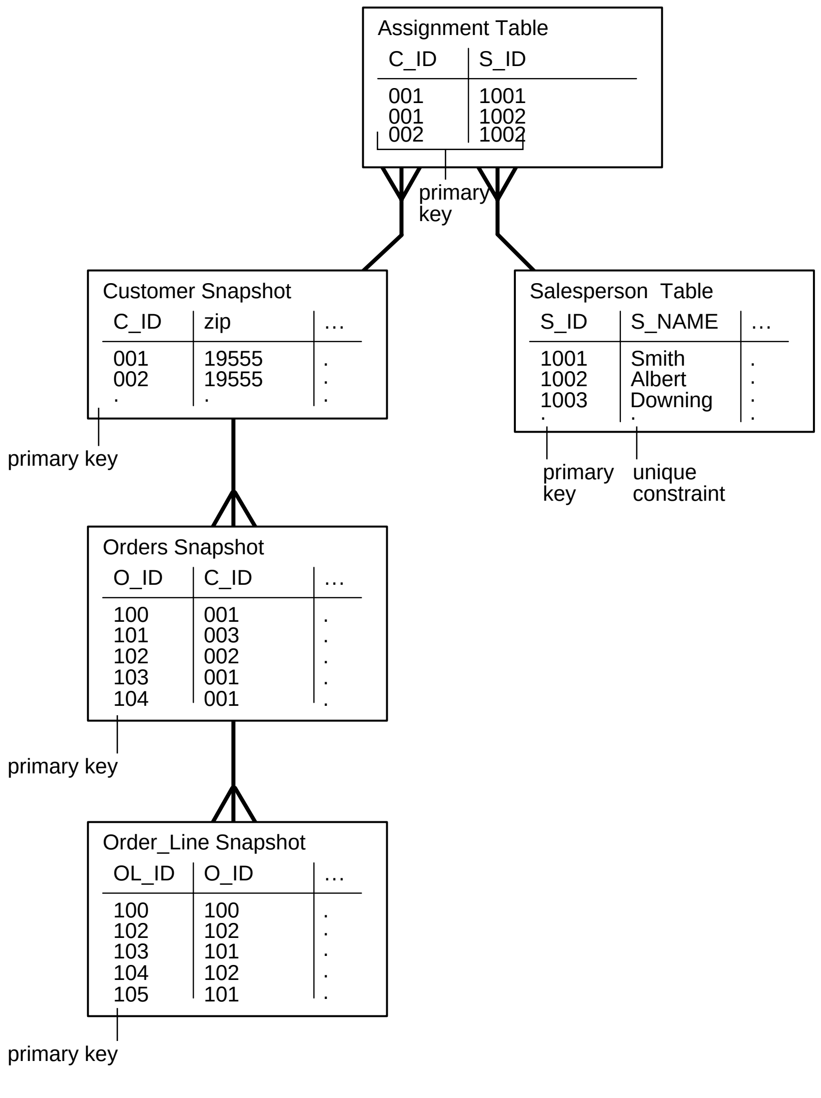
Using Assignment Tables

While the previous examples greatly enhance the flexibility of snapshots, there are certain limitations in the above example. Consider if the salesperson changed territories or the existing territory was assigned an additional zip code; the above snapshot definitions would need to be altered or recreated since the zip code 19555 was “hard coded” in the previous snapshot definitions.

With this in mind, if “assignment” tables are used in conjunction with subquery subsetting, changes to a snapshot environment can easily be controlled by the DBA. For example, consider the customer / salesperson relationship in Figure 3–4.

In this example, a salesperson is assigned their customers based on the `Assignment` table. If new salespersons are hired or other salespersons leave, the existing customers can be assigned to their new salesperson by simply modifying the contents of the assignment table. Besides creating a single point of administration, assignment tables used in conjunction with subquery subsetting enables this easy administration to remain secure. For example, salesman #`1001` will not be able to view the customer information of other salesmen (very important if the customer information contains sensitive data).

Figure 3–4 Customer/Salesperson Relationship



Considering the relationships pictured in Figure 3–4, if the `Orders` snapshot's defining query was specified as (pay special attention to the '`gsmith`' value in the last line of the `CREATE SNAPSHOT` statement):

```
CREATE SNAPSHOT sales.orders AS
 SELECT * FROM sales.orders@hq.acme.com o
    -- conditions for customers
  WHERE EXISTS
  ( SELECT c_id FROM sales.customers@hq.acme.com c
    WHERE o.c_id = c.c_id
      AND EXISTS
    ( SELECT * FROM sales.assignments@hq.acme.com a
     WHERE a.c_id = c.c_id
     AND EXISTS
     ( SELECT * FROM sales.salespersons@hq.acme.com s
       WHERE a.s_id = s.s_id AND s.s_id = 'gsmith')));
```

then the `Orders` snapshot will be populated with order data for the customers that are assigned to salesperson '`gsmith`'.

With this flexibility, managers can easily control snapshot data sets by making simple changes to the assignment table (without requiring a DBA to modify any SQL). For example, if the specified salesperson was assigned two new customers, the manager would simply assign these two new customers to the salesperson in the assignment table. After the next fast snapshot refresh, the data for these two customers will be propagated to the target snapshot site, such as the salesperson's laptop (see "Refresh Types" on page 3-22 for more information). Conversely, if a customer was taken away from the specified salesperson, all data pertaining to the specified customer would be removed from the snapshot site after the next refresh and the salesperson would no longer be able to access that information.

Restrictions for Snapshots with Subqueries

Snapshots with a subquery must be of the primary key type (see "Primary Key" on page 3-4 for more information about primary key snapshots). Additionally, the defining query of a snapshot with a subquery is subject to several other restrictions to preserve the snapshot's fast refresh capability.

Note: To determine whether a snapshot's subquery satisfies the many restrictions detailed in Table 3–1, create the snapshot with "fast refresh". Oracle will return errors if the snapshot violates any restrictions for simple subquery snapshots. If you specify "force refresh," you may not receive any errors because Oracle will automatically send data for a complete refresh.

Table 3–1 Restrictions for Snapshots with Subqueries

- ❏ Subqueries can “walk up” many-to-one references from child to parent tables when:

  A PRIMARY KEY or UNIQUE constraint exists on the join column(s) referenced in each parent table.

  The join expression uses exact match or equality comparisons (in other words, “equi-joins”).

- ❏ Subqueries can also traverse many-to-many references provided that:

  A PRIMARY KEY or UNIQUE constraint exists that includes both sets of join columns in the intersection (assignment) table.

  A PRIMARY KEY or UNIQUE constraint on the join column(s) and a separate UNIQUE constraint on the filter column(s) of the table the intersection table joins to. If there is only an intersection table, then there must be a PRIMARY KEY or UNIQUE constraint that includes both the join column(s) and filter column(s) of the intersection table.

  The subquery specifies an exact match or equality comparison.

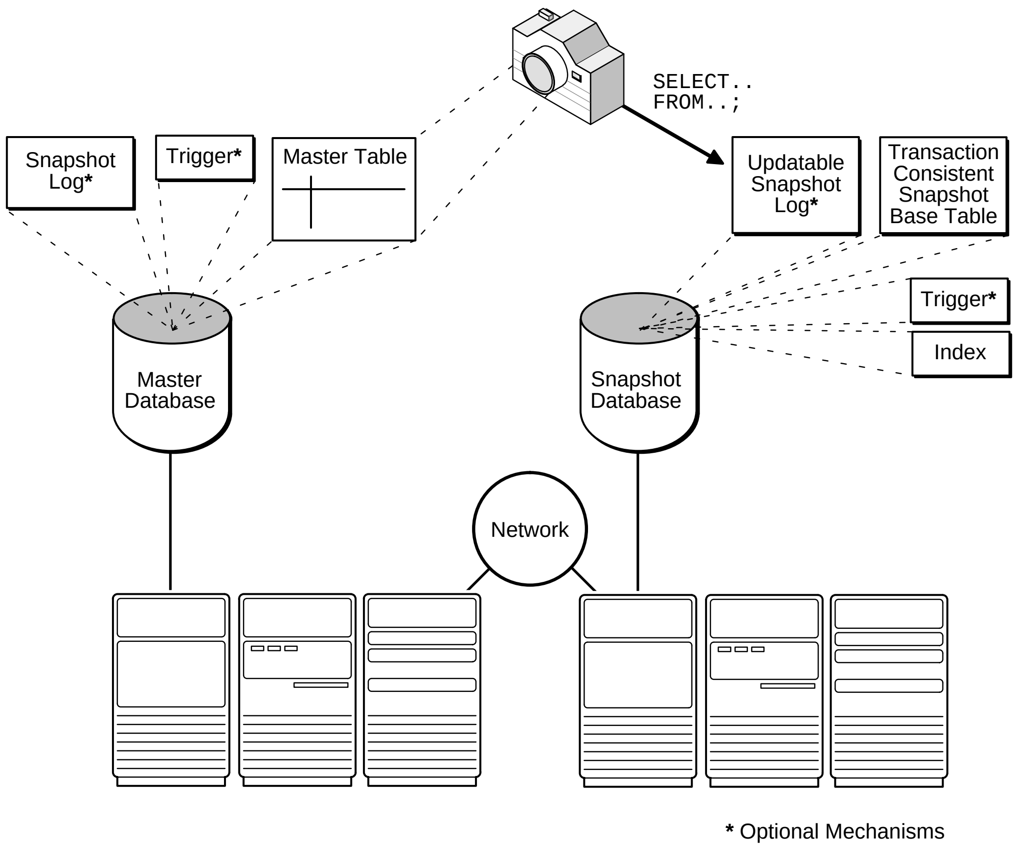
  Note: The combination of these two properties means that only one row is returned from the many-to-many reference. To illustrate this, see "Using Assignment Tables". The PRIMARY KEY constraint of the ASSIGNMENT table contains both the C_ID and S_ID join columns. The S_ID join column of the SALESPERSON table has a PRIMARY KEY constraint and the S_NAME filter column has a UNIQUE constraint. The subquery specifies that the value in S_NAME equals a constant, in this case, 'gsmith'.

- ❏ Snapshots must be primary key snapshots.
- ❏ The master table’s snapshot log must include all filter columns referenced in the subquery. For additional information about filter columns, see "Using Filter Columns" on page 3-31.
- ❏ The subquery must be a positive subquery. For example, you can use EXISTS, but not NOT EXISTS.
- ❏ The subquery must use EXISTS to connect each nested level (INs are not allowed).
- ❏ A primary key must exist for each table at each nested level.
- ❏ Each nested level can only reference the table in the above level.
- ❏ Subqueries can include AND expressions, but each OR expression may only reference columns contained within one row. Multiple OR expressions within a subquery can be ANDed.
- ❏ Each table can be joined only once within the subquery; each table can be in only one EXISTS expression.
- ❏ Each table referenced in the subquery must be located in the same master database.
- ❏ The sum of the number of selected columns plus the number of primary key columns for each table used in the subquery must be less than the maximum number of columns allowed in a table (1,000 columns).

Snapshot Architecture

The mechanisms used in snapshot replication are depicted in Figure 3–5. Some of these mechanisms are optional and are used only as needed to support the created snapshot environment. For example, if you have a read-only snapshot, then you will not have an updatable snapshot log or internal trigger at the remote site. Also, if you have a complex snapshot that cannot be fast refreshed, then you may not have a snapshot log at the master site.

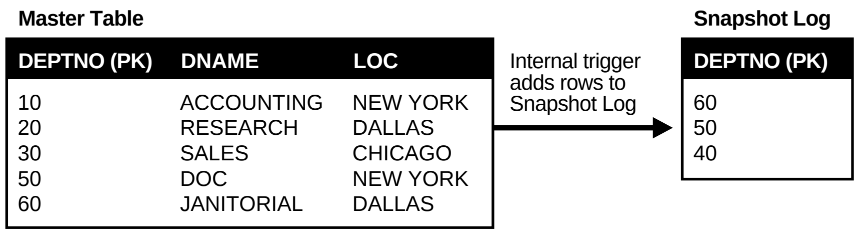
Figure 3–5 Snapshot Replication Mechanisms



Master Site Mechanisms

The three mechanisms displayed in Figure 3–6 are required at the master site to support fast refreshing of snapshots.

Figure 3–6 Master Site Mechanisms



Master Table

The master table is the basis for the snapshot and is located at the target master site. This table may be involved in both snapshot replication and multimaster replication (remember that a snapshot points to only one master site).

Changes made to the master table as recorded by the snapshot log will be propagated to the snapshot during the refresh process.

Internal Trigger

When changes are made to the master table using DML, an internal trigger records the primary key and/or ROWID of the rows affected and filter column[1] information in the snapshot log. This is an internal trigger that is automatically activated when you create a snapshot log for the target master table.

Snapshot Log

When you create a snapshot log for a master table, Oracle creates an underlying table as the snapshot log. A snapshot log holds the primary keys and optionally the ROWIDs of rows that have been updated in the master table. A snapshot log can also contain filter columns to support fast refreshes of snapshots with subqueries (see footnote 1). The name of a snapshot log's table is *MLOG$_master_table_name*. The snapshot log is created in the same schema as the target master table. A single snapshot log for a single master table can support any number of snapshots.

[1] Filter columns are required when the snapshot contains a subquery. See "Data Subsetting with Snapshots" on page 3-7 for information on subquery snapshots and "Using Filter Columns" on page 3-31 for more information.

As described in the previous section, the internal trigger adds the information to the snapshot log whenever a DML transaction has taken place at the target master table.

There are three types of snapshot logs:

- **Primary Key**: The snapshot records changes to the master table based on the primary key of the affected rows.
- **Row ID**: The snapshot records changes to the master table based on the ROWID of the affected rows.
- **Combination**: The snapshot records changes to the master tables based both on the primary key and the ROWID of the affected rows (this snapshot log supports both primary key and ROWID snapshots, which is helpful for mixed environments).

A combination snapshot log works in the same manner as the primary key and ROWID snapshot log, except both the primary key and the ROWID of the affected row are recorded.

Though the difference between snapshot logs based on primary keys and ROWIDs is very small (one records rows affected using the primary key, while the other records affected rows using the physical ROWID), the practical impact is quite large. Using ROWID snapshots and snapshot logs will make reorganizing and/or truncating your master tables very difficult as it will prevent your ROWID snapshots from being FAST refreshed. If you reorganize or truncate your master table, your ROWID snapshot will need to be COMPLETE refreshed since the ROWIDs of the master table will have changed.

Snapshot Site Mechanisms

When a snapshot is created, several additional mechanisms are created at the snapshot site to support the snapshot. Specifically, a base table, at least one index, and optionally a view are created. If you create an updateable snapshot, an internal trigger and a local log (updateable snapshot log) are also created at the snapshot site.

Base Table

Beginning with Oracle8*i* release 8.1.5, the base table is the actual snapshot (no view is required). The base table will have the name that you have specified during the creation process.

When the snapshot site compatibility setting is less than 8.1, the base table is the underlying support mechanism for the view described in the previous section (the compatibility setting is defined in the INIT.ORA file - see the *Oracle8i Reference* manual for more information). When the base table is the support mechanism for the view, it has the name SNAP$_*Snapshot_Name*.

Any indexes generated when you create the snapshot are created on the base table.

Datatype Considerations for Snapshots

Oracle supports snapshots of master table columns that use the following datatypes: NUMBER, DATE, VARCHAR2, CHAR, NVARCHAR2, NCHAR, RAW, ROWID.

Oracle also supports snapshots of master table columns that use the following large object types: binary LOBs (BLOBs), character LOBs (CLOBs), and national character LOBs (NCLOBs). However, you cannot reference LOB columns in a WHERE clause of a snapshot's defining query. The deferred and synchronous remote procedure call mechanism used for replication propagates only the piece-wise changes to the supported LOB datatypes when piece-wise updates and appends are applied to these LOB columns.

Note: Oracle8*i* does not support replication of LOB datatypes in replication environments where some sites are running Oracle7 release 7.3.

Oracle does not support the replication of columns that use the LONG datatype. Oracle simply omits the data in LONG columns from snapshots.

Oracle also does not support user-defined object types and external or file-based LOBs (BFILEs). Attempts to configure snapshots containing columns of these datatypes return an error message.

View

A view is only created to support snapshot replication with Oracle8 release 8.0 and earlier (if the snapshot sites compatibility setting is less than 8.1, then a view will always be created). If a view is created, the view will have the same name specified in the CREATE SNAPSHOT statement (i.e. `CREATE SNAPSHOT sales.snap_customers AS ...`).

Index

At least one index is created at the remote snapshot site for each snapshot created. This index corresponds to the primary key of the target master table. Additional indexes may be created by Oracle at the remote snapshot site to support fast refreshing of snapshots with subqueries. The index has the name I_SNAP$_*Snapshot_Name*.

Local Log

A local update log (USLOG$_*Materialized_View_Name*) is used to determine what data needs to be pulled from the target master table. A read-only snapshot does not require this local log.

Internal Trigger

Just like the internal trigger at the master site, the internal trigger at the snapshot site records DML changes applied to an updateable snapshot in the USLOG$_*Snapshot_Name* log.

Organizational Mechanisms

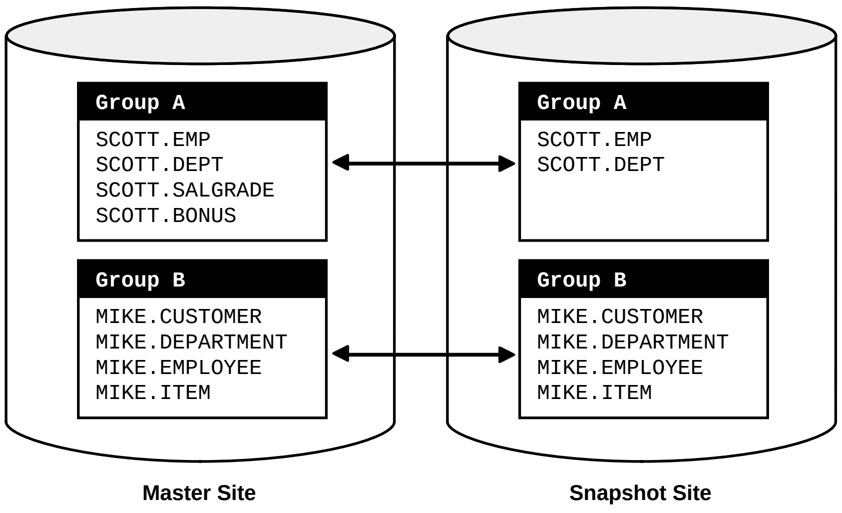
In addition to the snapshot mechanisms described in the previous section, there are several additional mechanisms that organize the snapshots at the snapshot site. These mechanisms maintain organizational consistency between the snapshot site and the master site as well as transactional (read) consistency with the target master group.

These mechanisms are *snapshot groups* and *refresh groups*.

Snapshot Groups

A *snapshot group* in an advanced replication system maintains a partial or complete copy of the objects at the target master group (snapshot groups cannot span across master group boundaries). Figure 3–7 displays the correlation between Groups A and B at the master site and Groups A and B at the snapshot site.

Figure 3–7 Snapshot Groups Correspond with Master Groups



Group A at the snapshot site (Figure 3–7) contains only some of the objects in the corresponding Group A at the master site. Group B at the snapshot site contains all objects in Group B at the master site. Under no circumstances, however, could Group B at the snapshot site contain objects from Group A at the master site. As illustrated in Figure 3–7, snapshot groups are named the same as the master groups that the snapshot group is based on. For example, a snapshot group based on a "PERSONNEL" master group will also be named "PERSONNEL."

In addition to maintaining organizational consistency between snapshot sites and master sites, snapshot groups are required for supporting updateable snapshots. If a snapshot does not belong to a snapshot group, then it can only be a read-only snapshot.

Using a Group Owner If you need to support multiple users within the same database at a snapshot site, you may want to create multiple snapshot groups for the target master group. This enables you to define different subqueries for your snapshot definitions in each snapshot group, allowing each user to access only his or her subset of data.

Defining multiple data sets with different snapshot groups is more secure than defining different WHERE clauses for multiple views supporting different users. Since you can grant users access to individual snapshot objects, you can control what the user views, deletes, and inserts; with a WHERE clause in a view, you can only control what a user views, but not the deleting or inserting of data.

Defining multiple snapshot groups gives you the ability to control data sets at a group level. For example, if you create different snapshot groups for the HR, PERSONNEL, and MANUFACTURING departments, you can administer each department as a group (versus individual objects). For example, you can refresh the snapshots as a departmental group or you can drop the objects as a group.

With respect to dropping a department, if you group all data sets into a single snapshot group and the MANUFACTURING department needs to be removed from the data set, you will need to drop and re-create the snapshot with a WHERE clause that does not contain the MANUFACTURING department. In addition to causing you additional work, it could disrupt other departments from accessing their data. Compartmentalizing your data into separate groups allows you to efficiently manage the data since changes to one group will not affect another group.

To accommodate multiple snapshot groups at the same snapshot site that are based on a single master group, you can specify a group owner as an additional identifier when defining your snapshot group.

After you have defined your snapshot group with the addition of a group owner, you add your snapshot objects to the target snapshot group by defining the same group owner. When using a group owner, remember that each snapshot object must have a unique name. If a single snapshot site will have multiple snapshot groups based on the same master group, a snapshot group's object names cannot have the same name as snapshot objects in another snapshot group. To avoid conflicting names, you might want to append the group owner name to the end of your object name. For example, if you have group owners "HR" and "PERSONNEL", you might name the "EMP" snapshot object as "EMP_HR" and "EMP_PERSONNEL," respectively.

Additionally, all snapshot groups that are based on the same master group at a single snapshot site must "point" to the same master site. For example, if the SCOTT_MG snapshot group owned by HR is based on the associated master group at the ORC1.WORLD master site, then the SCOTT_MG snapshot group owned by PERSONNEL must also be based on the associated master group at ORC1.WORLD, assuming that the HR and PERSONNEL owned groups are at the same snapshot site.

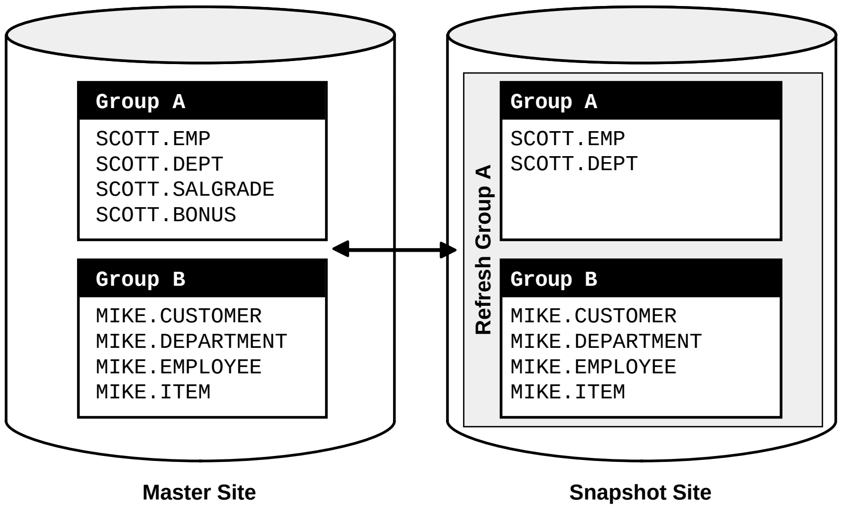
See the "Using a Group Owner" section in Chapter 7 of the *Oracle8i Replication API Reference* manual for more information on defining a group owner using the replication management API.

Refresh Groups

To preserve referential integrity and transactional (read) consistency among multiple snapshots, Oracle has the ability to refresh individual snapshots as part of a refresh group. After refreshing all of the snapshots in a refresh group, the data of all snapshots in the group will correspond to the same transactionally consistent point-in-time.

As illustrated in Figure 3–8, a refresh group can contain snapshots from more than one snapshot group to maintain transactional (read) consistency across master group boundaries.

Figure 3–8 Refresh Groups May Contain Objects from Multiple Snapshot Groups



While you may want to define a single refresh group per snapshot group, it may be more efficient to use one large refresh group that contains objects from multiple snapshot groups (such a configuration reduces the amount of “overhead” needed to refresh your snapshots). A refresh group can contain up to 400 snapshots (the number of snapshots that a refresh group can contain has increased from earlier versions of Oracle Server).

One configuration that you want to avoid is using multiple refresh groups to refresh the contents of a single snapshot group. Using multiple refresh groups to refresh the contents of a single snapshot group may introduce inconsistencies in the snapshot data, which may cause referential integrity problems at the snapshot site. This type of configuration should only be used when you have in-depth knowledge of the database environment and can prevent any referential integrity problems.

Refresh Process

A snapshot’s data does not necessarily match the current data of its master table. A snapshot is a transactionally (read) consistent reflection of its master table as the data existed at a specific point-in-time (i.e. at creation or at a refresh interval). To keep a snapshot's data relatively current with the data of its master table, the snapshot needs to be periodically refreshed. A *snapshot refresh* is an efficient batch operation that makes that snapshot reflect a more current state of its master table.

You must decide how and when to refresh each snapshot to make it more current. For example, snapshots based on master tables that applications update often require frequent refreshes. In contrast, snapshots based on relatively static master tables usually require infrequent refreshes. In summary, you must analyze application characteristics and requirements to determine appropriate snapshot refresh intervals.

To refresh snapshots, Oracle supports several refresh types and methods of initiating a refresh.

Refresh Types

Oracle can refresh a snapshot using either a FAST, COMPLETE, or FORCE refresh.

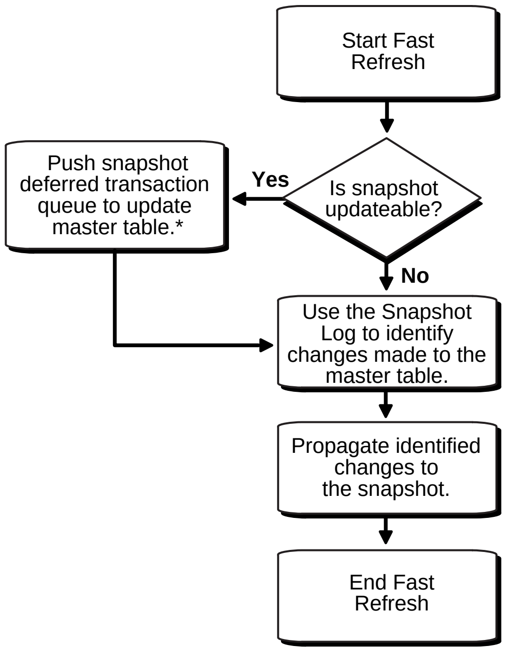
Complete Refreshes To perform a *complete refresh* of a snapshot, the server that manages the snapshot executes the snapshot's defining query. The result set of the query replaces the existing snapshot data to refresh the snapshot. Oracle can perform a complete refresh for any snapshot. Depending on the amount of data that satisfies the defining query, a complete refresh can take a substantially longer amount of time to perform than a fast refresh.

Note: If a snapshot is completely refreshed, set its PCTFREE to 0 and PCTUSED to 100 for maximum efficiency.

Fast Refreshes To perform a *fast refresh*, the server that manages the snapshot first identifies the changes that occurred in the master since the most recent refresh of the snapshot and then applies them to the snapshot. Fast refreshes are more efficient than complete refreshes when there are few changes to the master because the participating server and network replicate less data. Fast refreshes are available for snapshots only when the master table has a snapshot log.

If a fast refresh is not possible, an error is raised and the snapshot(s) will not be refreshed.

Figure 3–9 Fast Refresh of a Snapshot


Start Fast Refresh
Is snapshot updateable?
Yes
Push snapshot deferred transaction queue to update master table.*
No
Use the Snapshot Log to identify changes made to the master table.
Propagate identified changes to the snapshot.
End Fast Refresh

* This step can also be performed separately using the DBMS_DEFER_SYS.PUSH function.

Force Refreshes To perform a *force refresh* of a snapshot, the server that manages the snapshot first tries to perform a fast refresh. If a fast refresh is not possible, then Oracle performs a complete refresh. Use the Force setting when you want the snapshot to refresh if the fast refresh fails.

Initiating a Refresh

When creating a refresh group, administrators may configure the group so that Oracle can automatically refresh its snapshots at scheduled intervals. Conversely, administrators may omit scheduling information so that the refresh group needs to be refreshed manually or “on-demand” (manual refreshing is an ideal solution when refreshing is performed with a dial-up network connection).

Scheduled Refresh When you create a refresh group for scheduled refreshing, you must specify a scheduled refresh interval for the group during the creation process. When setting a group's refresh interval, consider the following characteristics:

- The dates or date expressions specifying the refresh interval must evaluate to a future point in time.
- The refresh interval must be greater than the length of time necessary to perform a refresh.
- Relative date expressions evaluate to a point in time relative to the most recent refresh date. If a network or system failure interferes with a scheduled group refresh, the evaluation of a relative date expression could change accordingly.
- Explicit date expressions evaluate to specific points in time, regardless of the most recent refresh date.
- Consider your environment's tolerance for stale data: if there is a low tolerance, then refresh often; whereas if there is a high tolerance, then refresh less often.

On-demand Refresh Scheduled snapshot refreshes may not always be the appropriate solution for your environment/situation. For example, immediately following a bulk data load into a master table, dependent snapshots will no longer represent the master table's data. Rather than wait for the next scheduled automatic group refreshes, you might want to *manually refresh* dependent snapshot groups to immediately propagate the new rows of the master table to associated snapshots.

You may also want to refresh your snapshots on-demand when your snapshots are integrated with a sales force automation system located on a disconnected laptop. Developers designing the sales force automation software can create an application control (i.e. a button) that a sales person can use to refresh the snapshots when they are ready to transfer the day's orders to the server after establishing a dial-up network connection.

Prepare for Snapshots

Most problems encountered with snapshot replication come from not preparing the environment properly. There are four essential tasks that you must perform before you begin creating your snapshot environment: create the necessary schema, create the necessary database links, assign the appropriate privileges, and allocate sufficient job processes.

Oracle's Replication Manager setup wizard automatically performs the tasks that are described below. The following is provided for your understanding of the replication environment and especially for those that use the Replication Management API. After the setup wizard is executed, you need to make sure to create the necessary snapshot logs (see "Create a Snapshot Log" on page 3-30). See "Setting Up Snapshot Site" on page 5-2 for instructions on using Replication Manager to setup your snapshot site. You are encourage to use Replication Manager whenever possible.

If you are not able to use Replication Manager, review the "Setup Snapshot Site" section in chapter 2 of the *Oracle8i Replication API Reference* for detailed instructions on setting up your snapshot site using the Replication Management API.

The following sections describe what the Replication Manager setup wizard or the script in the *Oracle8i Replication API Reference* will do to setup your snapshot site.

Create Snapshot Site Users

Each snapshot site needs several users to perform the administrative and refreshing activities at the snapshot site. You will need to create and grant the necessary privileges to the snapshot administrator and to the refresher.

Create Master Site Users

You will need equivalent proxy users at the target master site to perform tasks on behalf of the snapshot site users. Usually, a proxy snapshot administrator and a proxy refresher will be created.

Schema

A schema containing a snapshot in a remote database should correspond to the schema that contains the master table in the master database. Therefore, identify the schemas that contain the master tables that you want to replicate with snapshots. Once you have identified the target schemas at the master database, create the corresponding accounts with the same names at the remote database. For example, if all master tables are in the SALES schema of the DB1 database, create a corresponding SALES schema in the snapshot database DB2. (If you are reviewing the steps in *Oracle8i Replication API Reference* manual, the necessary schema(s) are created as part of the script described in chapter 5.)

Database Link

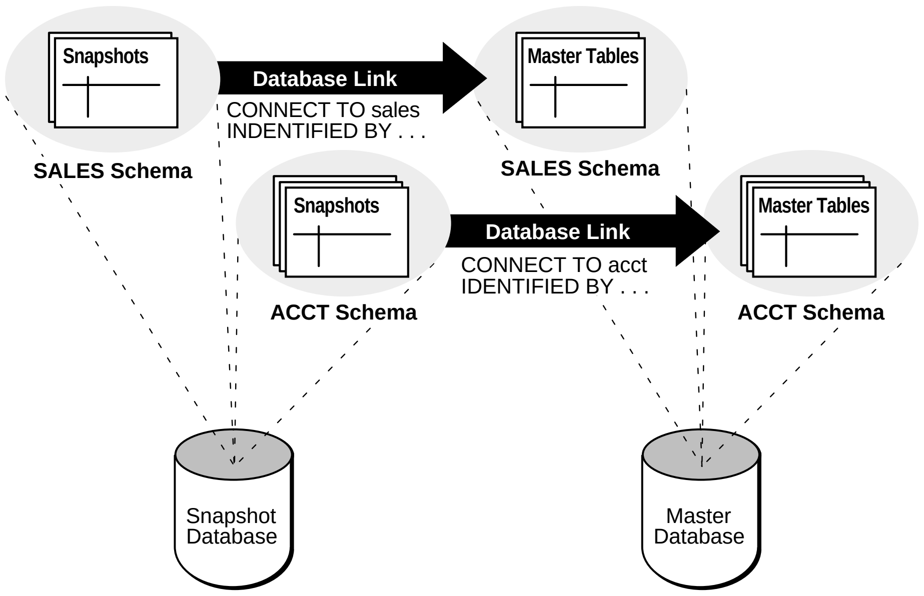
The defining query of a snapshot may use one or more database links to reference remote table data. Before creating snapshots, the database links you plan to use must be available. Furthermore, the account that a database link uses to access a remote database defines the security context under which Oracle creates and subsequently refreshes a snapshot.

To ensure proper behavior, a snapshot's defining query must use a database link that includes an embedded user name and password in its definition; you cannot use a public database link when creating a snapshot. A database link with an embedded name and password always establishes connections to the remote database using the specified account. Additionally, the remote account that the link uses must have the SELECT privileges necessary to access the data referenced in the snapshot's defining query.

Before creating your snapshots, you need to create several administrative database links. Specifically, you should create a PUBLIC database link from the snapshot site to the master site (this makes defining your private database links easier since you don't need to include the USING clause in each link). You will also need private database links from the snapshot administrator to the proxy administrator and from the propagator to the receiver (if you use the Replication Manager Setup Wizard, these database links will be created for you). See "Security Setup for Snapshot Replication" on page 8-24 for more information on snapshot users and database links. Additionally, see Chapter 2 of the *Oracle8i Replication API Reference* manual.

After the administrative database links have been created, a private database link needs to be created connecting each replicated snapshot schema at the snapshot database to the corresponding schema at the master database. Be sure to embed the associated master database account information in each private database link at the snapshot database. For example, the SALES schema at a snapshot database DB2 should have a private database link DB1 that connects using the SALES username and password.

Figure 3–10 Recommended Schema and Database Link Configuration



Privileges

Both the creator and the owner of (schema that contains) the snapshot must be able to issue the defining SELECT statement of the snapshot. If a user other than the replication or snapshot administrator will be creating the snapshot, then that user must have the CREATE snapshot privilege and the appropriate SELECT privileges to execute the defining SELECT statement. (If you are reviewing the steps in *Oracle8i Replication API Reference* manual, the necessary privileges are granted as part of the script described in chapter 5.)

Schedule Purge at Master Site

In order to keep the size of the deferred transaction queue in check, you need to schedule a purge operation to remove all successfully completed deferred transactions from the deferred transaction queue. This operation may have already been performed at the master site; re-scheduling the purge operation will not harm the master site, but may change the purge scheduling characteristics.

Schedule Push

Often referred to as a scheduled link, scheduling a push at the snapshot site will automatically propagate and deferred transactions at the snapshot site to the associated target master site. Typically, there will only be a single scheduled link per snapshot group at a snapshot site (since a snapshot group only has a single target master site).

SNP Background Processes and Interval

It is important that you have allocated sufficient SNP background processes to handle the automatic refreshing of your snapshots. Since your snapshot site will typically have only a single scheduled link to the target master site, the snapshot site will only require a single SNP process, but to handle additional activity, you may want to allocate at least two SNP processes at the snapshot site.

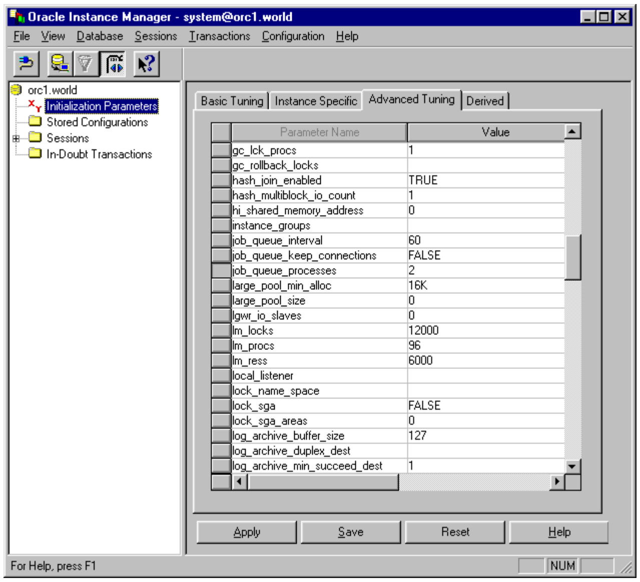
The SNP processes are defined using the `job_queue_processes` parameter in the `init.ora` file for your database. To set your SNP processes, you can either use Instance Manager, a component of Oracle Enterprise Manager, or manually edit the `init.ora` file.

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

Instance Manager

You will often use Instance Manager to configure the SNP processes and interval at the snapshot site if you have a dedicated network link to the snapshot site or you are able to schedule the network link. This is required because Instance Manager will, in most cases, not be at the snapshot site and thus the configuration will need to be done remotely from the master site (if remote configuration is not possible, see the next section).

Figure 3–11 Use Instance Manager to configure the amount of job processes.



Parameter Name	Value
gc_lck_procs	1
gc_rollback_locks	
hash_join_enabled	TRUE
hash_multiblock_io_count	1
hi_shared_memory_address	0
instance_groups	
job_queue_interval	60
job_queue_keep_connections	FALSE
job_queue_processes	2
large_pool_min_alloc	16K
large_pool_size	0
lgwr_io_slaves	0
lm_locks	12000
lm_procs	96
lm_ress	6000
local_listener	
lock_name_space	
lock_sga	FALSE
lock_sga_areas	0
log_archive_buffer_size	127
log_archive_duplex_dest	
log_archive_min_succeed_dest	1

Complete the following to set your job processes using Instance Manager (see the *Oracle Enterprise Manager Administrator's Guide* for more information using Instance Manager to configure your database):

1. Start Instance Manager and connect to the target database.
2. Select **Initialization Parameters** listed below your target database.
3. Press the **Advanced Tuning** tab.
4. Enter `2` in the **Value** field next to the `job_queue_process` **Parameter Name**.
5. If necessary, modify your `job_queue_interval` as necessary (entered values should be in seconds).
6. Press the **Apply** button.

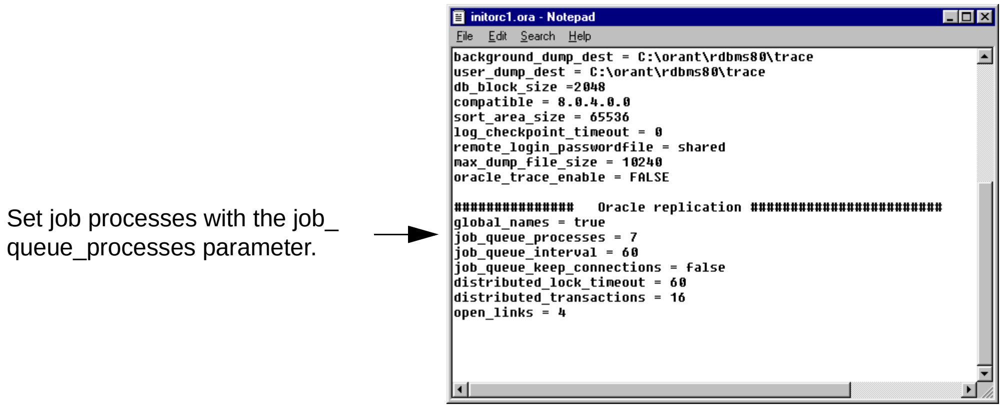
   You will have the opportunity to save this configuration; this is helpful if you use Instance Manager to manage your database. See the *Oracle Enterprise Manager Administrator's Guide* and / or online documentation for more information on using Instance Manager.

Manually Edit INIT.ORA

If you do not have access to Instance Manager, you can manually edit the init.ora file. Use a text editor, such as Notepad, EMACS, or vi (depending on your operating system), to modify the contents of your `init.ora` file.

In most cases, you will see all of the parameters used in replication grouped together under an `Oracle replication` heading in your init.ora file.

Figure 3–12   Use Notepad to edit your init.ora file in a Windows environment.



```
background_dump_dest = C:\orant\rdbms80\trace
user_dump_dest = C:\orant\rdbms80\trace
db_block_size =2048
compatible = 8.0.4.0.0
sort_area_size = 65536
log_checkpoint_timeout = 0
remote_login_passwordfile = shared
max_dump_file_size = 10240
oracle_trace_enable = FALSE

###############   Oracle replication ########################
global_names = true
job_queue_processes = 7
job_queue_interval = 60
job_queue_keep_connections = false
distributed_lock_timeout = 60
distributed_transactions = 16
open_links = 4
```

After you have modified the contents of your init.ora file, you will need to restart your database with these new settings (see *Oracle8i Administrator's Guide* for information on restarting your database).

Create a Snapshot Log

Before creating snapshot groups and snapshots for a remote snapshot site, make sure to create the necessary snapshot logs at the master site. A snapshot log is necessary for every master table that supports at least one snapshot with fast refreshes.

To create a snapshot log at the master site:

1. Select the database in the left pane of the Replication Manager user interface where you want to create the snapshot log.
2. Select **Create** from the **File** menu
3. Select **Snapshot Log** from the sub-menu.

You can optionally press the **Create Snapshot Log** button on the toolbar.

4. Select the schema from the **Schema** pull-down list that contains the table that you want to create a materialized log for.
5. Select the table that you want to create a snapshot log for from the **Table** pull-down list.
6. Select the tablespace that you want to support the snapshot log from the **Tablespace** pull-down list.
7. Select the appropriate **Supported Refresh Types**:
   - **Row ID**: Enable the Row ID checkbox only to support snapshots from an Oracle 7.3 or earlier database.
   - **Primary Key**: By default, Oracle8*i* creates Primary Key snapshot logs.

     If your snapshot log needs to support both Row ID and Primary Key snapshots, be sure that you enable both the **Row ID** and the **Primary Key** checkboxes.
8. If necessary, select any required filter columns from the **Available Columns** list and press the **Add** (right-arrow) button.

   See the following section, “Using Filter Columns" for more information on filter columns.
9. If necessary, press the **Storage** tab to modify the storage settings for your snapshot log.

   Press the **Help** button to see additional information about the available storage settings.
10. Press the **Create** button to complete your snapshot log.

Using Filter Columns

Filter columns are an essential component when using subquery snapshots (see "Data Subsetting with Snapshots" on page 3-7 for more information). A filter column must be defined in a snapshot log (see "Create a Snapshot Log" on page 3-30) that is supporting a snapshot that references a column in a WHERE clause and is not part of the equijoin (see "Restrictions for Snapshots with Subqueries" on page 3-12 for additional information).

Consider the following DDL:

```
1)  CREATE SNAPSHOT sales.orders AS
2)    SELECT * FROM sales.orders@dbs1.acme.com o
3)    WHERE EXISTS
4)      (SELECT c_id FROM sales.customers@dbs1.acme.com c
5)       WHERE o.c_id = c.c_id AND zip = 19555);
```

If you pay close attention to line 5 of the above DDL, you will see that three columns are referenced in the WHERE clause. Columns `o.c_id` and `c.c_id` are referenced as part of the equijoin clause; the column `zip` is an additional filter column. You will therefore need to create a filter column in the snapshot log for the `zip` column of the `sales.customers` table.

You are encouraged to analyze the defining queries of your planned snapshot(s) and identify which filter columns will need to be created in your snapshot log(s). If you try to create or refresh a snapshot that requires a filter column before creating the snapshot log containing the filter column, your snapshot creation or refresh may fail.

Create Snapshot Environment

Snapshot environments can be created in several different ways and from several different locations. In most cases, you will want to use deployment templates to locally pre-create a snapshot environment that will be individually deployed to the target snapshot site.

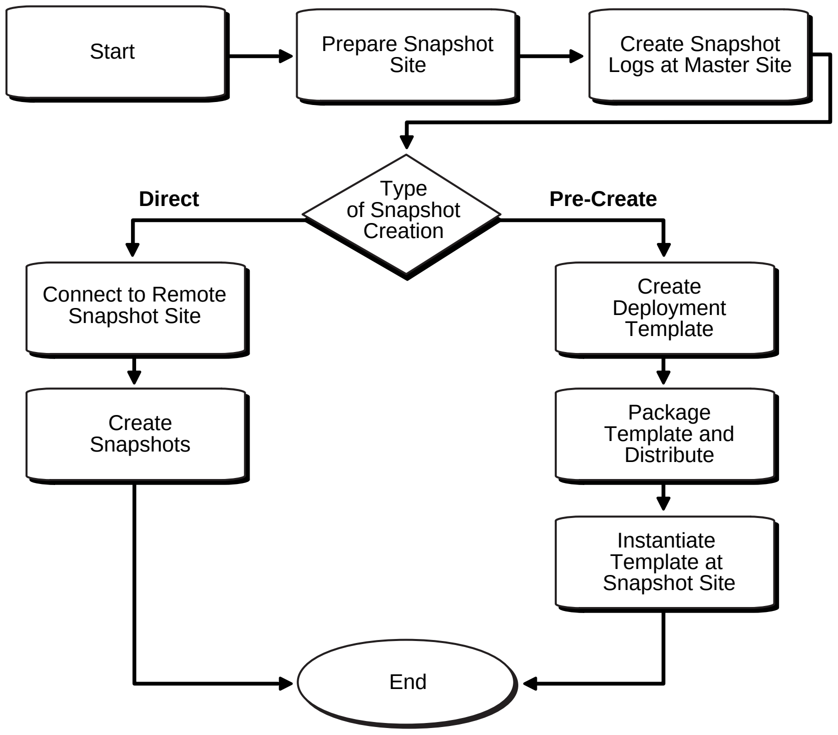
You can also individually create the snapshot environment by establishing a connection to the snapshot site and building the snapshot environment directly.

Replication Manager

See Chapter 4, "Creating Snapshots with Deployment Templates" for information on using deployment templates to centrally create a snapshot environment using Replication Manager.

See Chapter 5, "Directly Create Snapshot Environment" for information on individually creating the snapshot environment with a direct connection to the remote snapshot site using Replication Manager.

Figure 3–13 Creating Snapshot Process


Start
Prepare Snapshot Site
Create Snapshot Logs at Master Site
Type of Snapshot Creation
Direct
Pre-Create
Connect to Remote Snapshot Site
Create Snapshots
Create Deployment Template
Package Template and Distribute
Instantiate Template at Snapshot Site
End

Replication Management API

See Chapter 4 of the *Oracle8i Replication API Reference* manual for information on using deployment templates to centrally pre-create a snapshot environment using the Replication Management API.

See Chapter 5 of the *Oracle8i Replication API Reference* manual for information on individually creating the snapshot environment with a direct connection to the remote snapshot site using the Replication Management API.

4

Creating Snapshots with Deployment Templates

Chapter 4 introduces Deployment Templates and describes how to use them to easily and efficiently distribute snapshot environments. Topics covered in this chapter are:

Note: Be sure that you have read Chapter 3, "Snapshot Concepts & Architecture" before you create a deployment template. Understanding how snapshots are created and work will better prepare you to build deployment templates.

Oracle Deployment Templates provide the enterprise DBA with the tools to efficiently deploy and administer a widely distributed snapshot environment. Before learning about the concepts, architecture, or use of deployment templates, examine the challenges of a mass deployment environment.

Mass Deployment Challenge

The need to have accurate information at any time and at any place continues to grow rapidly. At the same time, information usage is becoming decentralized and often disconnected, requiring the information to be distributed to the active points-of-usage.

Consider the mobile sales force. Potentially hundreds, if not thousands, of professionals need accurate information about *their* customers on a laptop in a manner that causes the salesperson very little inconvenience. The challenge, therefore, is for the database administrator to rollout the data and the database infrastructure (tables, indexes, constraints, triggers, etc.) to all sites in an efficient and timely manner.

Traditionally, the DBA has been required to develop a deployment method of their own. This usually means that the DBA was responsible for developing a very complex script to create the snapshot environment at the remote snapshot site. In addition to building the script, the DBA was often forced to figure out a method for parameter substitution to allow custom data sets at the snapshot site. Once the DBA completed engineering the script, deploying the script required manual packaging and implementation, both of which often required extensive troubleshooting.

The problems encountered in the above scenario have spawned technologies and resources dedicated to the art of efficient *mass deployment*.

Mass deployment is the term used to describe the process of distributing database infrastructure, data, and front-end applications to a large number of users. For the purposes of Oracle replication, the discussion of mass deployment will be limited to the delivery of data and data infrastructure.

Deployment Goals

Mass deployment tools and technologies should aid the database administrator in delivering the data and database infrastructure to the target users such that:

- Each user receives database infrastructure to support the front-end application
- Each user (site) receives the particular data subset needed
- The DBA can centrally deploy hundreds or thousands of remote sites

Mass deployment has many applications, such as distributing information to mobile sales forces, field technicians, retail stores, remote inventory collection sites, etc. Such environments use snapshots to build the database infrastructure at the remote site, largely because of their support for data subsetting, disconnected replication, and their lower resource requirements (ideal for laptops users).

Oracle Deployment Templates Concepts

Oracle offers deployment templates to allow the database administrator to package a snapshot environment for easy, custom, and secure deployment. A deployment template can be as simple as containing a single snapshot with a fixed data set, or as complex as hundreds of snapshots with a dynamic data set based on one or more variables. The goal is to define the environment once and create as many instances of the deployment template as necessary. Deployment templates feature:

- Centralized control
- Capability to repeatedly deploy a snapshot environment
- Template parameters that allow data subsetting or customization at remote site
- Authorized user lists to control template instantiation and data access

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

Deployment Template Elements

Each Deployment Template contains the “blueprint” for creating the necessary snapshots and related objects at a snapshot site. Specifically, the DBA creates the Deployment Template at the master site, adding the necessary snapshots, triggers, views, etc. to the template as needed to create the snapshot environment. The DBA can optionally define template parameters and authorized users, giving the template greater flexibility and security during the instantiation process. Deployment template elements can be divided into four categories.

- General Template Information
- Object Definitions
- Template Parameters
- User Authorization

General Template Information

Oracle deployment templates center around the general template information, which consists of the template name, target refresh group, and private/public status. As illustrated in Figure 4–1, the `refresh_template_name` is used in all aspects of deployment template definitions. The DBA adds the snapshot environment objects to the template prior to releasing the template for distribution according to the specified template identification (see Figure 4–2).

A deployment template is defined at a single master site. While you cannot have two deployment templates at the master site with the same name, you could copy a deployment template to another site with the same deployment template name.

Figure 4–1 Refresh Group Template View Relationships

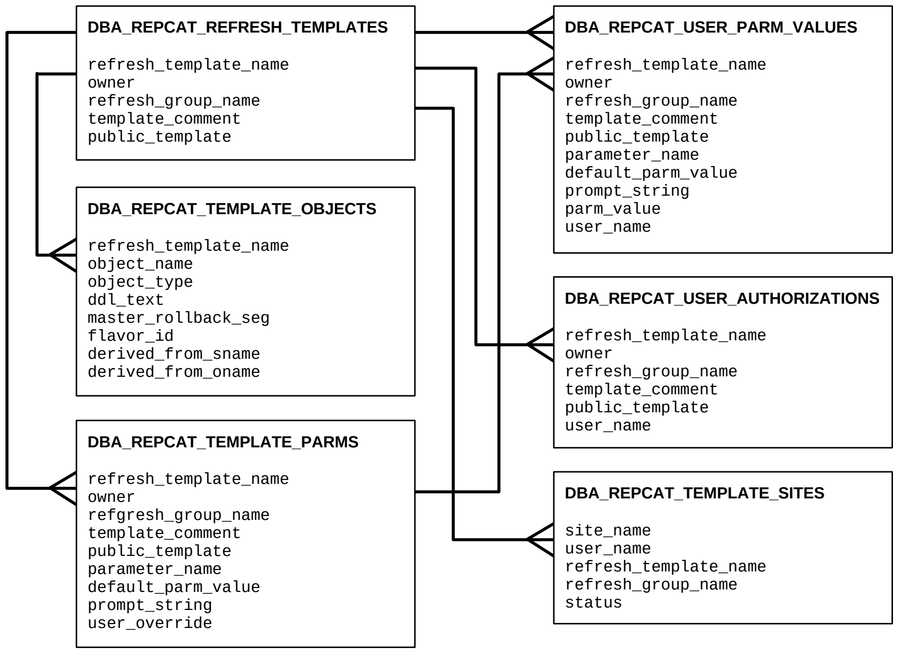
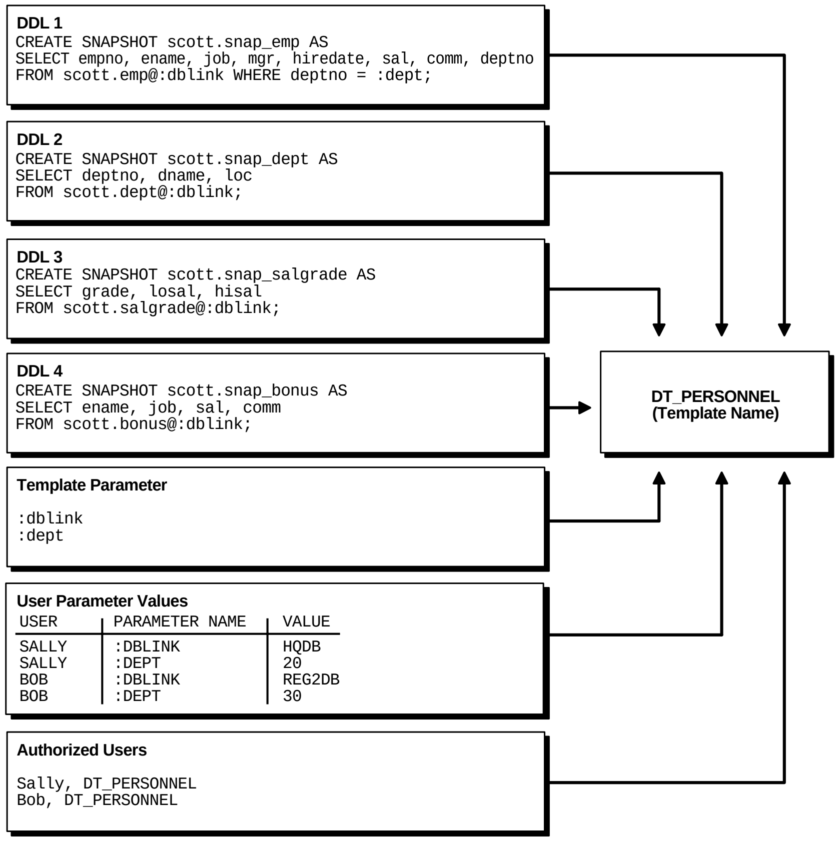


Figure 4–2 Deployment template elements are added to template.


DDL 1
CREATE SNAPSHOT scott.snap_emp AS
SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
FROM scott.emp@:dblink WHERE deptno = :dept;
DDL 2
CREATE SNAPSHOT scott.snap_dept AS
SELECT deptno, dname, loc
FROM scott.dept@:dblink;
DDL 3
CREATE SNAPSHOT scott.snap_salgrade AS
SELECT grade, losal, hisal
FROM scott.salgrade@:dblink;
DDL 4
CREATE SNAPSHOT scott.snap_bonus AS
SELECT ename, job, sal, comm
FROM scott.bonus@:dblink;
Template Parameter
:dblink
:dept
User Parameter Values
USER | PARAMETER NAME | VALUE
SALLY | :DBLINK | HQDB
SALLY | :DEPT | 20
BOB | :DBLINK | REG2DB
BOB | :DEPT | 30
Authorized Users
Sally, DT_PERSONNEL
Bob, DT_PERSONNEL
DT_PERSONNEL
(Template Name)

Object Definitions

After the template has been defined, the DBA adds the required objects to the template. When the template is instantiated at the snapshot site, the object DDL (e.g. CREATE SNAPSHOT... or CREATE TABLE...) will be executed to create the appropriate objects at the snapshot site.

Objects added to a deployment template can be created based on an existing *master* object (object contained in a master group), but if necessary, the DBA can create a new template object by defining new object DDL. Oracle will check any new object DDL to make sure that it is lexically correct, which prevents executing faulty DDL.

In most cases, the DBA will be adding snapshots to the template, but if necessary, the DBA can add other objects. For example, constraints can be added to enforce data integrity at the snapshot site, views can be added for displaying data, or tables can be added for local data storage. In some cases, the DBA may even want to include all objects for an application in a deployment template. Snapshots created using a deployment template will automatically be added to the refresh group defined for the template (see "General Template Information" on page 4-4 for more information).

Also, if the target snapshot sites will require data sets unique to their site, the DBA has the option of defining variables in the object DDL. These variables will create a parameterized template that allows for custom data sets when the same template is instantiated, allowing different snapshot sites to have different data sets.

Template Parameters

Since the DBA has the ability to define variables in the template object DDL, Oracle allows the DBA to specify default values, prompt text, and user-specific parameter values for a template.

User-specific parameter values are optional and if the DBA has not defined any, the user can specify runtime parameter values to be used during an instantiation. If the user is using the client instantiation tool, the user will see the prompt text specified by the DBA and can enter a value for a specified prompt or accept the default value, also specified by the DBA (if the API will be used, the user will not automatically see the prompt text and default values).

Note: Parameter values must be pre-defined by a DBA when performing an online instantiation from a site using Oracle Lite.

If user-specific parameter values do exist for either public or private templates, then these values will automatically be used when the specified user instantiates the template. For example, consider the variable `REGION`. If the DBA establishes the following user-specific parameter values for template `X`,

USER	REGION
SCOTT	EAST
LARRY	WEST

and users SCOTT and LARRY instantiate template X, their resulting snapshot data sets will be:

User SCOTT	
CUST ID	**REGION**
A123	EAST
A234	EAST
A345	EAST
A456	EAST

User LARRY	
CUST ID	**REGION**
B123	WEST
B234	WEST
B345	WEST
B456	WEST

User Authorization

Deployment templates can be either public or private (defined when the template is created). If a template is public, any user with access to the master site can instantiate the template.

If a template has been created for private use, only authorized users ar able to instantiate the target template. To enforce private use, the DBA creates a list of authorized users at the master site and if an unauthorized user attempts to instantiate the target template, the instantiation process fails.

Template Parameters and Private Templates Private templates that have user specific parameters defined securely limit the snapshot site to viewing and changing only the data that satisfies the WHERE clause of the snapshot.

For example, the DBA has added user SCOTT to the user authorization list, has specified the following for the REGION parameter in the user specific parameters list

USER	REGION
SCOTT	EAST
LARRY	WEST

and the defining SELECT statement for the snapshot is:

```
SELECT cust_id, sales_to_date, status FROM table_x WHERE region_id=:region;
```

Users accessing the snapshot instantiated by user SCOTT will only see data for region EAST and will only be able to view, update, or delete data that complies with this WHERE clause. In other words, a user of this snapshot will not be able to view, update, or delete data for region WEST.

Deployment Sites

Maintaining the emphasis on centralized control, the DBA can monitor and manage certain characteristics of the instantiated environment at the remote snapshot site. Specifically, the DBA has the ability to view the sites that have instantiated a deployment template (which includes the deployment template name, authorized user, and status of the instantiated environment).

Package Deployment Template

When you have completed defining your deployment template, you need to prepare the template for instantiation at the remote snapshot site. As you will learn in the next section, remote snapshot sites can perform either online or offline instantiations. Regardless of the method used, you need to package your deployment template for the appropriate instantiation method.

Either the DBA or the target user can package the deployment template. There is a public and private version of the packaging API (public is meant for users and private is meant for DBAs). In many cases, the DBA will perform the offline packaging on behalf of a user, while the user may perform their own packaging for an online instantiation, though there no restrictions on users or DBAs performing either type of instantiation.

Note: When you package a deployment template for offline instantiation, the related snapshot logs begin logging for the snapshots that were packaged in the template. This immediate logging enables the remote snapshot site to perform a fast refresh after completing the offline instantiation process. You will want to monitor the snapshot logs to make sure that remote snapshot sites refresh in a timely manner after performing an offline instantiation (remote snapshot sites that have not refreshed cause the snapshot log to grow quite large, since logging begins when the template is packaged).

Deployment Template Instantiation

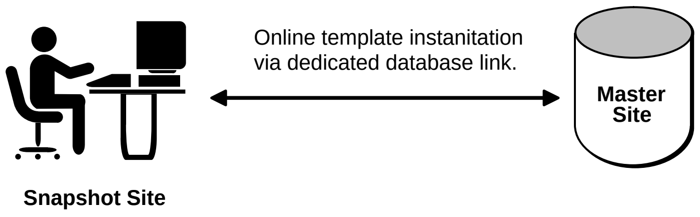
After a deployment template has been *packaged* for deployment, the snapshot environment can be created via online or offline instantiation.

The target instantiation site will be required to run one of several Oracle database clients: Oracle8*i* Server, Oracle8*i* Server Enterprise Edition, Personal Oracle8*i* or Oracle8*i* Lite. The typical mass deployment scenario will have Oracle8*i* Server Enterprise Edition at the master site and Oracle8*i* Lite (which has a footprint between 350K and 750K) at the remote site, which will often be a laptop.

Online Instantiation

Online instantiation allows a snapshot site to instantiate a deployment template while connected to the target master site. When the snapshot site connects to the master site, a list of available templates is displayed (if you are using the API, you must supply the template name as a parameter value). During the instantiation process, the specified data subset is pulled from the master site and stored in the appropriate snapshot base tables.

Figure 4–3 Online Instantiation



One of the benefits of online instantiation is that the data subset will be current as of the instantiation process. This data currency, however, comes at a cost. Online instantiation requires a “live” connection between the snapshot and master sites, which (depending on the size of the snapshot environment created) may cause extreme network traffic, possibly degrading other network services. Furthermore, laptop users connected by a modem may be required to stay connected for a long duration, depending on the number of objects created and the amount of data retrieved from the master site.

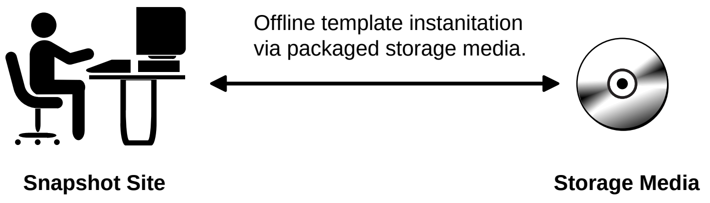
An online instantiation does not require the DBA to create the setup files in advance, create distribution media (e.g. CD-ROM), or physically distribute the offline instantiation media.

Offline Instantiation

To decrease server loads during peak usage periods and reduce remote connection times, you may choose to instantiate the template offline. To instantiate a template offline, the DBA packages the template and required data on to some type of storage media (tape, CD-ROM, etc.).

Instead of pulling the template and data from the master site, they are pulled from the storage media containing the template and data. While this has the benefit of reducing network traffic and the eliminating the need for a "live" connection, the data is not current until after the first fast refresh process.

Figure 4–4 Offline Instantiation



Offline instantiation is an ideal solution for mass deployment situations where many laptops and other disconnected computers will be instantiating the target template.

Deployment Template Architecture

Oracle uses standard snapshot architecture with Deployment Templates to distribute snapshot environments quickly and effectively. Deployment Templates use the same methods in creating snapshot definitions, refresh characteristics, conflict resolution, and grouping as used when manually building a snapshot environment. The distinction to remember is that instead of executing the DDL to create the object immediately, the object DDL is simply contained in a deployment template and will be executed when the template is instantiated.

Template Definitions Stored in System Tables

Instead of executing DDL at the snapshot site to immediately create a snapshot environment, the snapshot and other related object definitions are stored within the deployment template. After all of the object definitions have been added to the deployment template, the template can be instantiated and all of the stored DDL will be executed at the remote snapshot site to create the necessary snapshot environment.

All of these object definitions are stored in system tables maintained at the deployment template definition site, keyed on the deployment template name. When the deployment template is instantiated, the stored object DDL is pulled from these system tables to create the snapshot environment.

Use Standard DDL

Template object definitions are created using the same DDL that is used to create the objects locally at the snapshot site. For example, you would execute the following to create a snapshot locally:

```
CREATE SNAPSHOT foo_snap AS SELECT empid, region, dept, salary
   FROM scott.foo@hq.com;
```

To add this same snapshot to a Deployment Template, you would use the Deployment Template Wizard or execute the CREATE_TEMPLATE_OBJECT function:

```
DECLARE
a NUMBER;
BEGIN
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT(
      refresh_template_name => 'dt_snapenv',
      object_name => 'foo_snap',
      object_type => 'snapshot',
      ddl_text => 'CREATE SNAPSHOT foo_snap AS SELECT empid, region, dept, salary
                   FROM scott.foo@hq.com');
END;
```

Executing the above function adds the snapshot definition to the deployment template named DT_SNAPENV. When this particular snapshot is instantiated, the snapshot FOO_SNAP will be created. In addition to creating snapshots, you can add table, trigger, procedure, index, and other object definitions to the deployment template (see "DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT" in the *Oracle8i Replication API Reference* for usage information).

Packaging Process

When you package your deployment template in preparation for remote snapshot site instantiation, Oracle creates a script that is stored in a temporary table that contains all of the DDL and, in the case of offline instantiation, the DML (Data Manipulation Language) to create the snapshot environment.

Online Instantiation

When you package a template for online instantiation, the DDL that is required to create the remote snapshot environment is stored in the temporary view USER_REPCAT_TEMP_OUTPUT.

During an online instantiation, the remote snapshot site “pulls” the script directly from the master site. The script creates the snapshot environment using the stored DDL and the snapshots are populated with the data while online.

All template parameters are evaluated when the remote snapshot site begins the template instantiation procedure.

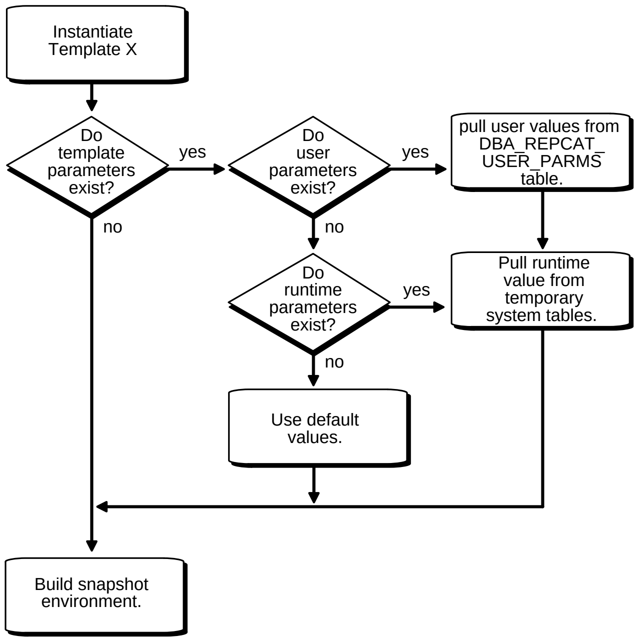
Offline Instantiation

When a deployment template is packaged for offline instantiation, any template parameters are evaluated at the time of packaging. Since the parameters are evaluated before the actual instantiation at the remote snapshot site, any runtime parameter values must be defined before packaging the template.

Note: When the user executes the offline instantiation script at the remote snapshot site, the user will not be able to define any runtime parameters.

When you package a template for offline instantiation, the DDL that is required to create the remote snapshot environment and the DML to populate the environment with the "snapshot" of data is stored in the temporary view USER_REPCAT_TEMP_OUTPUT.

Figure 4–5 Checking for parameters during offline packaging process.


Instantiate Template X
Do template parameters exist?
yes
no
Do user parameters exist?
yes
no
pull user values from DBA_REPCAT_ USER_PARMS table.
Do runtime parameters exist?
yes
no
Pull runtime value from temporary system tables.
Use default values.
Build snapshot environment.

The script that is created in the temporary table for offline instantiation is copied to a storage device using Replication Manager (e.g. hard disk, CD-ROM, tape, etc.). When the remote snapshot site instantiates the template, the script is executed from the storage media. This script creates the snapshot environment and uses DML to populate the environment according to the data set defined during the packaging process (recall that the data set according to the template parameters is defined during the packaging process).

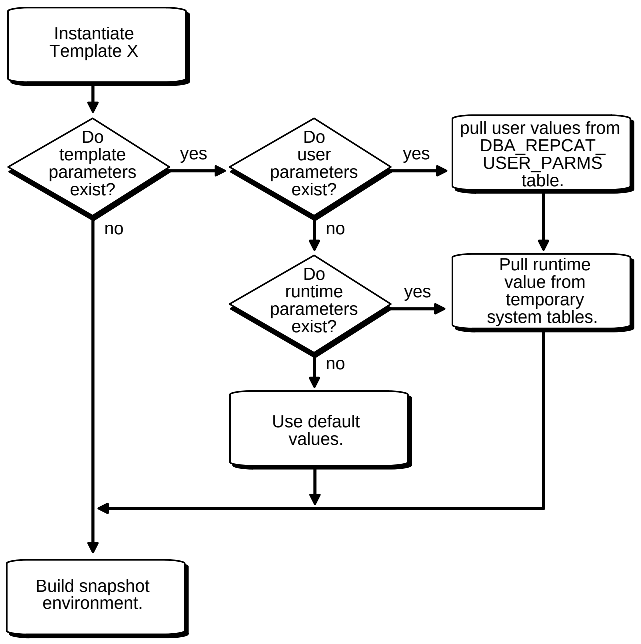
Instantiation Process

Once the remote snapshot site begins the instantiation procedure (either online or offline), Oracle immediately executes the script that was created during the packaging process. This script will create the remote snapshot environment and populate the environment as specified. The following sections will discuss online and offline instantiation in greater detail.

Online Instantiation

Once the remote snapshot site begins the online instantiation process, Oracle evaluates the parameters that have been defined for the deployment template. Any values defined for these parameters will be used when the object DDL in the template is executed so that custom data sets may be installed at the remote snapshot site.

Figure 4–6 Checking for parameters during online instantiation process.


Instantiate Template X
Do template parameters exist?
yes
no
Do user parameters exist?
yes
no
pull user values from DBA_REPCAT_ USER_PARMS table.
Do runtime parameters exist?
yes
no
Pull runtime value from temporary system tables.
Use default values.
Build snapshot environment.

There are several possible methods that can be used to define template parameter values: default parameter values, runtime parameter values, or user parameter values. Oracle checks to see if these parameter values exist and uses them according to the hierarchy:

1. User Parameter Values
2. Runtime Parameter Values
3. Default Parameter Values

If user parameter values have been defined and a listed user is instantiating the template, the user parameter values will be used when instantiating the template.

If user parameter values do not exist, Oracle will check to see if any runtime parameter values have been defined. If runtime parameter values have been defined, they will be used during the instantiation process.

Note: Parameter values must be pre-defined by a DBA when performing an online instantiation from a site using Oracle Lite.

If no user parameter values or runtime parameter values have been defined, Oracle will use the default parameter values.

Once any deployment template parameter values have been resolved, the object DDL is generated and stored in the temporary USER_REPCAT_TEMP_OUTPUT view. Remember the distinction between an online and offline instantiation; with an online instantiation, the contents of the USER_REPCAT_TEMP_OUTPUT view will only contain the DDL to create the snapshot environment (vs. DDL and DML for an offline instantiation). When this DDL is executed at the remote client site, there must be a connection to the master site in order to build the snapshot environment and populate it with the appropriate data.

The contents of this view can be saved to a file for distribution, or if the Oracle Client Instantiation Tool or similar tool is used at the remote site, the contents of this view will be retrieved and executed at the remote client site.

The objects created by the template are then added to the refresh group specified when the template was created.

Offline Instantiation

In a mass deployment environment, most snapshot environments will use the offline instantiation method to create the necessary snapshot environment. When the DBA prepares the deployment template for deployment, an image is created that stores the DDL needed to create the snapshot environment, the parameter values used during the instantiation process, and the data needed to populate the snapshot environment.

The image can be copied to a CD-ROM, floppy disk, or other storage media or can be posted on a Web or FTP site to be downloaded to the remote snapshot site. The flexibility in delivery mechanisms allows you and your users to choose the most effective method for instantiating a deployment template.

Once any deployment template parameter values have been resolved, the object DDL to create the environment and the DML to populate the environment is generated and is stored in the temporary USER_REPCAT_TEMP_OUTPUT view (again, online instantiation generates only the DDL whereas offline generates both DDL and DML to create and populate the environment).

With respect to the data, the longer the duration between DDL and DML generation and instantiation at the remote site, the longer it will take for the first refresh after instantiation at the remote snapshot site (the snapshot site will use the snapshot log at the master site to perform the fast refresh from the time that the template was packaged). Recall that changes made to the master table will be logged to the snapshot log as soon as you package the deployment template.

Post-Instantiation

After instantiating a deployment template at a remote snapshot site, the structure created is exactly the same as if you had created the snapshot environment locally at the snapshot site. Specifically, Oracle creates the snapshot (with the specified name) and an index based on the primary key to maintain constraint consistency. Other objects in the template are also created as if they were created manually at the snapshot site. (See "Snapshot Architecture" on page 3-14 for more information.)

Snapshot Groups

Objects created by an instantiated deployment template are added to a snapshot group with a name derived from the object's master group. For example, if you instantiated the DT_SNAPENV deployment template which contains objects from the PERSONNEL and TECHNICAL master groups, your template objects will be added to snapshot groups PERSONNEL01 and TECHNICAL01, respectively (the numbered suffix may change if multiple templates contain objects from the same master group). Remember that a snapshot group helps to maintain organizational consistency with the target master group and, more importantly, is required for updateable snapshots. See "Snapshot Groups" on page 3-18 for more information.

Refresh Groups

When you first begin building a deployment template, you define the name of the refresh group to which the template's snapshot objects will be added. Also, when you execute the instantiation procedure, you have the option of specifying values for the `next_refresh` and `refresh_interval` parameters (disconnected laptop users will not specify values since they will refresh their snapshots on demand).

After instantiation has been completed, the template objects will have been added to the specified refresh group and will have the refresh characteristics given at the time of instantiation. The ability to define refresh characteristics at the time of instantiation allows each site to specify refresh requirements specific to their site.

Note: The name of the refresh group must be the same as the name of the deployment template when building deployment templates to be instantiated at an Oracle Lite site.

Deployment Template Design

Before you begin assembling your deployment template, you should take some time to critically think about how to build your templates.

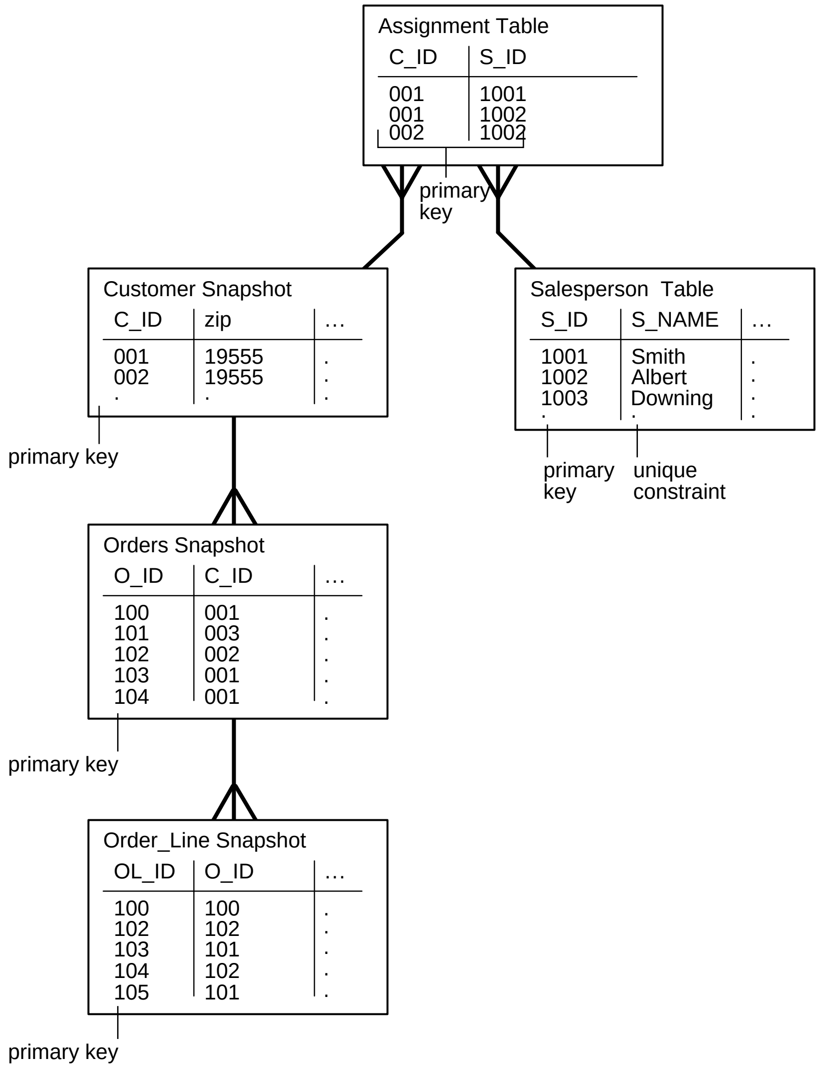
The combination of deployment templates and subquery subsetting (see "Data Subsetting with Snapshots" on page 3-7 for more information) gives the database administrator greater flexibility and power to administer a widely distributed database environment using *assignment tables* and horizontally partitioned data.

Additional design consideration needs to be given to vertical partitioning requirements and data sets needed for a replicated environment (see "Vertical Partitioning" on page 4-20 for more information).

Horizontal Partitioning with Assignment Tables

As discussed in the previous section, snapshot data sets are defined based on the snapshot's query, meaning that the user only sees data that complies with the snapshot's defining query.

Figure 4–7 Customer/Salesperson Relationship



With this in mind, if “assignment” tables are used in conjunction with subquery subsetting, changes to the snapshot environment can easily be controlled by the DBA (for additional information about horizontal partitioning and using assignment tables, see "Data Subsetting with Snapshots" on page 3-7). For example, consider the customer / salesperson relationship in Figure 4–7.

In this example, a salesperson is assigned his / her customers based on the Assignment table. If new salespersons are hired or other salespersons leave, the existing customers can be assigned to their new salesperson by simply modifying the contents of the assignment table. Besides creating a single point of administration, assignment tables used in conjunction with subquery subsetting makes this easy administration remain secure. For example, salesman #1001 will not be able to view the customer information of other salesmen (very important if the customer information contains sensitive data).

Considering the relationships pictured in Figure 4–7, if the Orders snapshot's defining query was specified as (pay special attention to the `:salesperson_id` variable in the last line of the CREATE SNAPSHOT statement):

```
CREATE SNAPSHOT sales.orders AS
 SELECT * FROM sales.orders@hq.acme.com o
    -- conditions for customers
  WHERE EXISTS
  ( SELECT c_id FROM sales.customers@hq.acme.com c
    WHERE o.c_id = c.c_id
      AND EXISTS
    ( SELECT * FROM sales.assignments@hq.acme.com a
     WHERE a.c_id = c.c_id
     AND EXISTS
     ( SELECT * FROM sales.salespersons@hq.acme.com s
       WHERE s.s_id = ':salespercon_id')));
```

then the Orders Snapshot will be populated with order data for the customers that are assigned to the salesperson specified for the `:salesperson_id` variable.

With this flexibility, managers can easily control snapshot data sets by making simple changes to the assignment table (without requiring a DBA to modify any SQL). For example, if the specified salesperson was assigned two new customers, the manager would simply assign these two new customers to the salesperson in the assignment table. After the next snapshot refresh, the data for these two customers will be propagated to the target snapshot site, such as the salesperson's laptop. Conversely, if a customer was taken away from the specified salesperson, all data pertaining to the specified customer will be removed from the snapshot site after the next refresh and the salesperson will no longer be able to access that information.

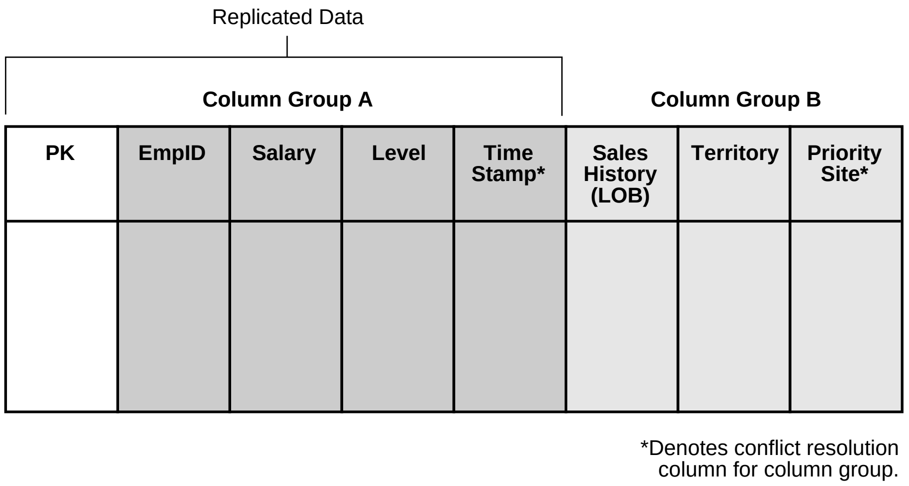
Vertical Partitioning

Deployment templates offer the additional advantage of allowing you to build vertically partitioned updateable snapshots. For example, in a mass deployment environment with many “lightweight” clients, you may need to replicate tables that contain LOB data without actually replicating the LOB data itself. This can be achieved by excluding the LOB column from the selected columns to be replicated when defining the vertical partition.

The vertically partitioned snapshot that you add to your deployment template must contain the following:

- Primary Key
- All columns used for conflict resolution for the replicated columns (see Figure 4–8)

Figure 4–8 Replicate Vertically Partitioned Data



If you are adding a snapshot object that replicates columns PK, EmpID, Salary, and Level (illustrated in Figure 4–8), you need to also include the Time Stamp column since it is used for conflict resolution for columns contained in Column Group A.

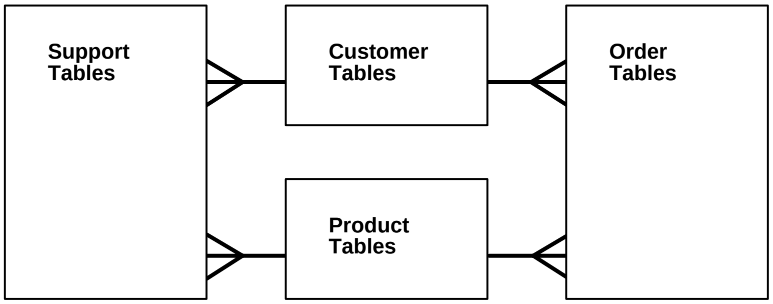
Note: Vertical partitioning is only available when you add a snapshot object to a deployment template using Oracle Replication Manager (vertical partitioning is not available when using the Replication Management API).

Note: The master definition site must be available when defining a vertical partition. If your deployment template contains vertically partitioned snapshots from multiple master groups, the master definition site for each group must be available.

Data Sets

When designing your deployment templates, you need to consider the different sets of users that need to access the target data. For example, salesmen and technicians both need customer information, but they both don't need sales information.

Figure 4–9 Some users require all data.



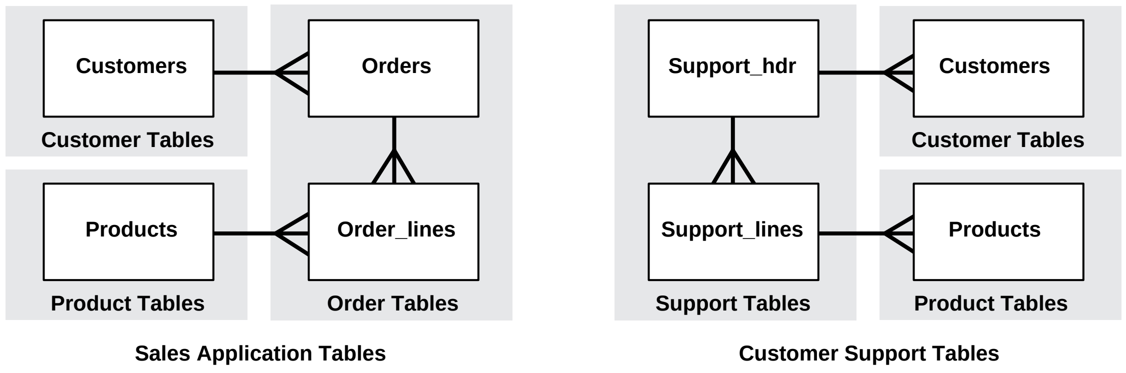
You don't want users to instantiate deployment templates that may contain extraneous data, as it will only require extra storage space and additional refresh times.

On the other hand, if you have users that will require both sales and customer support information, you don't want users to have to instantiate multiple deployment templates that may share redundant data. Instantiating multiple templates may cause data consistency problems; each deployment template uses a different refresh group, which means that data in the two deployment template may be refreshed at different times, possibly causing data consistency problems.

In this case, the best solution would be to have one deployment template for salesmen, one for customer service technicians, and one for users that require both sets of data.

To save time and DBA efforts, the best way to create the above three templates is to create the template with both sets of data first, copy the template twice, deleting unneeded items to create the other deployment templates.

Figure 4–10 Salesmen and customer support technicians have different needs.



Another design consideration you should take into account is the usage of parameters. If many of the tables above use the Customer ID field, you could define the same parameter in each of the template objects. By using the same parameter, you would only need to define the default parameter value and prompt string once, and it would be used for all objects during the instantiation process. Also, when you use the same parameter when defining multiple objects, you will be able to significantly reduce the amount of user defined values that need to be specified, reducing both deployment template development and maintenance times.

Using a single template parameter is even more useful when used with snapshots that use subquery subsetting. One parameter would allow a user to receive only the data for the customers that they need. Consider the following CREATE SNAPSHOT statements:

```
CREATE SNAPSHOT sales.orders AS
 SELECT * FROM sales.orders@hq.acme.com o
    -- conditions for customers
  WHERE EXISTS
  ( SELECT c_id FROM sales.customers@hq.acme.com c
    WHERE o.c_id = c.c_id
      AND EXISTS
    ( SELECT * FROM sales.assignments@hq.acme.com a
     WHERE a.c_id = c.c_id
     AND EXISTS
     ( SELECT * FROM sales.salespersons@hq.acme.com s
       WHERE s.s_id = ':salesperson_id')));


CREATE SNAPSHOT sales.customers AS
SELECT c_id FROM sales.customers@hq.acme.com c
  -- conditions for customers
  WHERE EXISTS
   ( SELECT * FROM sales.assignments@hq.acme.com a
     WHERE a.c_id = c.c_id
     AND EXISTS
     ( SELECT * FROM sales.salespersons@hq.acme.com s
       WHERE s.s_id = ':salesperson_id')));
```

Even though the two snapshots being created don't explicitly contain the SALESPERSON_ID field, using subquery subsetting makes using parameters very effective for instantiating only required data sets. Using a single parameter (:salesperson_id) makes managing and instantiating these snapshots easier for both the DBA and the user instantiating the deployment template.

Additional Design Considerations

Finally, you should consider what other objects need to be created at the remote snapshot site. Ask yourself the following questions:

- Do you need to include the DDL to create the necessary database links from the snapshot site to the master site?
- What triggers or procedures will the snapshot environment require?
- Do any tables need to be created that will store non-replicated data?
- Are any extra indexes required?

Creating a Deployment Template

This section will teach you how to use Replication Manager to build a deployment template. For more information about using the Replication Management API, please see *Oracle8i Replication API Reference*.

Before you begin the following procedures, please make sure that you have completed the following:

- Read chapter 3 before you create a deployment template. Understanding how snapshots are created and work will better prepare you to build deployment templates.
- Start Replication Manager and connect to the master site where you want to create the deployment template.
- Create all of the snapshot logs necessary to support fast refreshing of snapshots.

Note: Vertically partitioned snapshots are available only when using the Oracle Replication Manager Deployment Template Wizard.

Overview

The Deployment Template wizard follows the approach listed below to construct a deployment template.

- Create Template
- Add Objects to Template
- Create User Authorization List
- Create User Parameters for Template

Create Template

The Replication Manager Deployment Template wizard allows you to quickly and intuitively build a deployment template. The following steps guide you in starting the wizard and defining the general information for your deployment template:

1. Select **Create** from the **File** menu.
2. Select **Deployment Template** from the **Create** sub-menu.

3. Select your user name from the **Owner** pull-down list.

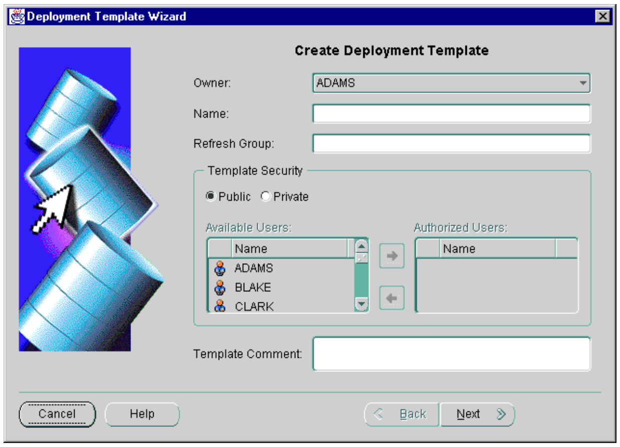
   If you will not be considered the owner of this deployment template, select the appropriate user name from the **Owner** pull-down list.

4. Enter the name of your deployment template in the **Name** field and press the <TAB> key.

5. If necessary, change the name of the refresh group to which you want your template objects added in the **Refresh Group** field and press <TAB>.

   A refresh group with the same name as your deployment template is automatically entered in the **Refresh Group** field.

Figure 4–11 Template Properties and Security Information


Deployment Template Wizard

Create Deployment Template

Owner: ADAMS

Name:

Refresh Group:

Template Security

[✓ Public] Private

Available Users: Name
ADAMS
BLAKE
CLARK

Authorized Users: Name

Template Comment:

6. Select the **Template Security** mode of your new template by enabling the appropriate radio-button:

   - **Public**: Selected template will be available for public usage.
   - **Private**: Selected template will be available to only authorized users.

   If you selected **Public**, skip to step 8.

7. If you enabled the **Private** radio-button in step 6 above, you need to authorize users to instantiate this deployment template. Complete the following to add authorized users:

a. Select the users in the **Available Users** list that you want to authorize; press the <SHIFT> key to select a range of users or press the <CTRL> key to individually select multiple users.

b. Press the **Add** button to authorize the selected users to instantiate this deployment template (you will see these users appear in the **Authorized Users** list).

8. Press the **Next** button.

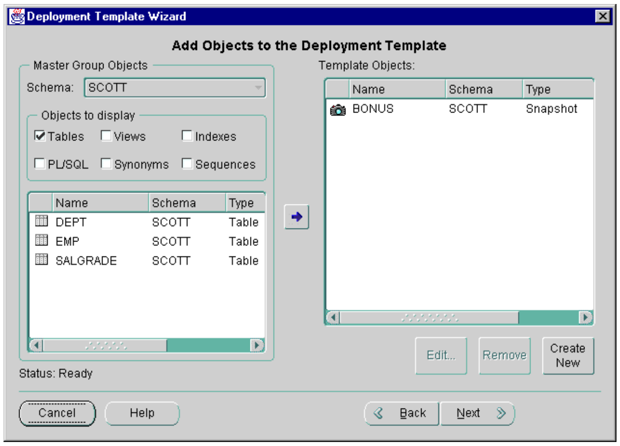
Add Template Objects

The second page of the Deployment Templates wizard allows you to add objects to your template based on existing replicated objects. For example, if you have a PERSONNEL master group, you can add objects from this existing master group to your deployment template (Oracle will create snapshots when you add a replicated table).

1. Select the schema from the **Schema** pull-down list that contains the replicated object that you want to add to your deployment template.
2. Enable the checkbox(es) that corresponds with the type(s) of objects that you want to add.
3. Select the master group object(s) in the **Available Objects** list that you want to add to your deployment template

   Press the <SHIFT> key to select a range of objects or press the <CTRL> key to individually select multiple objects.

Figure 4–12   Add Replicated Object to Deployment Template



Schema: SCOTT

Objects to display: [✓ Tables] Views Indexes PL/SQL Synonyms Sequences

Name	Schema	Type
DEPT	SCOTT	Table
EMP	SCOTT	Table
SALGRADE	SCOTT	Table

Name	Schema	Type
BONUS	SCOTT	Snapshot

Status: Ready

4. Press the **Add (right arrow)** button to add the selected objects to the **Template Objects** list.
5. Repeat steps 1 — 5 above to add another existing object to your deployment template.
6. After you have added existing objects to your deployment template you have several options:
   - **Edit Template Objects**: Complete the steps in the following section "Edit Template Objects" to modify the characteristics of your template object (e.g. to define a WHERE clause, change refresh type, etc.).
   - **Add New Objects to Template**: The steps described in the "Add New Objects to Template" on page 4-30 illustrate how to add a new object to your deployment template (helpful for creating views or triggers at the remote snapshot site).
   - **Proceed to Parameter Definitions**: If you have added all of the necessary objects to your template and you do not need to modify any of these objects, press the **Next** button to proceed to the template parameter definition screen.

Edit Template Objects

After you add an existing object to your deployment template, you have the ability to customize the selected object. In most cases, you will edit a snapshot object to define a WHERE clause, modify storage parameters, and select individual columns for replication (column subsetting). Complete the following to modify an existing object:

1. Select the object to edit from the list of **Template Objects**.
2. Press the **Edit** button.

   The following four sections describe how to modify your selected template object:

   - "Edit General Information" on page 4-28
   - "Edit Refresh Information" on page 4-28
   - "Edit Storage Information" on page 4-29
   - "Edit Column Subsetting Information" on page 4-29

Note: Not all tabs and settings are available for each type of object. For example, the **Refresh** and **Column Subsetting** tabs are available only when a snapshot is selected. Additionally, the **Updateable** checkbox on the **General** tab is only available when a snapshot is selected.

Edit General Information

1. Verify that the **General** tab is selected.
2. If necessary, modify the **Name**, **Schema**, and **Tablespace** for the selected object.
3. Enable the **Updateable** checkbox if you want the selected snapshot to be updateable.

   In most cases, you will want to leave the **Logging** and **Cache** checkboxes enabled (press the **Help** button to view detailed information about these properties).
4. If necessary, modify the **Snapshot Index** characteristics to meet your snapshot needs.
5. Enter a WHERE clause in the **Where Clause** field to define a horizontal subset of data to replicate to your snapshot.

   If you want to create a parameterized template object, insert the variable name preceded by a colon (:) in your WHERE clause. For example, you might enter the following:

   ```
   WHERE empno = :employee_id
   ```

   An `employee_id` variable will automatically be created; you can use this variable to define a default value for the template or unique values for individual users.

Edit Refresh Information

1. Press the **Refresh** tab.
2. If necessary, modify the **Refresh Type** and **Refresh Method** for the selected object.
3. If necessary, modify the **Rollback Segment** characteristics to meet your refresh needs (press the **Help** button to view detailed information about these properties).

Edit Storage Information

1. Press the **Storage** tab.
2. If necessary, modify the storage characteristics to meet your snapshot needs (press the **Help** button to view detailed information about these properties).

Edit Column Subsetting Information

1. Press the **Column Subsetting** tab.
2. To exclude columns from the replicated snapshot:
   - Select the columns to exclude from the list of **Replicated Columns**.

     You can only select columns that have been identified to be “skipped” during conflict checking. See "Identifying Subset Columns" on page 2-29 for more information.
   - Press the **Remove** button.

     The selected column is no longer displayed in the list of **Replicated Columns**.
3. To add columns that were previously removed from a replicated snapshot:
   - Select the column group that contains the target column from the list of **Column Groups** (if no specific column group has been defined, select the `Shadow Group`).
   - Select the column to add from the list of **Selected Column Groups**.
   - Press the **Add** button.

     The selected column is now displayed in the list of **Replicated Columns**.

     Press the **OK** button to complete editing your deployment template object.
4. After you have edited the template object you have several options:
   - **Edit Another Template Object**: Repeat the steps in the section "Edit Template Objects" to modify the characteristics of another template object.
   - **Add New Objects to Template**: The steps described in the "Add New Objects to Template" on page 4-30 illustrate how to add a new object to your deployment template (helpful for creating views or triggers at the remote snapshot site).
   - **Proceed to Parameter Definitions**: If you have added all of the necessary objects to your template and you have complete all necessary modifications, press the **Next** button to proceed to the template parameter definition screen.

Add New Objects to Template

You may want to create a new object at the snapshot site that does not exist at the master site. For example, you may want to create views or non-replicated tables at the snapshot site. The following procedure shows you how to define the DDL to execute at the snapshot site when the deployment template is instantiated:

1. Press the **Create** button.

   The Create Template Object dialog box appears.

2. Enter the name of your new template object in the **Name** field.
3. Select the type of object you are adding to your template from the **Type** pull-down list.
4. Enter the DDL to create your template object in the **Enter DDL Text** field.

   Do not include the semi-colon at the end of your object DDL. Also, if you wish to create a parameter in your DDL, be sure to place a colon (:) at the beginning of your parameter (e.g. :region).For example, you might enter the following:

   ```
   CREATE SNAPSHOT scott.emp_snap AS
      SELECT * from scott.emp@orc1.world
      WHERE empno = :employee_id
   ```

   An `employee_id` variable will automatically be created; you can use this variable to define a default value for the template or unique values for individual users (the next page of the Deployment Template Wizard allows you to customize your parameters).

5. Press the **OK** button when you have completed defining your new template object.

   You will be returned to the main body of the Deployment Template wizard; the new object will appear in the list of **Template Objects**.

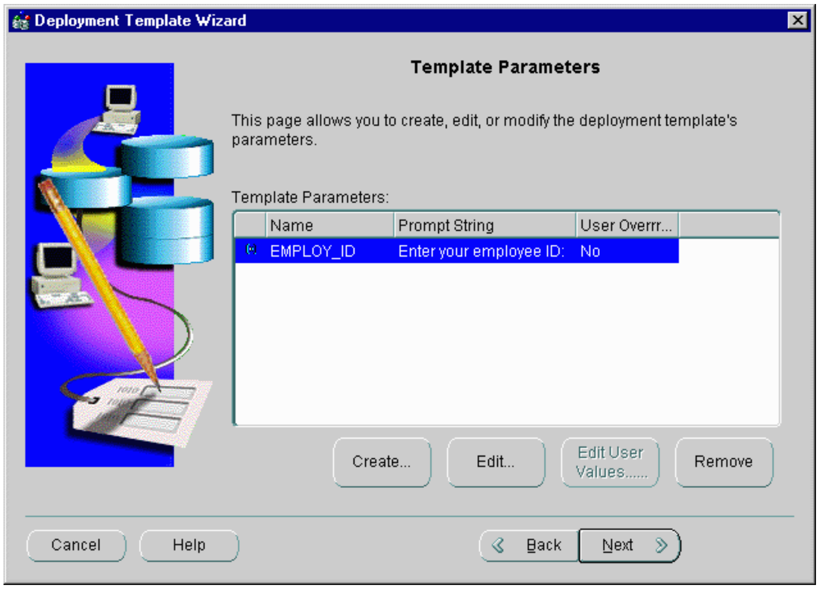
6. Repeat steps 1 — 5 until you have added all new objects to your template.
7. After you have added existing objects to your deployment template you have several options:

   - **Edit Template Objects**: Complete the steps in the section "Edit Template Objects" on page 4-27 to modify the characteristics of your template object (e.g. to define a WHERE clause, change refresh type, etc.).
   - **Proceed to Parameter Definitions**: If you have added all of the necessary objects to your template and you do not need to modify any of these objects, press the **Next** button to proceed to the template parameter definition screen.

Template Parameters

When you add a template object to your deployment template, Oracle evaluates the DDL to see if any parameters have been specified (parameters are prefixed with a “:”). If any parameters have been specified in the DDL, use the Template Parameters page of the deployment template wizard to define default parameter values and prompt text. You can also use this form to create new parameters and define unique parameter values for individual users. See "Template Parameters" on page 4-6 for more information about user parameter values.

Figure 4–13 Define Template Parameters for Custom Instantiation



This page allows you to create, edit, or modify the deployment template's parameters.

Template Parameters:

Name	Prompt String	User Overrr...
EMPLOY_ID	Enter your employee ID:	No

Edit Template Parameter

If you defined a parameter in a template object’s DDL, you can select the parameter and add a default parameter value, define prompt text, and specify whether the user can override the default value. Complete the following steps to edit a parameter:

1. Select the parameter to edit from the list of **Template Parameters**.
2. Press the **Edit** button.
3. If necessary, modify the name of your template parameter in the **Name** field.

If you modify the name of an existing parameter, you need to edit the DDL for the objects that use the existing parameter (modifying the parameter name will not rename the parameter in existing DDL).

4. Enter the text that you want a user to see if prompted for a parameter value in the **Prompt String** field.
5. Enable the **User Override** checkbox if you want to define user specific user parameter values.
6. Enter the default value for your parameter in the **Default** field.
7. Press the **OK** button when you have completed editing your template parameter.
8. Repeat steps 1 — 7 until you have completed editing your exiting template parameters.
9. After you have edited any existing template parameters, you have several options:
   - **Create User Parameter Values**: To help template deployment, you can create unique parameter values for individual users. This feature allows users to instantiate a template with a data set that is unique to them without causing you to modify the template or any scripts. Complete the steps in the "Create User Parameters" section on page 4-32.
   - **Finish Deployment Template Wizard**: If you do not want to create a new template parameter or do not want to define any user parameter values, press the **Next** button to proceed to the last page of the Deployment Template wizard.

Create User Parameters

If you have parameters defined in your deployment template, you have the option of defining user parameter values. When a particular user instantiates your deployment template, the value that you specify for that user will be used (this is helpful when you want to replicate custom data sets to individual users). See "Template Parameters" on page 4-6 for more information about user parameter values.

1. Select the template parameter for which you want to define user values from the list of **Template Parameters**.
2. Press the **Edit User Values** button.

3. Select the users for which you want to define user values from the list of **Available Users** and press the **Add** button.

   Press the <SHIFT> key to select a range of users or press the <CTRL> key to individually select multiple users.

   If you are working with a public template, all database users will be listed in the **Available Users** list. If you are working with a private template, only authorized users will be listed.

4. Select the user or users for whom you want to define a parameter value from the list of **Selected Users** (select multiple users as instructed above).
5. Enter the user value for the selected user(s) in the **Parameter Value** field (if you have selected multiple users, the entered value will be used for each user).
6. Repeat steps 4 - 5 until you have defined all user values for the selected parameter.

   If you do not know all of the user parameter values at this time, you will be able to specify additional user parameter values by modifying your deployment template or during the instantiation process.

7. Press the **OK** button.
8. Repeat steps 1 - 7 until you have defined all of the necessary user values for your individual template parameters.
9. Press the **Next** button to proceed to the last page of the Deployment Template wizard.

Finish Deployment Template Wizard

Once you reach the Finish page of the Deployment Template wizard, you have defined all of the objects for your deployment template and optionally defined any template parameters and user parameter values. After Replication Manager completes building the deployment template, you will be ready to instantiate this deployment template at remote snapshot sites.

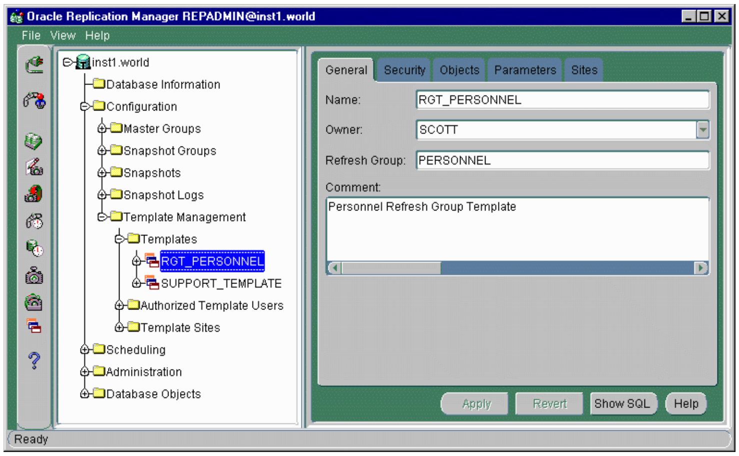
1. Press the **Finish** button to complete building your deployment template.

   Your new deployment template will be displayed under the **Template Management | Templates** node (see Figure 4–14).

Modifying a Deployment Template

Before you begin editing your deployment template, make sure that you expand the **Template Management | Templates** node in the left pane (tree) of the Replication Manager user interface.

Figure 4–14 Template Management Node and Template Property Sheet



Name: RGT_PERSONNEL
Owner: SCOTT
Refresh Group: PERSONNEL
Comment: Personnel Refresh Group Template

Note: Once a template has been instantiated at a snapshot site, you will not be able to modify the template. Copy the template and make the necessary modifications (see "Copy Template" on page 4-43 for more information).

Template Properties

The **General** page of the Deployment Template property sheet allows you to modify several characteristics of a deployment template, including the target refresh group, owner, and deployment template name.

If you need to modify the general properties of any existing deployment template, the following steps will guide you through the modification process:

1. Verify that the **Template Management | Templates** node is expanded.

2. Select the deployment template that you want to modify.

   The property sheet for the selected deployment template will appear in the right-pane of the Replication Manager user interface, displaying the current settings for the selected deployment template.

3. Modify the contents of the **Name, Owner**, **Refresh Group**, and **Comment** fields as necessary.

   Name: The name of the deployment template.

   Owner: The name of the user responsible for maintaining the deployment template.

   Refresh Group: The name of the refresh group to which all snapshots in the deployment template will be added during the instantiation process.

   Comment: User comments are stored in this field.

4. Press the **Apply** button when you have made all of the necessary modifications to the selected deployment template.

Add Template Objects

Since your replication requirements may grow, you may need to add additional replicated objects to your deployment template.

You can add template objects using one of two methods: create a new template object or create from an existing object.

Creating a new template object requires that you specify the object DDL used to create the object at the remote snapshot site (e.g. CREATE SNAPSHOT AS...). Adding an object based on existing object requires that you select the existing object and Oracle will determine the DDL for you.

Complete the following steps and, depending on how you want to create the new object, complete the steps in either the Add Template Objects or Add New Objects to Template sections:

1. Verify that the **Template Management | Templates** node is expanded.
2. Select the deployment template that you want to modify.

   The property sheet for the selected deployment template appears in the right-pane of the Replication Manager user interface, displaying the current settings for the selected deployment template.

3. Press the **Objects** page.

4. Press the **Add** button.

   Complete the steps in one of the following two sections:

   - Add Template Objects
   - Add New Objects to Template

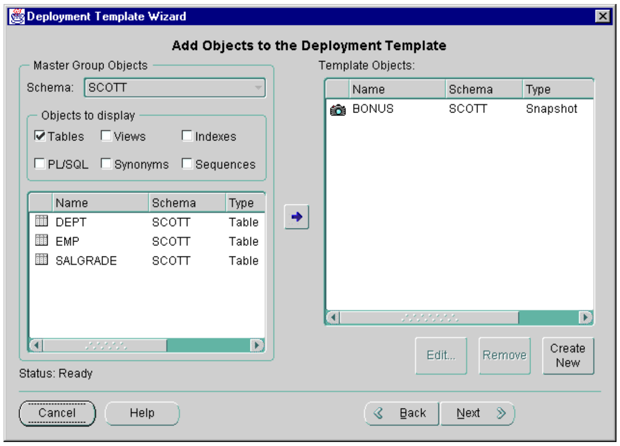
Add Template Objects

Complete the following steps to add an object to your template by using an existing object:

1. Select the schema from the **Schema** pull-down list that contains the replicated object that you want to add to your deployment template.
2. Enable the checkbox(es) corresponding with the type(s) of objects that you want to add.
3. Select the master group object(s) in the **Available Objects** list that you want to add to your deployment template.

   Press the <SHIFT> key to select a range of objects or press the <CTRL> key to individually select multiple objects.

Figure 4–15 Add Replicated Object to Deployment Template



Schema: SCOTT

[✓] Tables [ ] Views [ ] Indexes
[ ] PL/SQL [ ] Synonyms [ ] Sequences

Name	Schema	Type
DEPT	SCOTT	Table
EMP	SCOTT	Table
SALGRADE	SCOTT	Table

Name	Schema	Type
BONUS	SCOTT	Snapshot

Status: Ready

4. Press the **Add (right arrow)** button to add the selected objects to the **Template Objects** list.

5. Repeat steps 1 — 4 above to add another existing object to your deployment template.
6. After you have added existing objects to your deployment template you have several options:
   - **Edit Template Objects**: Complete the steps in the following section "Edit Template Objects" to modify the characteristics of your template object (e.g. to define a WHERE clause, change refresh type, etc.).
   - **Add New Objects to Template**: The steps described in the "Add New Objects to Template" on page 4-40 illustrate how to add a new object to your deployment template (helpful for creating views or triggers at the remote snapshot site).
   - **Close Deployment Template Wizard**: If you have added all of the necessary objects to your template and you do not need to modify any of these objects, press the **Finish** button to finish adding these objects to your deployment template.

Edit Template Objects

After you add an existing object to your deployment template, you have the ability to customize the selected object. In most cases, you will edit a snapshot object to define a WHERE clause, modify storage parameters, and select individual columns for replication (column subsetting). Complete the following to modify an existing object:

1. Select the object to edit from the list of **Template Objects**.
2. Press the **Edit** button.

   The following four sections describe how to modify your selected template object:

   - "Edit General Information" on page 4-38
   - "Edit Refresh Information" on page 4-38
   - "Edit Storage Information" on page 4-39
   - "Edit Column Subsetting Information" on page 4-39

Note: Not all tabs and settings are available for each type of object. For example, the **Refresh** and **Column Subsetting** tabs are available only when a snapshot is selected. Additionally, the **Updateable** checkbox on the **General** tab is only available when a snapshot is selected.

Edit General Information

1. Verify that the **General** tab is selected.
2. If necessary, modify the **Name**, **Schema**, and **Tablespace** for the selected object.
3. Enable the **Updateable** checkbox if you want the selected snapshot to be updateable.

   In most cases, you will want to leave the **Logging** and **Cache** checkboxes enabled (press the **Help** button to view detailed information about these properties).
4. If necessary, modify the **Snapshot Index** characteristics to meet your snapshot needs.
5. Enter a WHERE clause in the **Where Clause** field to define a horizontal subset of data to replicate to your snapshot.

   If you want to create a parameterized template object, insert the variable name preceded by a colon (:) in your WHERE clause. For example, you might enter the following:

   ```
   WHERE empno = :employee_id
   ```

   An `employee_id` variable will automatically be created; you can use this variable to define a default value for the template or unique values for individual users.

Edit Refresh Information

1. Press the **Refresh** tab.
2. If necessary, modify the **Refresh Type** and **Refresh Method** for the selected object.
3. If necessary, modify the **Rollback Segment** characteristics to meet your refresh needs (press the **Help** button to view detailed information about these properties).

Edit Storage Information

1. Press the **Storage** tab.
2. If necessary, modify the storage characteristics to meet your snapshot needs (press the **Help** button to view detailed information about these properties).

Edit Column Subsetting Information

1. Press the **Column Subsetting** tab.
2. To exclude columns from the replicated snapshot:
   - Select the columns to exclude from the list of **Replicated Columns**.

     You can only select columns that have been identified to be "skipped" during conflict checking. See "Identifying Subset Columns" on page 2-29.
   - Press the **Remove** button.

     The selected column is no longer displayed in the list of **Replicated Columns**.
3. To add columns that were previously removed from a replicated snapshot:
   - Select the column group that contains the target column from the list of **Column Groups** (if no specific column group has been defined, select the `Shadow Group`).
   - Select the column to add from the list of **Selected Column Groups**.
   - Press the **Add** button.

     The selected column is now displayed in the list of **Replicated Columns**.

     Press the **OK** button to complete editing your deployment template object.
4. After you have edited the template object you have several options:
   - **Edit Another Template Object**: Repeat the steps in the section "Edit Template Objects" on page 4-37 to modify the characteristics of another template object.
   - **Add New Objects to Template**: The steps described in the "Add New Objects to Template" on page 4-40 illustrate how to add a new object to your deployment template (helpful for creating views or triggers at remote site).
   - **Close Deployment Template Wizard**: If you have added all of the necessary objects to your template and you do not need to make any further modifications to these objects, press the **Finish** button to finish adding these objects to your deployment template.

Add New Objects to Template

You may want to create a new object at the snapshot site that does not exist at the master site. For example, you may want to create views or non-replicated tables at the snapshot site. The following procedure will show you how to define the DDL to execute at the snapshot site when the deployment template is instantiated:

1. Press the **Create** button.

   The Create Template Object dialog box will appear.

2. Enter the name of your new template object in the **Name** field.

3. Select the type of object you are adding to your template from the **Type** pull-down list.

4. Enter the DDL to create your template object in the **Enter DDL Text** field.

   Do not include the semi-colon at the end of your object DDL. Also, if you wish to create a parameter in your DDL, be sure to place an colon (:) at the beginning of your parameter (e.g. :region).For example, you might enter the following:

   ```
   WHERE empno = :employee_id
   ```

   An `employee_id` variable will automatically be created; you can use this variable to define a default value for the template or unique values for individual users (the next page of the Deployment Template Wizard will allow you to customize your parameters).

5. Press the **OK** button when you have completed defining your new template object.

   You will be returned to the main body of the Deployment Template wizard; the new object will appear in the list of **Template Objects**.

6. Repeat steps 1 — 5 until you have added all new objects to your template.

7. After you have added existing objects to your deployment template you have several options:

   - **Edit Template Objects**: Complete the steps in the section "Edit Template Objects" on page 4-37 to modify the characteristics of your template object (e.g. to define a WHERE clause, change refresh type, etc.).

   - **Close Deployment Template Wizard**: If you have added all of the necessary objects to your template and you do not need to modify any of these objects, press the **Finish** button to finish adding these objects to your deployment template.

Modify Existing Template Objects

If you need to modify an existing template object's DDL to, for example, add a parameter, or you need to change an object's rollback segment, you can use the Template Objects Property page to make such modifications.

Complete the following steps to modify an existing deployment template object:

1. Verify that the **Template Management | Templates** node is expanded.
2. Expand the deployment template node that you want to modify.
3. Expand the **Objects** node.
4. Select the template object that you want to modify

   A property sheet for the selected deployment template object will appear in the right-pane of the Replication Manager user interface.
5. Modify the contents of the **Object Name**, **Object DDL**, **Master Rollback Segment**, and **Build Order** fields as necessary.
   - **Object Name**: The name of the deployment template object.
   - **Object DDL**: The DDL that creates the selected object at the remote snapshot site.
   - **Master Rollback Segment**: The name of the rollback segment used during the object creation at the remote snapshot site.
   - **Build Order**: Identifies where in the sequence this object will be built at the remote snapshot site (an object with a build order of 1 will be built before an object with a build order of 2).
6. Press the **Apply** button when you have completed the necessary modifications.

Remove Template Objects

If a template object or objects are no longer required, you can use Replication Manager to remove these objects from a deployment template. You can either remove all deployment template objects at once or remove them individually:

1. Verify that the **Template Management | Templates** node is expanded.
2. Select the deployment template node that you want to modify.
3. Press the **Objects** tab.
4. Select the object to delete from the **Name** pull-down list.
5. Press the **Remove** button.

Template Parameters

Each time you add a new authorized user for a specific template, you may need to define one or more user parameter values. The following steps guide you through modifying template parameters and user parameter values:

1. Verify that the **Template Management | Templates** node is expanded.
2. Expand the deployment template node that contains the parameters that you want to modify.
3. Expand the **Parameters** node.
4. Select the parameter that you want to modify.

   A property sheet for the selected deployment template parameter appears in the right-pane of the Replication Manager user interface.
5. If necessary, modify the contents of the **Name**, **Prompt String**, **User Override**, and **Default Value** fields as necessary.
   - **Name**: The name of the template parameter (if you change a parameter name, be sure that you update any template objects that use this parameter).
   - **User Override:** Allows you to define user specific parameter values (if disabled, user will automatically use the default value)
   - **Prompt String**: The text a user will see when prompted for a parameter value
   - **Default Value**: The default value for the selected template parameter.
6. Press the **User Values** tab.
7. Select the users that you want to edit from the list of **Available Users** and press the **Add** button. If the users that you want to edit are already listed in the **Selected Users** list, skip to step 8.

   Press the <SHIFT> key to select a range of users or press the <CTRL> key to individually select multiple users. If you are working with a public template, all database users will be listed in the **Available Users** list. If you are working with a private template, only authorized users will be listed.
8. Select the user or users that you want to define a parameter value for from the list of **Selected Users** (select multiple users as instructed above).
9. Enter the user value for the selected user(s) in the **Parameter Value** field (if you have selected multiple users, the entered value will be used for each user).

10. Repeat steps 8 - 9 until you have defined all user values for the selected parameter.

    If you do not know all of the user parameter values at this time, you will be able to specify additional user parameter values by modifying your deployment template again or during the instantiation process.

11. Press the **Apply** button when you have made all of the necessary modifications.

Copy Template

There may be cases when you need to copy a deployment template to multiple master sites. For example, this may be necessary before you deploy the templates so that the snapshot sites will be distributed across multiple master sites to achieve more efficient load balancing. You may also need to create a new template that contains many objects contained in an existing template. It is easy to copy an existing template with a new name and modify as necessary instead of creating the template from "scratch."

1. Select **Copy Deployment Template** from the **File** menu.
2. If you do not want to copy the user authorizations for the selected template, disable the **User Auth** checkbox next to the appropriate deployment template(s).
3. Enter a name for the copied template in the **New Template Name** field next to appropriate template(s). If you do not specify a name, it will be copied as <*template name*>_copy.
4. Select the template(s) that you want to copy from the list of templates.

   Press the <SHIFT> key to select a range of templates or press the <CTRL> key to individually select multiple templates.
5. Select the appropriate radio-button to define the destination of the copied template:
   - **Copy to the Same Master Site**: The copied template will be copied to the site where the original template is located.
   - **Copy to Different Master Site(s)**: The copied template will be copied to the sites that you select from the Database Link Name list (select multiple sites to copy the template to more than one site).
6. Press the **Finish** button.

Compare Template

If you have copied templates (see previous section), you may want to determine the differences between two templates. Replication Manager allows you to select two templates at the same site to compare. A report is generated that displays all of the differences at the object level between these two templates.

1. Select **Compare Deployment Templates** from the **File** menu.
2. Select the two templates that you want to compare from the **Template Name** list.

   Press the <SHIFT> key to select two sequentially listed templates or press the <CTRL> key to individually select two templates.
3. Enter the path where you want to store the results of the template compare in the **Save the Compare Results Into** field. (Press the **Browse** button to navigate to a target directory.)
4. Press the **Compare** button.

   The results of the compare operation are displayed at the bottom of the Compare Deployment Template Wizard. Results are also written to the path specified in step 3 above.
5. Repeat steps 1 — 4 to compare additional templates.
6. Press the **Cancel** button when you have completed comparing your deployment templates.

Remove Template

When a deployment template becomes obsolete, use Replication Manager to delete the target template.

Note: Removing a deployment template does not affect any remote snapshot sites that have already instantiated the template.

1. Verify that the **Template Management | Templates** node is expanded.
2. Right-click on the target deployment template and select **Remove**.

Deploying a Template

After a deployment template has been created and is ready for deployment, you need to package the template, prepare the remote snapshot site, and perform the actual deployment template instantiation.

Package Template

Once a deployment template contains all of the objects and parameters that are required by the snapshot environment, you need to prepare the deployment template for distribution to your end-users. You are essentially *packaging* the template to be instantiated at remote snapshot sites.

Replication Manager allows you to quickly and easily package your deployment templates for offline instantiation.

Note: The following procedures describe the steps to package your deployment template using Replication Manager. For information on packaging your template with the Replication Management API, see "Package for Instantiation" in the *Oracle8i Replication API Reference* book.

Package for Online Instantiation

If you are using Oracle's Client Instantiation tool, the packaging of the online instantiation file is automated and does not require you to generate and distribute an instantiation file.

For additional online instantiation options, see the "Package for Instantiation" section in Chapter 4 of the *Oracle8i Replication API Reference* manual.

Package for Offline Instantiation

After you have built your deployment templates, you may want to perform an offline instantiation of the template at a target snapshot site. Offline instantiation allows you to write all of the necessary DDL and data to an offline instantiation file that you will transfer to and run at your snapshot site.

This solution is ideal for laptop users who do not have high-speed LAN connections (which could make online instantiation difficult).

1. Expand the **Template Management** node.

2. Right-click on the **Templates** node and select **Off Line Instantiation File Generation** from the pop-up menu.
3. Select the template that you want to package from the **Select a Deployment Template** pull-down list.
4. Select the type of site for which you want to generate an instantiation file from the **Select the Type of Off Line Instantiation** object group:
   - **Oracle Server**: The offline instantiation file(s) is generated for sites that are running Oracle8*i* Enterprise Edition, Oracle8*i* Server, or Personal Oracle8*i*.
   - **Oracle Lite**: The offline instantiation file(s) is generated for sites that are running Oracle8*i* Lite.
5. Enter the path where you want to store the generated files in the **Please Enter a Location for the Generated Files** field (press the **Browse** button to navigate to the target directory).
6. Press the **Next** button.
7. Select the users who will instantiate the template at the remote snapshot site from the list of **Available Users**.

   Press the <SHIFT> key to select a range of users or press the <CTRL> key to individually select multiple users.
8. Press the **Add** (right arrow) button to add the users to the **Selected Users** list.
9. Enter the global database name of the site where the selected users will instantiate the deployment template in the **Site ID** column. You need to enter a global database name for each user that was added in step 8.

   If you are generating offline instantiation files for Oracle8*i* Lite sites, skip this step.
10. Press the **Next** button.
11. Select the parameter that you want to modify from the **Parameters** list.
12. Select the user or users for whom you want to define a new parameter value from the **Selected Users** list (if you select multiple users, each will be assigned the new value).

    Press the <SHIFT> key to select a range of users or press the <CTRL> key to individually select multiple users.
13. Enter the new parameter value in the **Parameter Value** field.

14. Repeat steps 12 - 13 to define new parameter values for other user(s). Repeat steps 11 - 13 if you want to define new values for another parameter.
15. Press the **Next** button.
16. Press the **Finish** button to begin generating your offline instantiation files.

    The generated files are stored in the location that you specified in step 5.

After you have generated the offline instantiation file, you need to distribute this file (that was saved in the directory that you specified during step 5) to the target end-user. You can distribute the file by posting it to an FTP or web site or by saving the file to some storage medium (CD-ROM, floppy disk, etc.). Your user needs this offline instantiation file to build the snapshot environment at the remote site.

Prepare Remote Snapshot Site for Instantiation

If a deployment template has been designed well, little preparation will need to be performed at the remote snapshot site. This section describes the most common preparations that need to be performed at the remote snapshot site. Once any required preparations have been completed, you are ready to perform either an online or offline instantiation.

Use the following questions as a guide to evaluate the remote snapshot site to determine what preparations need to be performed:

- Does the remote snapshot site have network connectivity to the target master site?
- Does the snapshot site have an Oracle8*i* Enterprise Edition, Oracle8*i* Server, or Personal Oracle8*i* release 8.1.5 or greater database? Or, does the snapshot site have an Oracle8*i* Lite release 4.0 or greater database?
- Has the remote snapshot site been setup to support snapshot replication?
- If required database links are not part of the deployment template, does the required database link from the snapshot site to the master site exist?
- Does the schema required by the deployment template exist at the snapshot site?
- Do the rollback segments required by the deployment template exist at the snapshot site and are they online?

Network Connectivity

As with all replicated environments, network connectivity is a key component in Oracle Advanced Replication. Verify that the remote snapshot site has a proper SQL*Net or Net8 connection to the target master site. If the snapshot site does not have the appropriate network connectivity, consult the *Net8 Administrator's Guide* for information about setting up an Oracle network connection.

Database Version

The snapshot site must have an Oracle8*i* release 8.1.5 or greater database to instantiate a deployment template. If your snapshot site is using Oracle8*i* Lite, release 4.0 or greater must be used. If your snapshot site does not meet the database version requirements, you need to upgrade your database before instantiating a deployment template.

Snapshot Site Setup

Each snapshot site needs several users that have special privileges to support a snapshot site. In addition to having the administrative privileges, these users also participate in the propagation and refreshing of data.

Snapshot site setup also includes scheduling several automated jobs to handle the automatic refreshing of the snapshot (optional) and the purging of the deferred transaction queue.

You can setup your snapshot site with:

- **Replication Manager:** You can connect to the remote snapshot site with Replication Manager and use the Setup Wizard. See "Setting Up Snapshot Site" on page 5-2 for details on setting up your snapshot site with Oracle Replication Manager.
- **Replication Management API:** Using the replication management API to setup your snapshot site is an ideal solution when you are not able to connect to the remote snapshot site with Replication Manager. When you build a SQL script containing the API calls to setup your snapshot site, you can also add the DDL and API calls to complete the remaining preparation (e.g. creating necessary schema, database links, and rollback segments, as described in the following three sections). The script that you create should be distributed with the offline instantiation file and executed before the offline instantiation file.

  See "Setup Snapshot Site" in Chapter 2 of the *Oracle8i Replication API Reference* manual.

Create Necessary Schema

If the deployment template that you are instantiating will create objects in multiple schemas, you need to make sure that all of the necessary schemas have been created. Additionally, the user instantiating the deployment template needs to have the appropriate CREATE privilege(s) on that schema. For example, if the deployment template will create a procedure in schema FOO and the user SCOTT is instantiating the template, SCOTT must have the CREATE ANY PROCEDURE privilege on schema FOO.

Create Database Links

While it is advantageous to include the DDL to create all necessary database links for a remote snapshot site in the deployment template, it is certainly not required. In the event that the database link DDL is not in the deployment template, you will need to manually create the database link to the target master site prior to instantiating the deployment template (the database link will be required to populate the snapshot base tables during an online instantiation and will of course be required for the proper operation of the snapshot environment).

Create Necessary Rollback Segment

If the deployment template that you are instantiating will use specific rollback segments that don't currently exist at the remote snapshot site, you will need to create the necessary rollback segment(s). To see if your template objects use the default rollback segment or a specific rollback segment, query the `dba_repcat_template_objects` view or complete the following steps in Replication Manager:

1. Expand the **Templates** node.
2. Expand the deployment template node with which you are working.
3. Expand the **Objects** node.
4. Select the object whose rollback segment you want to check.

   A property sheet for the selected deployment template object appears in the right-pane of the Replication Manager user interface. View the contents in the MASTER ROLLBACK SEGMENT field to determine which rollback segment will be used when the selected object is created at the remote snapshot site.

Remote Snapshot Site Instantiation

Instantiation of a deployment template can be performed at remote snapshot sites that are using Oracle8*i* Enterprise Edition, Oracle8*i* Server, Personal Oracle8*i*, and/or Oracle8*i* Lite.

The following section describes how to perform an instantiation at a remote Oracle8*i* Enterprise Edition, Oracle8*i* Server, or Personal Oracle8*i* site using an instantiation scrip; for more information about performing an instantiation at an Oracle8*i* Lite site, please see the Oracle8*i* Lite documentation.

The instantiation scrips can either be generated using Replication Manager (see "Package for Offline Instantiation") or by using the replication management API (see "Package for Instantiation" section in chapter 4 of the *Oracle8i Replication API Reference* manual). The procedure in the next section will work for either offline or online insanitation of a deployment template.

Note: It is recommended that you test the following instantiation method with the files that you have generated before distributing these files to your end-users. Additionally, you are encouraged to provide a variation of the following instruction to your end-users with specific information (e.g. exact file names and file locations); the more testing and information that you provide to your end-users, the better the deployment template process will proceed.

Instantiate at Remote Snapshot Site

Before the end-user can begin the instantiation process at the remote snapshot site, make sure that the end-user has received the generated instantiation file (see "Package for Offline Instantiation" on page 4-45 or the "Package for Instantiation" section in chapter 4 of the *Oracle8i Replication API Reference* manual).

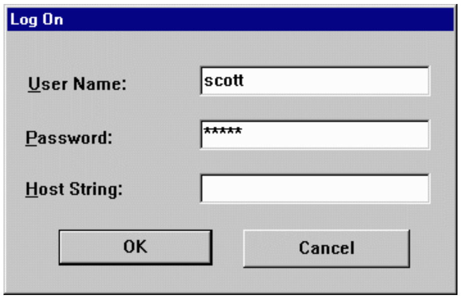
This section describes how to use SQL*Plus to perform the instantiation. If the end-user has the Oracle Client Instantiation tool, please see the appropriate documentation for instantiation instructions.

The following steps are to be performed at the remote snapshot site:

1. Start SQL*Plus.
2. Login to the local database as the owner of the snapshot environment.

For example, if SCOTT will be the owner of the snapshot objects contained in the deployment template, you will need to connect as SCOTT to the local database.

Figure 4–16 Login to local database using SQL*Plus at Snapshot Site


Log On

User Name: scott

Password: *****

Host String:

Note: If your snapshot database has more than one Oracle database running, you will need to specify the proper **Host String** value. If you do not know your host string, contact your database administrator.

---

3. If you have received a script file from your administrator to setup your snapshot site, execute it now by typing

   ```
   RUN <path\filename>
   ```

   at the `SQL>` prompt (where *`path\filename`* is the location and name of the snapshot site setup script). For example, you might execute the following:

   ```
   SQL> RUN d:\temp\snap_setup.sql
   ```

Note: If you have not received a snapshot setup script, you need to verify that your snapshot site has been setup to support snapshot replication before continuing with step 4.

If you have not received a snapshot site setup script, skip to step 4.

4. Execute the instantiation script by typing

   ```
   RUN <path\filename>
   ```

   at the `SQL>` prompt (where *`path\filename`* is the location and name of the generated offline instantiation file). For example, you might execute the following:

   ```
   SQL> RUN d:\temp\personnel.sql
   ```

5. If you have executed an offline instantiation script, be sure to refresh your snapshot environment with the master site. Refreshing your snapshots will make the data current with the associated master tables.

After Deployment

There is very little that you need to do after you have completed instantiating the deployment template at the remote snapshot site. The most critical step is to perform a fast refresh after performing an offline instantiation.

Refresh Snapshots

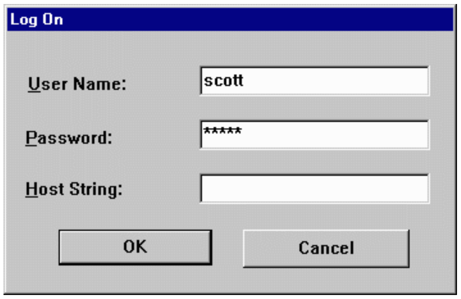
After you have performed an offline instantiation of a deployment template, you will need to perform a fast refresh of your snapshots. It is important that you refresh as soon as possible; the longer the duration between packaging the template and actual instantiation, the greater the amount of changes being stored in the snapshot logs for the snapshots in your deployment template.

If you do not have the Oracle Client Instantiation tool and/or your application cannot refresh your snapshot environment, the following procedure illustrates how to refresh your snapshot environment using SQL*Plus:

1. Start SQL*Plus.
2. Login to the local database as the owner of the snapshot environment.

   For example, if SCOTT will be the owner of the snapshot objects contained in the deployment template, you will need to connect as SCOTT to the local database.

Figure 4–17 Login to local database using SQL*Plus at Snapshot Site



User Name: scott
Password: *****
Host String:

Note: If your snapshot database has more than one Oracle database running, you will need to specify the proper **Host String** value. If you do not know your host string, contact your database administrator.

---

3. After you have successfully logged into your database, refresh your snapshots by typing

```
EXECUTE DBMS_REFRESH.REFRESH('<refresh_group>');
```

at the SQL> prompt (where *refresh_group* is the name of the refresh group that your snapshots have been added to — in most cases, the name of the refresh group has the same name as the deployment template). For example, you might execute the following:

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

Contact your database administrator for more information about the name of the refresh group that you should specify when executing the REFRESH procedure.

Grant Privileges

After your snapshot environment has been created, you will need to grant users at the snapshot site the privileges required to view, update, insert, and delete the data contained in the newly created replicated objects.

Local Control of Snapshot Creation

A Deployment Template is the most effective method of building and distributing a snapshot environment, even if distribution is limited. Even if distribution is limited to only two or three sites, you still significantly reduce the amount of steps needed to build a snapshot environment using Deployment Templates as opposed to individually creating the snapshot environment at those two or three sites. With deployment templates, you build once and distribute as needed.

One question remains though: If a deployment template is the most effective means for building and distributing a snapshot environment, when should you locally build the snapshot environment at the remote snapshot site? In most cases, you will want to build a snapshot environment using the Snapshot Group wizard or locally at the snapshot site when local control needs to be maintained at the snapshot site.

One scenario where you might find local control of snapshot creation helpful is when it is desirable for the snapshot site to control what data it receives. For example, this is especially true of decision support sites (DSS), which are typically read-only snapshot sites. A DDS site may occasionally need to run complex queries and they do not want to slow the OLTP site and/or bother the DBA at the OLTP site.

Local Snapshot Control

One of the major benefits of Deployment Templates is that control is maintained centrally by the DBA building the deployment template. There are instances, however, when the snapshot site needs to retain some control.

Local control may be required if the snapshot site:

- has an experienced DBA
- is considered a trusted site
- and/or the snapshot site is a snapshot instead of a master site because of horizontal data partitioning requirements

Since snapshot groups are created with the Replication Manager Snapshot Group wizard locally at the snapshot site by its DBA or maybe a systems analyst with SQL knowledge, control can also be maintained at the snapshot site.

Consider the following as a perfect example for maintaining local control. Since multi-master replication does not allow for data partitioning, updateable snapshot sites are sometimes created primarily for their ability to partition data. These sites are typically secure, have experienced DBAs, and will require the ability to maintain control locally to meet user and application requirements. Snapshot groups created with the wizard or with the API allow for the localized control necessary to meet the requirements of the secure updateable snapshot site(s).

Also remember that when a snapshot environment is created with a deployment template, all objects in the snapshot environment are added to the same refresh group. While this might be OK for most installations, certain situations may require that the objects in a snapshot group are assigned to several different refresh groups.

5

Directly Create Snapshot Environment

This chapter explains how to build a snapshot environment while directly connected to the remote snapshot site. Instead of pre-creating the snapshot environment at a master site with deployment templates, this chapter will illustrate how to build the snapshot environment directly at the remote snapshot site. This chapter covers the following topics:

Overview

If deployment templates do not meet your requirements for building and distributing a snapshot environment, you have two other options:

- Build individual snapshot sites with replication manager.
- Build individual snapshot sites with API (see the Oracle8*i* Server Replication API Reference for more information).

This chapter will describe the concepts and procedures of building a snapshot site using Oracle's Replication Manager.

Setting Up Snapshot Site

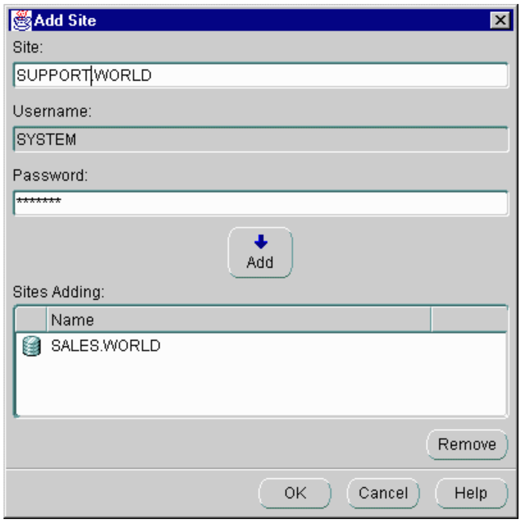
Before you can begin to replicate using snapshots, you need to setup your remote snapshot site to support replication. Oracle's Replication Manager contains a wizard that allows you to quickly and easily prepare your remote site to support snapshots.

Complete the following steps to setup your remote snapshot site:

1. Select **Setup Wizard** from the **File** menu.
2. Enable the **Setup Snapshot Sites** radio-button and press the **Next** button.
3. Enter the TNSNAMES alias for the target master site in the **Master Site** field.

   This is the site that contains the master objects for your snapshot site (this site will be the target for the database links created at the snapshot site).
4. Enter the password for the user `SYSTEM` (at the master site) in the **SYSTEM Password** field and press the **Next** button.
5. Press the **Add** button to identify the remote database(s) that you want to setup for snapshot replication.
6. Enter the TNSNAMES alias for the new snapshot site in the **Site** field.
7. Enter the password for the user `SYSTEM` (at the site specified in step 6) in the **Password** field and press the **Add** button (down arrow).

Figure 5–1 Identify Snapshot Sites with the Add Site Form



Site: SUPPORT.WORLD
Username: SYSTEM
Password: ********
Sites Adding:

Name
SALES.WORLD

8. Repeat steps 6 — 7 to setup additional snapshot sites.

9. Press the **OK** button when you have identified all of the remote sites to setup for snapshot replication.

10. Press the **Next** button.

11. Enter a new snapshot administrator password in the **Password** field of the **Administration** object group.

    To maintain security between the master and snapshot site, you are encouraged to change the default password (SNAPADMIN) to another alphanumeric password of at least 8 characters in length.

12. If necessary, enable the **Use a propagator different from the administrator** checkbox to specify a different user to handle the propagating and receiving functions.

    a. If you enabled the **Use a propagator different from the administrator** checkbox, enter a new name in the **Schema** field of the **Propagator** object group.

    b. Enter a new propagator password in the **Password** field of the **Propagator** object group.

Using a Different Propagator

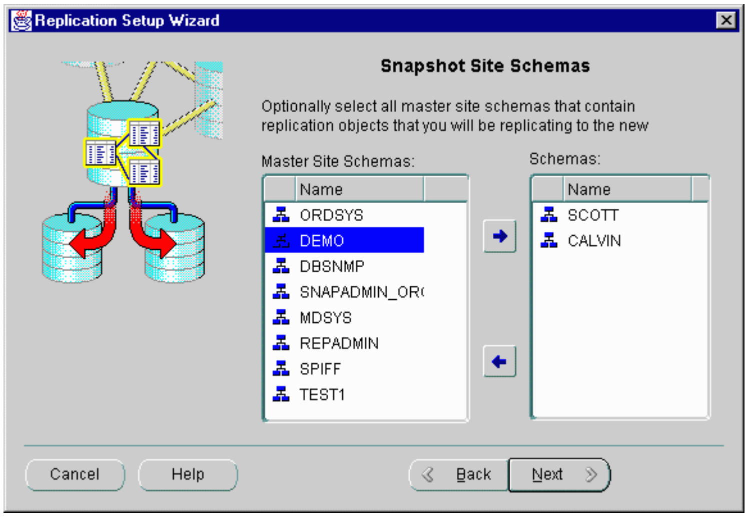
When you use the Setup Wizard to create your snapshot site, Replication Manager will use the ADMINISTRATOR user to manage both administrative and propagation tasks by default.

For security reasons, you may want to identify a different “propagator” user to manage the propagation tasks. This user will have significantly less privileges than the administrator (only the privileges required to propagate deferred transaction will be granted to the “propagator” user).

13. Press the **Next** button after you have completed configuring your default users.
14. Select the schema(s) at the target master site that contains the objects that will be replicated to the snapshot site(s) and press the **Add** button (right arrow).

    Press the <SHIFT> key to select a range of schema or press the <CTRL> key to individually select multiple schema.

Figure 5–2 Select Master Site Schemas to Replicate to Snapshot Site(s)



Snapshot Site Schemas

Optionally select all master site schemas that contain replication objects that you will be replicating to the new

Master Site Schemas:

Name
ORDSYS
DEMO
DBSNMP
SNAPADMIN_OR(
MDSYS
REPADMIN
SPIFF
TEST1

Schemas:

Name
SCOTT
CALVIN

15. Press the **Next** button.
16. If necessary, modify the Default Link Scheduling settings and press the **Next** button.

In most cases you will only modify the Schedule settings. These settings control how often the deferred transaction queue is pushed to the target master site and the snapshot is refreshed.

Press the **Help** button to view details about the individual settings on this page.

17. If necessary, modify the Default Purge Job Scheduling settings and press the **Next** button.

The Purge Job Scheduling settings define how often the deferred transaction queue is purged. Typically once a day (sysdate +1/24) is adequate for most replication installations.

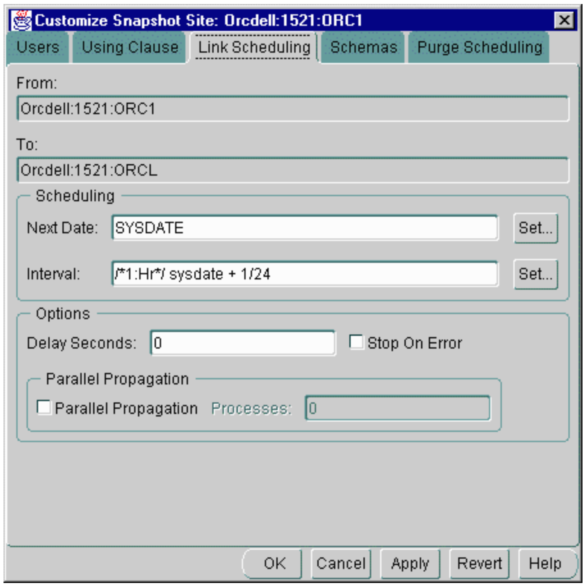
Press the **Help** button to view details about the individual settings on this page.

18. If necessary, select a snapshot site and press the **Customize** button to modify the settings of a single snapshot site.

The Customize feature is helpful if you are using the Setup Wizard to configure more than one snapshot site at a time.

If you do customize a default site, you will notice that you can select the available tabs to modify the settings specified during the Setup Wizard.

Figure 5–3 Use Customize Snapshot Form to Modify a Single Snapshot Site



From: Orcdell:1521:ORC1

To: Orcdell:1521:ORCL

Scheduling

Next Date: SYSDATE

Interval: /*1:Hr*/ sysdate + 1/24

Options

Delay Seconds: 0

☐ Stop On Error

Parallel Propagation

☐ Parallel Propagation Processes: 0

Press the **OK** button to complete the customization and return to the Setup Wizard.

19. Press the **Next** button when you have finished customizing the selected snapshot sites (or you want to skip customization).
20. Press the **Finish** button.

    If you would like Replication Manager to record a script instead of building the actual snapshot site, enable the **Record Script** checkbox.
21. Press the **OK** button when you have confirmed your settings.

    Replication Manager will now build (or generate a script for) your snapshot site with the setting you defined during the Setup Wizard.

Creating Snapshot Groups

A snapshot group in an advanced replication system maintains a partial or complete copy of all or some of the objects in a corresponding master group. Replication Manager has many features that help you create and manage snapshot groups. The following sections explain more about creating snapshot groups.

Before you create a new snapshot group, make sure that the following preliminary tasks have been completed.

- Run the Setup Wizard to prepare your snapshot site (see "Setting Up Snapshot Site" on page on page 5-2 for details).
- Create the necessary Snapshot Logs to support FAST refreshable snapshots (see "Create a Snapshot Log" on page on page 3-30 for details).

To create a new snapshot group:

1. Select the database in the left pane of the Replication Manager user interface where you want to create the snapshot group.

   If you do not see the target database, you may need to close Replication Manager and relogin to the snapshot site.
2. Select **Create** from the **File** menu
3. Select **Snapshot Group** from the sub-menu.

   You can optionally press the **Create Snapshot Group** button on the toolbar.
4. Select the **Available** radio-button that matches the type of database link that you want to access the target master database with.

- **Scheduled**: Displays all database links that are currently scheduled to push a deferred transaction queue.
- **Public**: Displays all PUBLIC database links.
- **All**: Displays all database links, including scheduled, PUBLIC, and private.

5. Select the database link that connects to the target master group and press the **Next** button.

   The database link will use the global database name of the target database.

6. Select the target master group that contains the objects that you want to replicate.

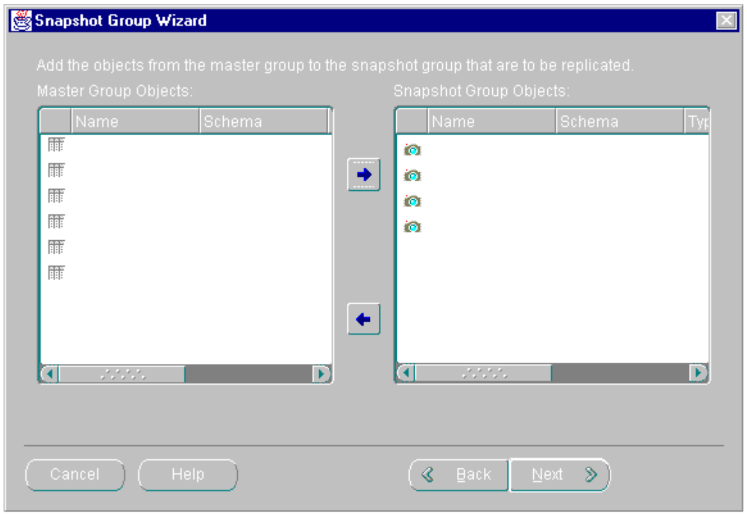
7. If necessary, select the desired **Propagation Mode** radio-button and press the **Next** button.

   - **Asynchronous**: Sometimes known as *store-and-forward data replication*, changes to an updateable snapshot are recorded in a deferred transaction queue and are periodically propagated (controlled by the scheduled link) to the target master site.
   - **Synchronous**: Known also as *real-time replication*, changes made locally to an updateable snapshot are also propagated to the master table during the same transaction; if either update fails, then the whole transaction will fail.

8. Select the master group objects that you want to replicate and press the **Add** button (right arrow).

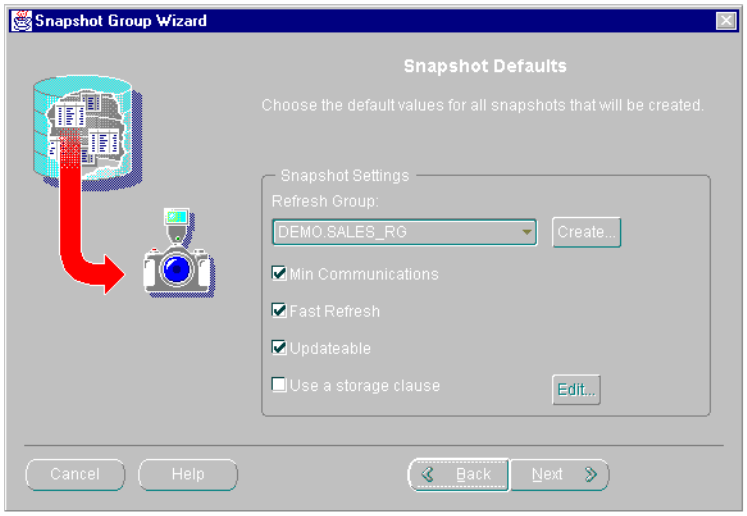
   Press the <SHIFT> key to select a range of objects or press the <CTRL> key to individually select multiple objects.

Figure 5–4 Snapshot Group Wizard Converts Tables to Snapshots



9. Press the **Next** button when you have completed selecting master objects.
10. Modify your snapshot default settings as necessary:

Figure 5–5 Default Settings will be Applied to All Generated Snapshots



Snapshot Defaults

Choose the default values for all snapshots that will be created.

Snapshot Settings

Refresh Group: DEMO.SALES_RG

☑ Min Communications
☑ Fast Refresh
☑ Updateable
☐ Use a storage clause

If you selected multiple tables in step 8 to become snapshots, the following settings will be applied to each of those generated snapshots.

a. Select the refresh group that you want your replicated objects added to from the **Refresh Group** pull-down list. (Press the **Create** button to create a new refresh group.)

b. Enable the following snapshot settings as necessary:

   Min Communications: Reduces the amount of data required to support conflict resolution mechanisms. This feature should only be enabled for Oracle8 and greater databases.

   Fast Refresh: Snapshot will be refreshed using the FAST refresh method (see "Refresh Types" on page on page 3-22 for details).

   Updateable: Allows users to modify the contents of a snapshots and have those changes propagated to the target master table. (See the "Primary Key" section on page on page 3-4 for additional information.)

   Use a storage clause: Allows user to define custom storage settings for the generated snapshots. If enabled, you can modify the extents, space usage, and number of transaction settings. Press the **Edit** button to modify these settings and see "Using a Storage Clause" on page on page 5-13 for detailed information.

11. Press the **Next** button when you have completed modifying your default snapshot settings.

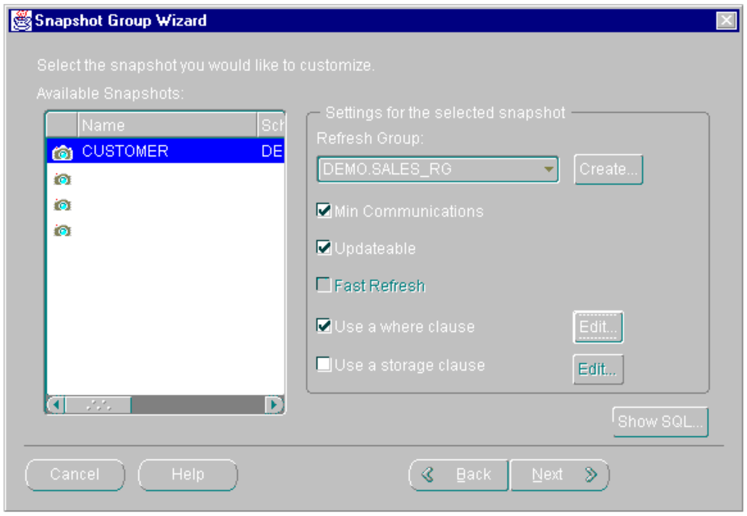
12. Modify your individual snapshot settings if necessary:

    Whereas the previous page of the wizard allowed you to define global snapshot settings, this page allows you to individually modify snapshot settings.

    a. Select the snapshot that you want to modify from the list of **Available Snapshots**.

    b. Select the refresh group that you want your selected snapshot added to from the **Refresh Group** pull-down list. (Press the **Create** button to create a new refresh group.)

Figure 5–6 Settings will be Applied to Selected Snapshot Only


Snapshot Group Wizard

Select the snapshot you would like to customize.

Available Snapshots:

Name	Sch
CUSTOMER	DE

Settings for the selected snapshot

Refresh Group: DEMO.SALES_RG

☑ Min Communications

☑ Updateable

☐ Fast Refresh

☑ Use a where clause

☐ Use a storage clause

c. Enable the following snapshot settings as necessary:

Min Communications: Reduces the amount of data required to support conflict resolution mechanisms. This feature should only be enabled for Oracle8 and greater databases.

Updateable: Allows users to modify the contents of a snapshot and have those changes propagated to the target master table. (See the "Primary Key" section on page on page 3-4 for additional information.)

Fast Refresh: Snapshot will be refreshed using the FAST refresh method (see "Refresh Types" on page on page 3-22 for details).

Use a WHERE clause: Snapshot will contain a data subset of the target master table; this is helpful if you want to replicate only a range of data to the snapshot site. Press the **Edit** button to define the WHERE clause and see "Using a WHERE Clause" on page on page 5-11 for detailed information.

Use a storage clause: Allows user to define custom storage settings for the generated snapshots. If enabled, you can modify the extents, space usage, and number of transaction settings. Press the **Edit** button to modify these settings and see "Using a Storage Clause" on page on page 5-13 for detailed information.

d. Repeat steps 11a — 11c to modify additional snapshots.

Minimizing Data Propagation

The **Minimize Communication** setting lets you determine how much data snapshot sites must transfer to perform conflict detection for an updateable snapshot (primary key snapshots only). When you use the default setting, to minimize communication, Oracle propagates only the new values for updated columns plus the old values of the primary key and the columns in each updated column group.

Note: The default setting, to minimize communication, is valid only for Oracle8 and greater databases. When you base an updateable snapshot on a master table in an Oracle7 release 7.3 database, you must disable the Minimize Communication setting. When disabled, Oracle propagates the old and new values of all columns in a row when any column in the row is updated. This is the behavior expected by Oracle7 release 7.3.

Additional Information: See "Minimizing Data Propagation for Update Conflict Resolution".

13. Press the **Next** button when you have completed modifying the settings of your individual snapshots.
14. Press the **Finish** button to complete the Snapshot Group Wizard.
15. Press the **OK** button to confirm your settings.

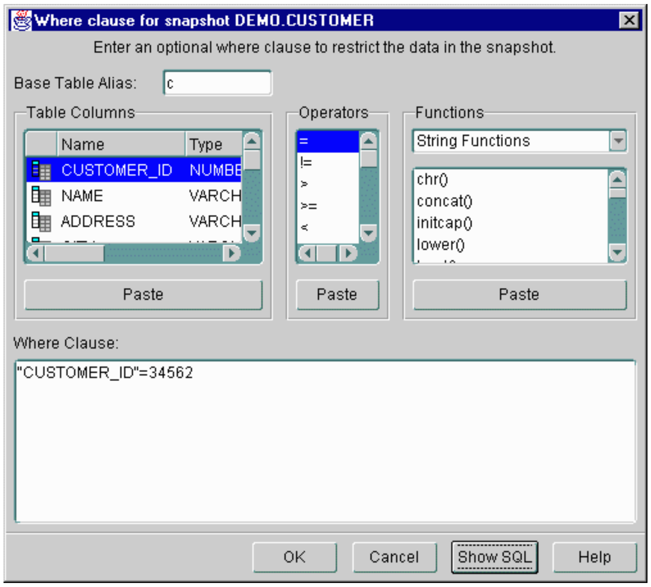
    Replication Manager will now build your snapshot group with the objects and settings that you defined using the Create Snapshot wizard.

Using a WHERE Clause

As described in Chapter 3, "Snapshot Concepts & Architecture", snapshots may contain data subsets of a target master table (see "Data Subsetting with Snapshots" on page on page 3-7 for additional information). The Using a WHERE Clause feature allows you to define the data subset for an individual snapshot.

If you enable the **Using a WHERE Clause** checkbox in step 12 of the "Creating Snapshot Groups" section (page on page 5-9) and press the **Edit** button, then you will see the screen illustrated in Figure 5–7.

Figure 5–7 Define WHERE Clause Screen



"CUSTOMER_ID"=34562

There are two methods of defining your WHERE clause:

- Manually type the WHERE clause
- Use the **Table Columns**, **Operators**, and **Functions** elements to build the WHERE clause

Manually Define WHERE Clause

Complete the following steps to manually define your WHERE clause:

1. Type the WHERE clause using standard SQL in the **Where Clause** field.
2. Press the **OK** button when you have completed defining your WHERE clause.

Use Elements to Define WHERE Clause

Complete the following steps to define your WHERE clause:

1. Select the elements (Table Columns, Operators, and Functions) that you want to use in your WHERE clause and press the appropriate **Paste** button to add it to the generated WHERE clause.

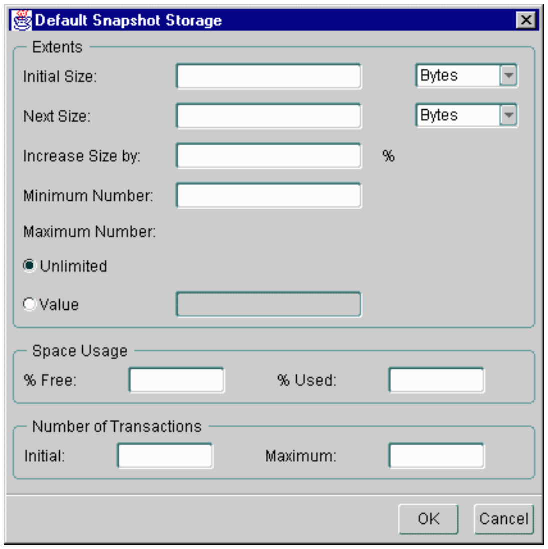
As you add elements, you will see them appear in the **Where Clause** field below the lists of elements.

2. Press the **OK** button when you have completed defining your WHERE clause.

Using a Storage Clause

If you enable the **Use Storage Clause** checkbox in steps 10 or 12 of the "Creating Snapshot Groups" section (page on page 5-9) and press the **Edit** button, then you will see the Snapshot Storage screen illustrated in Figure 5–8. Use this screen to modify the storage characteristics of your snapshots(s).

Figure 5–8 Snapshot Storage Screen



1. Modify the storage settings as necessary.

   For detailed information, see the references mentioned in the following descriptions:

   - **Extents**: The settings in the Extents object group define the values in the Storage clause of your CREATE SNAPSHOT statement. See “Storage Clause” in the *Oracle8i SQL Reference* book for a detailed description of the parameters in the Extents object group. See the *Oracle8i Tuning* book for additional information about these settings.

- **Space Usage**: These parameters control the operation of data blocks. See “physical_attribute_clause” in the *Oracle8i SQL Reference* book for detailed information.
- **Number of Transactions**: These parameters control the amount of initial transactions allocated and the number of concurrent transactions. See “physical_attribute_clause” in the *Oracle8i SQL Reference* book for detailed information.

2. Press the **OK** button when you have completed your storage modifications.

Managing Snapshot Groups

After you have created your snapshot groups, you may need to add, alter, or delete replicated objects. You may also need to alter the characteristics of your snapshot group and, at times, you may even need to delete a snapshot group. The following sections will describe how to use Oracle’s Replication Manager to manage your snapshot groups.

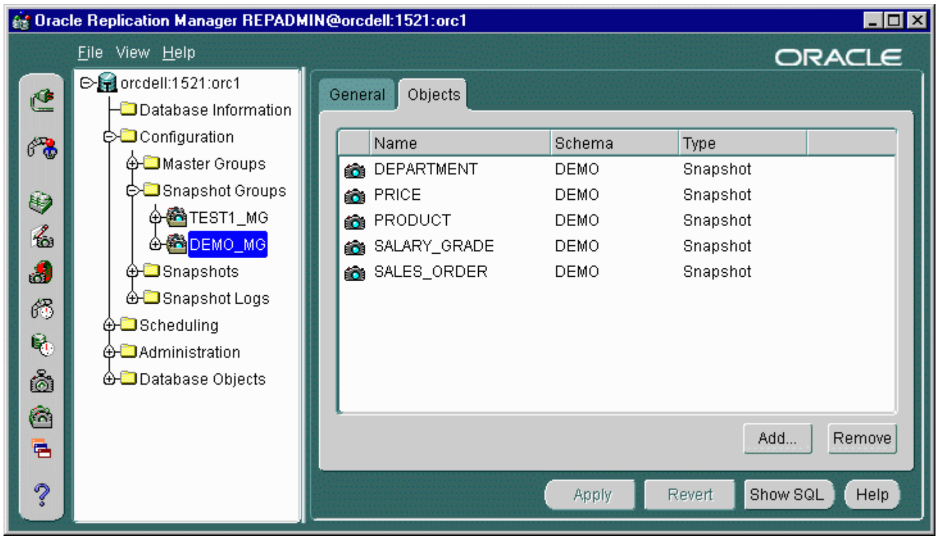
Before you begin working with your snapshot groups, make sure that you are connected to the remote snapshot site. If you are not connected, you may need to disconnect and relogin to the target database.

Editing a Snapshot Group

To edit a snapshot group:

1. Expand the database node that contains the snapshot group that you want to edit.
2. Expand the **Configuration** node.
3. Expand the **Snapshot Groups** node.

Figure 5–9 "Drill down" until you find the target snapshot group.



Name	Schema	Type
DEPARTMENT	DEMO	Snapshot
PRICE	DEMO	Snapshot
PRODUCT	DEMO	Snapshot
SALARY_GRADE	DEMO	Snapshot
SALES_ORDER	DEMO	Snapshot

4. Select the snapshot group that you want to alter.

   You will now be able to modify the snapshot group settings and add or remove objects to or from your snapshot group. Please complete the steps in one of the following sections to modify your snapshot group: "Alter Snapshot Group Settings", "Adding Objects to a Snapshot Group", or "Deleting Objects from a Snapshot Group".

Alter Snapshot Group Settings

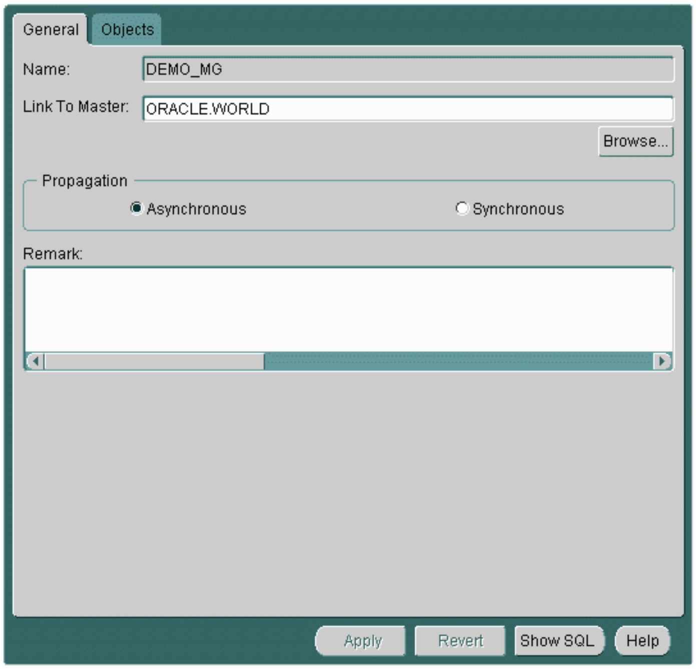
The General tab of the Snapshot Group property sheet allows you to change the propagation type and the target master group database. Complete the following steps to modify these settings:

1. Verify that you have selected the target snapshot group (see "Editing a Snapshot Group" on page on page 5-14 for instructions).
2. Verify that the **General** tab is selected.
3. If necessary, enter the new TNSNAMES alias to the new target master group database in the **Link to Master** field (you need to make sure to enter a target that has a database link pointing to it from the current snapshot database).

If you are not sure what TNSNAMES alias to enter, press the **Browse** button to select from a list of existing database links.

Changing the **Link to Master** database is very useful if the current target database is no longer available and you need to point your snapshot database to an alternate master database.

Figure 5–10 The General Tab of the Snapshot Group Property Sheet


General | Objects

Name: DEMO_MG

Link To Master: ORACLE.WORLD

Propagation: [✓ Asynchronous] Synchronous

Remark:

4. If necessary, alter the propagation type by selecting the appropriate radio-button:

   - **Asynchronous**: Sometimes known as *store-and-forward data replication*, changes to an updateable snapshot are recorded in a deferred transaction queue and are periodically propagated (controlled by the scheduled link) to the target master site.
   - **Synchronous**: Known also as *real-time replication*, changes made locally to an updateable snapshot are also propagated to the master table during the same transaction; if either update fails, then the whole transaction will fail.

5. Press the **Apply** button to save your settings.

Adding Objects to a Snapshot Group

To add objects to a snapshot group:

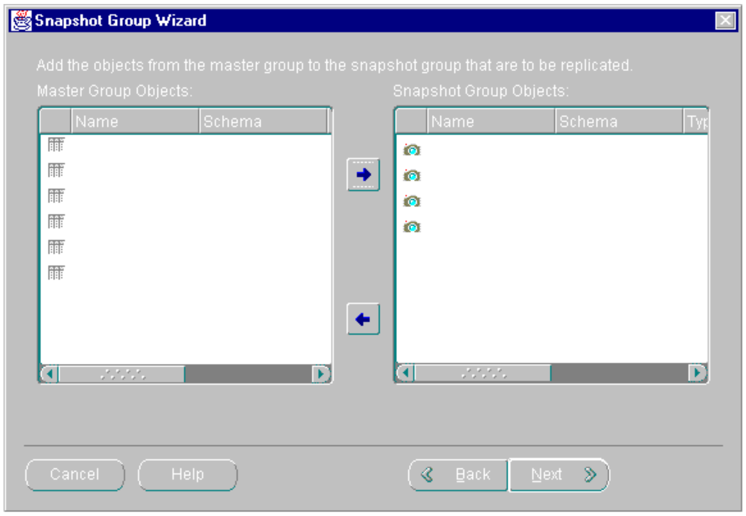
1. Verify that you have selected the target snapshot group (see "Editing a Snapshot Group" on page on page 5-14 for instructions).
2. Press the **Objects** tab.

   You will see a list of all objects that are currently contained in the selected snapshot group (the schema, name, and type properties will be displayed for each object).
3. Press the **Add** button to add an object or objects to the selected snapshot group.

   The **Snapshot Group Edit Wizard** will appear.
4. Select the master group objects that you want to replicate and press the **Add** button (right arrow).

   Press the <SHIFT> key to select a range of objects or press the <CTRL> key to individually select multiple objects.

Figure 5–11 Snapshot Group Wizard Converts Tables to Snapshots


Snapshot Group Wizard

Add the objects from the master group to the snapshot group that are to be replicated.

Master Group Objects:

Snapshot Group Objects:

5. Press the **Next** button when you have completed selecting master objects.
6. Modify your snapshot default settings as necessary:

If you selected multiple tables in step 4 to become snapshots, the following settings will be applied to each of those generated snapshots.

a. Select the refresh group that you want your replicated objects added to from the **Refresh Group** pull-down list. (Press the **Create** button to create a new refresh group.)

b. Enable the following snapshot settings as necessary:

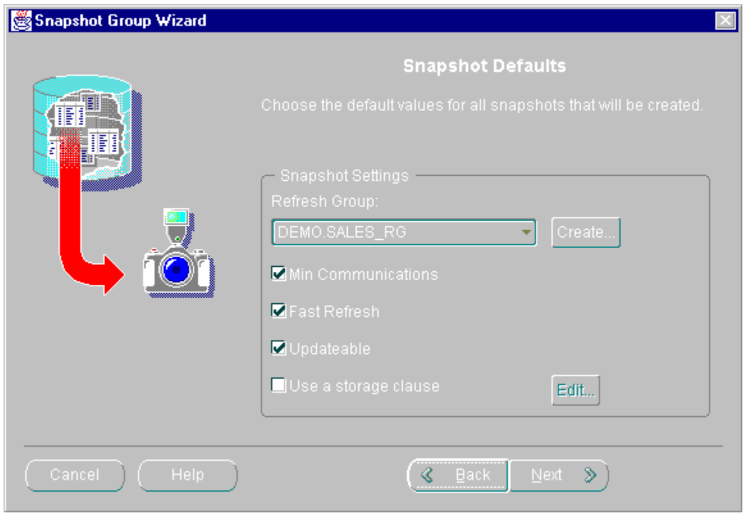
Min Communications: Reduces the amount of data required to support conflict resolution mechanisms. This feature should only be enabled for Oracle8 and greater databases.

Fast Refresh: Snapshot will be refreshed using the FAST refresh method (see "Refresh Types" on page on page 3-22 for details).

Updateable: Allows users to modify the contents of a snapshot and have those changes propagated to the target master table. (See the "Primary Key" section on page on page 3-4 for additional information.)

Use a storage clause: Allows user to define custom storage settings for the generated snapshot. If enabled, you can modify the extents, space usage, and number of transaction settings. Press the **Edit** button to modify these settings and see "Using a Storage Clause" on page on page 5-13 for detailed information.

Figure 5–12 Default Settings will be Applied to All Generated Snapshots



Snapshot Defaults

Choose the default values for all snapshots that will be created.

Snapshot Settings

Refresh Group: DEMO.SALES_RG

☑ Min Communications

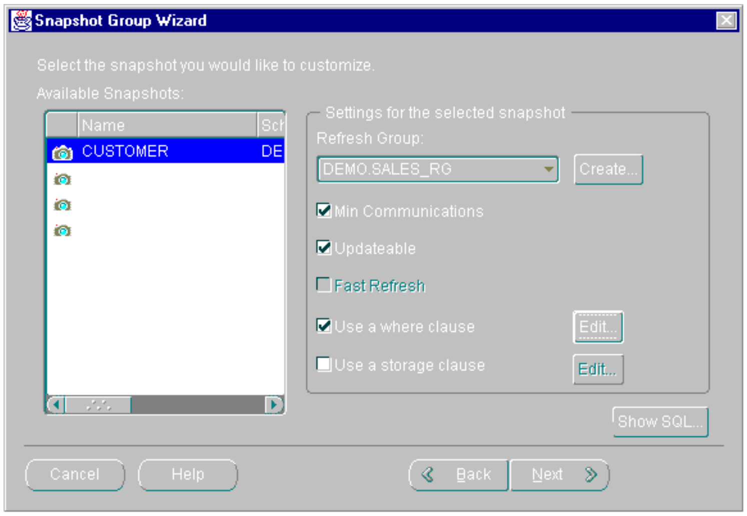
☑ Fast Refresh

☑ Updateable

☐ Use a storage clause

7. Press the **Next** button when you have completed modifying your default snapshot settings.
8. Modify your individual snapshot settings if necessary:

Figure 5–13 Settings will be Applied to Selected Snapshot Only


Snapshot Group Wizard

Select the snapshot you would like to customize.

Available Snapshots:

Name	Sch
CUSTOMER	DE

Settings for the selected snapshot

Refresh Group: DEMO.SALES_RG

[✓] Min Communications

[✓] Updateable

[ ] Fast Refresh

[✓] Use a where clause

[ ] Use a storage clause

Whereas the previous page of the wizard allowed you to define global snapshot settings, this page allows you to individually modify snapshot settings.

a. Select the snapshot that you want to modify from the list of **Available Snapshots**.

b. Select the refresh group that you want your selected snapshot added to from the **Refresh Group** pull-down list. (Press the **Create** button to create a new refresh group.)

c. Enable the following snapshot settings as necessary:

   Min Communications: Reduces the amount of data required to support conflict resolution mechanisms. This feature should only be enabled for Oracle8 and greater databases.

   Updateable: Allows users to modify the contents of a snapshot and have those changes propagated to the target master table. (See the "Primary Key" section on page on page 3-4 for additional information.)

Fast Refresh: Snapshot will be refreshed using the FAST refresh method (see "Refresh Types" on page on page 3-22 for details).

Use a WHERE clause: Snapshot will contain a data subset of the target master table; this is helpful if you want to replicate only a range of data to the snapshot site. Press the **Edit** button to define the WHERE clause and see "Using a WHERE Clause" on page on page 5-11 for detailed information.

Use a storage clause: Allows user to define custom storage settings for the generated snapshots. If enabled, you can modify the extents, space usage, and number of transaction settings. Press the **Edit** button to modify these settings and see "Using a Storage Clause" on page on page 5-13 for detailed information.

d. Repeat steps 11a — 11c to modify additional snapshots.

9. Press the **Next** button when you have completed modifying the settings of your individual snapshots.

10. Press the **Finish** button to complete the Snapshot Group Wizard.

11. Press the **OK** button to confirm your settings.

Replication Manager will now add the selected objects to your snapshot group that you selected using the Edit Snapshot Group wizard.

Altering Objects in a Snapshot Group

To alter the definition of a replication object in a snapshot group, you should always use Replication Manager (or an equivalent API call). Use of Enterprise Manager or a SQL DDL command (for example, ALTER TABLE) to alter an object in a replicated environment can create inconsistencies.

Note: Local customization of individual replicas at a snapshot site is outside the scope of Oracle’s advanced replication facility. As a replication administrator, you must ensure that local customizations do not interfere with any global customizations done with Replication Manager.

You cannot alter the definition of nonsnapshot objects in a snapshot group. For more information about altering individual snapshots in a snapshot group, see "Altering a Snapshot" on page 5-27.

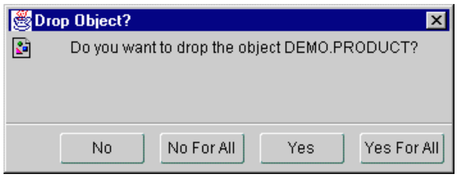
Deleting Objects from a Snapshot Group

To remove objects from a snapshot group:

1. Verify that you have selected the target snapshot group (see "Managing Snapshot Groups" on page on page 5-14 for instructions).
2. Press the **Objects** tab.
3. Select the object or objects that you want to delete.

   Press the <SHIFT> key to select a range of objects or press the <CTRL> key to individually select multiple objects.
4. Press the **Remove** button.
5. The Drop Object dialog box will appear.

Figure 5–14 Drop Object Dialog Box


Drop Object?

Do you want to drop the object DEMO.PRODUCT?

   Press the button that meets your needs:

   - **No**: The master object that corresponds with the displayed snapshot object will not be dropped.
   - **No for All**: No master objects will be dropped (helpful if you have selected multiple objects to be removed).
   - **Yes**: The master object that corresponds with the displayed snapshot object will be dropped.
   - **Yes for All**: All master objects that corresponds with the snapshot object(s) that you have chosen will be dropped (helpful if you have selected multiple objects to be removed).
6. After the Drop Object dialog box disappears, press the **Apply** button.

Note: When you drop an object from a snapshot group, Replication Manager automatically removes all corresponding system-generated objects that were necessary to support the object.

Regenerating Replication Support for an Updateable Snapshot

If it ever becomes necessary to regenerate replication support, Replication Manager allows you to do this quickly and easily. To generate support for an updateable snapshot:

1. Expand the database node that contains the snapshot object that you want to generate replication support for.
2. Expand the **Configuration** node.
3. Expand the **Snapshot Groups** node.
4. Expand the snapshot group that contains the updateable snapshot that you want to generate replication support for.
5. Expand the **Replicated Objects** node.
6. Expand the **Snapshots** node.
7. Expand the schema that contains the target updateable snapshot.
8. Select the updateable snapshot that you want to generate replication support for.

   The snapshot property sheet will appear in the right pane of the Replication Manager user interface.
9. Press the **Generate** button.

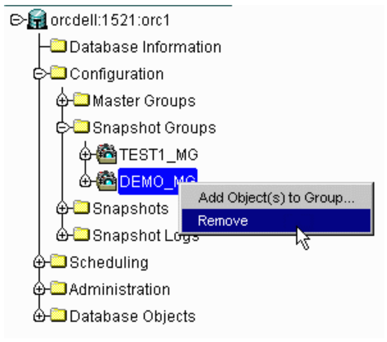
API Equivalent: DBMS_REPCAT.GENERATE_SNAPSHOT_SUPPORT

Deleting a Snapshot Group

To delete a snapshot group:

1. Expand the database node that contains the snapshot group that you want to delete.
2. Expand the **Configuration** node.
3. Expand the **Snapshot Groups** node.

Figure 5–15 Right-click on Snapshot Group and Select Remove



4. Right-click on the snapshot group that you want to delete and select **Remove**.
5. Press the **OK** button to confirm that you want to delete the selected snapshot group.

Note: When you drop a snapshot group, you can also decide whether to drop the group's objects from the database. If you choose not to drop a group's objects when you drop the group, the objects remain in the database at the snapshot site. Snapshots that remain after dropping the corresponding snapshot group persist as ungrouped snapshots and will be refreshed as long as they remain in a refresh group. However, Oracle does not propagate any changes made to the updateable snapshot to the master site.

API Equivalent: DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP

Managing Individual Snapshots

A snapshot is a local copy of table data that originates from one or more remote master tables. Applications can query the data in a read-only table snapshot, but cannot insert, update, or delete rows in the snapshot. However, applications can query, insert, update, or delete the rows in an updateable snapshot.

Oracle's data replication facility supports independent table snapshots as well as snapshots that are part of a snapshot group. Consider the following issues when deciding whether to create a new snapshot as part of a snapshot group:

- A snapshot group can contain simple, read-only and updateable snapshots.
- A snapshot group does not support complex read-only snapshots. Oracle creates all complex read-only snapshots independent of any snapshot group.

Replication Manager has many features that help you create and manage snapshots in an advanced replication environment. The following sections explain more about managing snapshots.

Creating a Snapshot

Before you create a new snapshot, make sure that the following preliminary tasks have been completed.

- Create the necessary Snapshot Logs to support FAST refreshable snapshots (see "Create a Snapshot Log" on page on page 3-30 for details).
- Create a refresh group that snapshots can use for automatic refreshes.

To create a read-only snapshot independent of a snapshot group:

1. Select the database in the left pane of the Replication Manager user interface where you want to create a snapshot.

   If you don't see the target database, you may need to disconnect and relogin to the target database.
2. Select **Create** from the **File** menu.
3. Select **Snapshot**.
4. Enter the name of your snapshot in the **Name** field.
5. Select the schema from the **Schema** pull-down list that will store the new snapshot.
6. If necessary, select a specific tablespace for your new snapshot from the **Tablespace** pull-down list.
7. If necessary, modify the following snapshot **Options** according to the following:
   - **Updateable:** If enabled, indicates that the current snapshot is updateable (changes to the snapshot will be propagated to the target master site).
   - **Logging**: If enabled, specifies that the creation of the snapshot and any related indexes, partitions, or LOG storage characteristics will be logged in the redo log file.

- **Cache**: If enabled, specifies that Oracle pre-allocates and retains LOB data values in memory for faster access.
- **Snapshot Index:** This object group allows you to modify the storage characteristics and tablespace location for the index that is generated for you new snapshot.

  Edit Storage: Press the **Edit Storage** button to modify the storage characteristics of the index that will be created for your new snapshot. See "Using a Storage Clause" on page on page 5-13 for more information.

  Tablespace: Select the tablespace that will hold the generated index from the **Tablespace** pull-down list.

8. Enter the SELECT statement for you snapshot in the **Snapshot Subquery** field. For example, you might enter:

   ```
   SELECT * FROM scott.emp WHERE empno = 7899
   ```

   You do not need to enter a semi-colon at the end of your SELECT statement. To see the entire CREATE SNAPSHOT DDL, press the **Show SQL** button.

   Your SELECT statement can be unfiltered, contain a simple WHERE clause, or contain several nested subqueries. For more information about subquery subsetting, see "Data Subsetting with Snapshots" on page on page 3-7.

9. Press the **Refresh** tab.
10. Selected the refresh method from the **Refresh Type** pull-down list that you want to use during the refresh of this snapshot (see "Refresh Types" on page on page 3-22 for more information):

    - **FORCE**: A FAST refresh will be tried first and if it fails, a COMPLETE refresh will be performed.
    - **COMPLETE**: All of the data that satisfies the snapshot's defining query will be propagated to the snapshot.
    - **FAST**: Only data that has changed will be propagated to the snapshot. If a fast refresh fails, then the snapshot will not be refreshed.

11. Enable the **Row ID** checkbox to refresh your snapshot based on the Row ID of the changed data.

    The Row ID checkbox should only be enabled when interoperating with Oracle 7.3 and earlier databases. See "Snapshot Log" on page 3-15 for more information.

12. If your individual snapshot will NOT be part of a refresh group, you will need to specify the start date and refresh interval for you your individual snapshot. If your snapshot will be part of a refresh group, skip to step 13 and be sure to follow the steps described in the "Adding Snapshots to a Refresh Group" section on page 5-29.

    a. Press the **Set** button next to the **Start Date** field to specify when the first refresh should occur.

    b. Select the appropriate date (or manually enter a date expression in the **Date Expression** field) and press the OK button.

    c. Press the **Set** button next to the **Next Date** field to specify when the next refresh should occur.

    d. Select the appropriate date (or manually enter a date expression in the **Date Expression** field) and press the **OK** button.

13. Enter the name of the master rollback segment in the **Master** field.

14. Select the local rollback segment from the **Local** pull-down list.

15. Press the **Storage** tab.

16. Modify the contents of the **Storage** tab as needed (see "Using a Storage Clause" on page on page 5-13 for additional information).

17. Press the **Create** button to complete creating your individual snapshot.

Datatype Considerations for Snapshots

Oracle supports snapshots of master table columns that use the following datatypes: NUMBER, DATE, VARCHAR2, CHAR, NVARCHAR2, NCHAR, RAW, ROWID.

Oracle also supports snapshots of master table columns that use the following large object types: binary LOBs (BLOBs), character LOBs (CLOBs), and national character LOBs (NCLOBs). However, you cannot reference LOB columns in a WHERE clause of a snapshot's defining query. The deferred and synchronous remote procedure call mechanism used for replication propagates only the piece-wise changes to the supported LOB datatypes when piece-wise updates and appends are applied these LOB columns.

Note: Oracle8*i* does not support replication of LOB datatypes in replication environments where some sites are running Oracle7 release 7.3.

Oracle does not support the replication of columns that use the LONG datatype. Oracle simply omits the data in LONG columns from snapshots.

Oracle also does not support user-defined object types and external or file-based LOBs (BFILEs). Attempts to configure snapshots containing columns of these datatypes returns an error message.

Altering a Snapshot

The following sections explain how to alter an ungrouped snapshot in an advanced replication environment.

Editing a Snapshot's Storage Settings

To edit a snapshot's storage settings:

1. Expand the database node that contains the snapshot that you want to alter.
2. Expand the **Configuration** node.
3. Expand the **Snapshots** node.
4. Expand the schema that contains the snapshot that you want to alter.
5. Select the snapshot to alter.

   The snapshot property sheet will appear in the right pane of the Replication Manager user interface.
6. Modify the settings on the **General**, **Refresh**, **Storage**, and **Master Info** pages as necessary.
7. Press the **Apply** button.

Manipulating a Snapshot's Base Table

Do not manipulate, modify, add to, or subtract from, the data in the base table of a snapshot. You can declare integrity constraints, such as referential or uniqueness constraints, for the base table of a snapshot. However, such constraints must be configured for deferred constraint checking.

You can also define PL/SQL triggers on the base table of a snapshot. However, such triggers must be coded so that they do **not** fire during snapshot refresh. Triggers that fire during snapshot refresh may generate unexpected results.

Additional Information: See "Triggers and Replication" on page 8-36.

Altering a Snapshot Definition

Never use Enterprise Manager or a SQL DDL command (for example, ALTER TABLE) to alter a snapshot definition. To alter the definition of a snapshot, drop the snapshot and then re-create it.

Note: Local customization of individual replicas at snapshot sites is outside the scope of Oracle's advanced replication facility. As a replication administrator, you must ensure that local customizations do not interfere with any global customizations done with Replication Manager.

Deleting a Snapshot

To remove a snapshot from a snapshot group:

1. Expand the database node that contains the snapshot that you want to delete.
2. Expand the **Configuration** node.
3. Expand the **Snapshots** node.
4. Expand the schema node that contains the snapshot that you want to delete.
5. Right-click on the target snapshot and select **Remove**.

Note: When you drop a snapshot from a snapshot group, Replication Manager automatically removes all corresponding system generated objects that were necessary to support the snapshot.

Managing Refresh Groups

To preserve referential integrity and transactional consistency among the snapshots of several related master tables, Oracle organizes and refreshes individual snapshots as part of a refresh group. After refreshing all of the snapshots in a refresh group, the data of all snapshots in the group corresponds to the same transactionally consistent point-in-time. Replication Manager has many features that help you create and manage refresh groups in a replication environment. The following sections explain more about managing refresh groups.

Additional Information: See "Refresh Groups" on page 3-21 for more information about refresh groups.

Creating a Refresh Group

To create a new refresh group for a snapshot site:

1. Select the database in the left pane of the Replication Manager user interface where you want to create the refresh group.
2. Select **Create** from the **File** menu
3. Select **Refresh Group** from the sub-menu.

   You can optionally press the **Create Refresh Group** button on the toolbar.

The **Create Refresh Group** property sheet has two pages: **General**, and **Snapshots**.

- Use the **General** page to specify the name, owner, and refresh settings for the group.
- Use the **Snapshots** page to add snapshots to the refresh group.

Note: To refresh a snapshot, the user account of the database link used by the snapshot must have SELECT privileges on the master base table and, for fast refreshes, on the corresponding snapshot log.

Adding Snapshots to a Refresh Group

To add one or more snapshots to a refresh group:

1. Expand the database node that contains the refresh group that you want to add a snapshot to.
2. Expand the **Scheduling** node.
3. Expand the **Refresh Groups** node.
4. Expand the schema node that contains the refresh group.
5. Right-click on the target refresh group and select **Add Snapshot(s) to Group**.
6. Select the schema that contains the snapshot you want to add from the **Schema** pull-down list.
7. Select the snapshot that you want to add from the **Available Snapshots** list.

   Press the <SHIFT> key to select a range of snapshots or press the <CTRL> key to individually select multiple snapshots.

8. Press the **Down Arrow** to add the selected objects to the **Selected Snapshots** list.
9. Press the **OK** button when you have completed selecting the snapshot(s) to add to the refresh group.

Deleting Snapshots from a Refresh Group

To remove one or more snapshots from a refresh group:

1. Expand the database node that contains the refresh group that you want to remove a snapshot from.
2. Expand the **Scheduling** node.
3. Expand the **Refresh Groups** node.
4. Expand the schema node that contains the refresh group.
5. Select the target refresh group that contains the snapshot that you want to remove.

   The refresh group property sheet will appear in the right pane of the Replication Manager user interface.
6. Press the **Snapshots** tab.
7. Select the snapshot(s) that you want to remove.

   Press the <SHIFT> key to select a range of snapshots or press the <CTRL> key to individually select multiple snapshots.
8. Press the **Remove** button.
9. Press the **Apply** button.

Changing Refresh Settings for a Snapshot Group

To edit a snapshot group's refresh settings:

1. Expand the database node that contains the refresh group that you want to alter.
2. Expand the **Scheduling** node.
3. Expand the **Refresh Groups** node.
4. Expand the schema node that contains the refresh group.

5. Select the target refresh group that you want to alter.

   The refresh group property sheet will appear in the right pane of the Replication Manager user interface.

6. Modify the scheduling characteristics in the Refresh object group.
7. Press the **Apply** button when you have made all of the necessary refresh scheduling modifications.

Manually Refreshing a Group of Snapshots

To force the immediate refresh of a refresh group:

1. Expand the database node that contains the refresh group that you want to manually refresh.
2. Expand the **Scheduling** node.
3. Expand the **Refresh Groups** node.
4. Expand the schema node that contains the refresh group.
5. Select the target refresh group that contains the snapshot that you want to refresh.

   The refresh group property sheet will appear in the right pane of the Replication Manager user interface.

6. Press the **Refresh** button.

Deleting a Refresh Group

To delete a refresh group:

1. Expand the database node that contains the refresh group that you want to delete.
2. Expand the **Scheduling** node.
3. Expand the **Refresh Groups** node.
4. Expand the schema node that contains the refresh group.
5. Right-click on the target refresh group that you want to delete and select **Remove**.

Note: After you drop a refresh group, Oracle no longer refreshes the group's snapshots automatically. To refresh the snapshots, you must add them to another snapshot refresh group or refresh them manually.

Other Snapshot Site Administration Issues

The preceding sections of this chapter explained the most commonly performed administrative procedures that involve snapshot sites. For additional information on less commonly performed administrative procedures for snapshot sites, see "Advanced Management of Master and Snapshot Groups".

Data Dictionary Views

In addition to using Replication Manager to view information about an snapshot site in an advanced replication environment, you can also query the following data dictionary views:

At the Snapshot Site:

- DBA_SNAPSHOTS
- DBA_REFRESH
- DBA_REFRESH_CHILDREN

At the Master Site:

- DBA_REGISTERED_SNAPSHOT
- DBA_SNAPSHOT_LOGS

6 Conflict Resolution

This chapter covers the following topics:

Note: This chapter has examples of how to use the Oracle Replication Manager tool to manage conflict resolution in an advanced replication system. Each section also lists equivalent replication management API procedures for your reference. For complete information about Oracle's replication management API, see the *Oracle8i Replication API Reference* book.

Introduction to Replication Conflicts

Replication conflicts can occur in an advanced replication environment that permits concurrent updates to the same data at multiple sites. For example, when two transactions originating from different sites update the same row at nearly the same time, a conflict can occur. When you configure an advanced replication environment, you must consider whether replication conflicts can occur. If your system design permits replication conflicts and a conflict occurs, the system data does not converge until the conflict is resolved in some way.

In general, your first choice should always be to design a replicated environment that avoids the possibility of conflicts. Using several techniques, most system designs can avoid conflicts in all or a large percentage of the data that you replicate. However, many applications require that some percentage of data be updateable at multiple sites. If this is the case, you must then address the possibility of replication conflicts.

The next few sections introduce general information about replication conflicts, how to design an advanced replication system with replication conflicts in mind, how you can avoid replication conflicts in your replicated system design, and how Oracle can detect and resolve conflicts in designs where conflict avoidance is not possible.

Understanding Your Data and Application Requirements

When you design any type of database application and supporting database, it is critical that you understand the requirements of the application before you begin to build the database or the application itself. For example, each application should be modular, with clearly defined functional boundaries and dependencies (for example, order-entry, shipping, billing). Furthermore, you should normalize supporting database data to reduce the amount of hidden dependencies between modules in the application system.

In addition to basic database design practices, there are additional requirements that you must investigate when building a database that operates in an advanced replication environment. Start by considering the general requirements of applications that will work with replicated data. For example, some applications might work fine with basic read-only table snapshots, and as a result, can avoid the possibility of replication conflicts altogether. Other applications might require that most of the replicated data be read-only and a small fraction of the data (for example, one or two tables or even one or two columns in a specific table) be updateable at all replication sites. In this case, you must determine how to resolve replication conflicts when they occur so that the integrity of replicated data remains intact.

Some Examples

To better understand how to design a replicated database system with conflicts in mind, consider the following environments where conflict detection and resolution is feasible in some cases but not possible in others:

- Conflict resolution is often not possible in reservation systems where multiple bookings for the same item are not allowed. For example, when reserving specific seats for a concert, different agents accessing different replicas of the reservation system cannot book the same seat for multiple customers because there is no way to resolve such a conflict.
- Conflict resolution is often possible in customer management systems. For example, salespeople can maintain customer address information at different databases in a replicated environment. Should a conflict arise, the system can resolve the conflicting updates by applying the most recent update to a record.

Types of Replication Conflicts

Advanced Replication includes facilities for detecting and resolving three types of conflicts: update conflicts, uniqueness conflicts, and delete conflicts.

Update Conflicts

An *update conflict* occurs when the replication of an update to a row conflicts with another update to the same row. Update conflicts can happen when two transactions, originating from different sites, update the same row at nearly the same time.

Uniqueness Conflicts

A *uniqueness conflict* occurs when the replication of a row attempts to violate entity integrity (a PRIMARY KEY or UNIQUE constraint). For example, consider what happens when two transactions originate from two different sites each inserting a row into a respective table replica with the same primary key value. In this case, replication of the transactions causes a uniqueness conflict.

Delete Conflicts

A *delete conflict* occurs when two transactions originate from different sites, with one transaction deleting a row that the other transaction updates or deletes.

Avoiding Conflicts

If application requirements permit, you should first design an advanced replication system that avoids the possibility of replication conflicts altogether. The next few sections briefly suggest several techniques that you can use to avoid some or all replication conflicts.

Primary Site and Dynamic Site Ownership Data Models

One way you can avoid the possibility of replication conflicts is to limit the number of sites in the system with simultaneous update access to the replicated data. Two replicated data ownership models support this approach: primary site ownership and dynamic site ownership.

Primary Site Ownership Primary ownership is the replicated data model that basic read-only replication environments support. Primary ownership prevents all replication conflicts, because only a single server permits update access to a set of replicated data.

Rather than control the ownership of data at the table level, applications can employ horizontal and vertical partitioning to establish more granular static ownership of data. For example, applications might have update access to specific columns or rows in a replicated table on a site-by-site basis.

Additional Information: For more information about Oracle’s basic, read-only replication features, see Chapter 2.

Dynamic Site Ownership The dynamic ownership replicated data model is less restrictive than primary site ownership. With dynamic ownership, capability to update a data replica moves from site to site, still ensuring that only one site provides update access to specific data at any given point in time. A workflow system clearly illustrates the concept of a dynamic ownership. For example, related departmental applications can read the status code of a product order, for example, ENTERABLE, SHIPPABLE, BILLABLE, to determine when they can and cannot update the order.

Additional Information: For more information about using dynamic ownership data models, see "Using Dynamic Ownership Conflict Avoidance" on page 8-30.

Avoiding Specific Types of Conflicts

When both primary site ownership and dynamic ownership data models are too restrictive for your application requirements, you must use a shared ownership data model. Even so, typically you can use some simple strategies to avoid specific types of conflicts.

Avoiding Uniqueness Conflicts It is quite easy to configure an advanced replication environment to prevent the possibility of uniqueness conflicts. For example, you can create replica sequences at each site so that each sequence generates a mutually exclusive set of sequence numbers; however, this solution can become problematic as the number of sites increase or the number of entries in the replicated table grows. Alternatively, you can allow each site's replica sequences to use the full range of sequence values and include a unique site identifier as part of a composite primary key.

Avoiding Delete Conflicts Delete conflicts should always be avoided in all replicated data environments. In general, applications that operate within an asynchronous, shared ownership data model should not delete rows using DELETE statements. Instead, applications can mark rows for deletion and then configure the system to periodically purge logically deleted rows using procedural replication.

Avoiding Update Conflicts After trying to eliminate the possibility of uniqueness and delete conflicts in an advanced replication system, you should also try to limit the number of update conflicts that are possible. However, in a shared ownership data model, update conflicts cannot be avoided in all cases. If you cannot avoid all update conflicts, you must understand exactly what types of replication conflicts are possible and then configure the system to resolve conflicts when they occur.

Conflict Detection at Master Sites

Each master site in an advanced replication system automatically detects and resolves replication conflicts when they occur. For example, when a master site pushes its deferred transaction queue to another master site in the system, the remote procedures being called at the receiving site detect replication conflicts automatically, if any.

When a snapshot site pushes deferred transactions to its corresponding master site, the receiving master site performs conflict detection and resolution. A snapshot site refreshes its data by performing snapshot refreshes. The refresh mechanism ensures that, upon completion, the data at a snapshot is the same as the data at the corresponding master, including the results of any conflict resolution; therefore, it is not necessary for a snapshot site to perform work to detect or resolve replication conflicts.

How Oracle Detects Different Types of Conflicts

The receiving master site in an advanced replication system detects update, uniqueness, and delete conflicts as follows:

- The receiving site detects an update conflict if there is any difference between the old values of the replicated row (the value before the modification) and the current values of the same row at the receiving site.
- The receiving site detects a uniqueness conflict if a uniqueness constraint violation occurs during an INSERT or UPDATE of a replicated row.
- The receiving site detects a delete conflict if it cannot find a row for an UPDATE or DELETE statement because the primary key of the row does not exist.

Note: To detect and resolve an update conflict for a row, the propagating site must send a certain amount of data about the new and old versions of the row to the receiving site. For maximum performance, tune the amount of data that Oracle uses to support update conflict detection and resolution. For more information, see "Minimizing Data Propagation for Update Conflict Resolution" on page 6-42.

Identifying Rows During Conflict Detection

To detect replication conflicts accurately, Oracle must be able to uniquely identify and match corresponding rows at different sites during data replication. Typically, Oracle’s advanced replication facility uses the primary key of a table to uniquely identify rows in the table. When a table does not have a primary key, you must designate an alternate key—a column or set of columns that Oracle can use to identify rows in the table during data replication.

Warning: Do not permit applications to update the identity columns of a table. This ensures that Oracle can identify rows and preserve the integrity of replicated data.

Conflict Resolution

When replication conflicts occur at a receiving master site, you must resolve them to ensure that the data throughout the system eventually converges. *Data convergence* means that all sites managing replicated data will ultimately agree on a set of matching information. If replication conflicts happen and you neglect to resolve them, the replicated data at various sites remains inconsistent. Furthermore, there can be undesirable cascading affects. An inconsistency can create additional conflicts, which create additional inconsistencies, and so on.

If you cannot avoid all types of replication conflicts in your system, you can configure the system to use Oracle's automatic conflict resolution features. The following sections explain more about Oracle's conflict resolution features for each type of replication conflict.

Automatic versus Manual Conflict Resolution

You should always use Oracle's automatic conflict resolution features to resolve conflicts when they occur. When you do not configure automatic conflict resolution for replicated tables, Oracle simply logs conflicts at each site. In this case, you are forced to resolve conflicts manually to preserve the integrity of replicated data. Manual conflict resolution can be challenging to perform. Furthermore, delays in performing manual conflict resolution can leave inconsistencies in the data that can create cascading effects mentioned in the previous section.

Update Conflict Resolution and Column Groups

Oracle uses column groups to detect and resolve update conflicts. A column group is a logical grouping of one or more columns in a replicated table. Every column in a replicated table is part of a single column group. When configuring replicated tables at the master definition site, you can create column groups and then assign columns and corresponding conflict resolution methods to each group.

Ensuring Data Integrity with Multiple Column Groups Having column groups allows you to designate different methods of resolving conflicts for different types of data. For example, numeric data is often suited for an arithmetical resolution method, and character data is often suited for a timestamp resolution method. However, when selecting columns for a column group, it is important to group columns wisely. If two or more columns in a table must remain consistent with respect to each other, place the columns within the same column group to ensure data integrity. For example, if the zip code column in a customer table uses one resolution method while the city column uses a different resolution method, the sites could converge on a zip code that does match the city. Therefore, all components of an address should typically be within a single column group so that conflict resolution is applied to the address as a unit.

Shadow Column Groups By default, every replicated table has a shadow column group. A table's shadow column group contains all columns that are not within a specific column group. You *cannot* assign conflict resolution methods to a table's shadow group. Therefore, make sure to include a column in a column group when conflict resolution is necessary for the column.

Uniqueness Conflict Resolution

In most cases, you should build an advanced replication system and corresponding applications so that uniqueness conflicts are not possible. However, if you cannot avoid uniqueness conflicts, you can assign one or more conflict resolution methods to a PRIMARY KEY or UNIQUE constraint in a replicated table to resolve uniqueness conflicts when they occur. Oracle provides a few prebuilt uniqueness conflict resolution methods. However, you will typically want to use these methods with conflict notification so that you can validate the accuracy of resolved uniqueness conflicts.

Delete Conflict Resolution

You should always design advanced replication environments to avoid delete conflicts. If avoiding delete conflicts is too restrictive for an application design, you can write custom delete conflict resolution methods and assign them to replicated tables. Oracle does not offer any prebuilt delete conflict resolution methods. See "User-Defined Conflict Resolution Methods" on page 6-48 for more information about writing your own conflict resolution methods.

Conflict Resolution Methods

To resolve replication conflicts automatically, you can assign one or more conflict resolution methods. Oracle has many prebuilt conflict resolution methods that you can use to resolve conflicts. If necessary, you can build your own methods to resolve conflicts. The following sections explain more about prebuilt and custom conflict resolution methods.

Prebuilt Update Conflict Resolution Methods

Oracle offers the following prebuilt methods for update conflicts that you can assign to a column group.

- Overwrite and discard value.
- Minimum and maximum value.
- Earliest and latest timestamp value.

- Additive and average value.
- Priority groups and site priority.

Additional Information: For complete information about each prebuilt update conflict resolution method, "Prebuilt Update Conflict Resolution Methods" on page 6-18.

Oracle’s prebuilt update conflict resolution methods have varying characteristics in their ability to converge replicated data. For example, the additive conflict resolution method can converge replicated data managed by more than one master site, but the earliest timestamp method cannot.

Additional Information: For specific information about the data convergence property of each prebuilt conflict resolution method, "Guaranteeing Data Convergence" on page 6-38.

Prebuilt Uniqueness Conflict Resolution Methods

Oracle offers the following prebuilt uniqueness conflict resolution methods that you can assign to PRIMARY KEY and UNIQUE constraints.

- Append site name to duplicate value.
- Append sequence to duplicate value.
- Discard duplicate value.

Oracle’s prebuilt uniqueness conflict resolution methods do not converge the data in a replicated environment; they simply provide techniques for resolving PRIMARY KEY and UNIQUE constraint violations. Therefore, when you use one of Oracle’s uniqueness conflict resolution methods to resolve conflicts, you should also employ a notification mechanism to alert you to uniqueness conflicts when they happen and then manually converge replicated data, if necessary.

Additional Information: For complete information about Oracle’s prebuilt uniqueness conflict resolution methods, "Prebuilt Uniqueness Resolution Methods" on page 6-35.

Restrictions of Prebuilt Conflict Resolution Methods

Oracle’s prebuilt conflict resolution methods do not support the following situations.

- Delete conflicts.
- Changes to primary key (or identity key) columns.

- NULLs in the columns that you designate to resolve the conflict.
- Referential integrity constraint violations.

For these situations, you must either provide your own conflict resolution method or determine a method of resolving error transactions manually.

Custom Conflict Resolution and Notification Methods

In addition to using Oracle's prebuilt conflict resolution methods, you can also consider using conflict logging and conflict notification to compliment conflict resolution. Oracle lets you configure a replicated table to call user-defined methods that record conflict information or notify you when Oracle cannot resolve a conflict. You can configure column groups, constraints, and replicated tables to notify you for all conflicts, or for only those conflicts that Oracle cannot resolve.

Additional Information: For more information about writing your own conflict resolution and notification methods, see "User-Defined Conflict Resolution Methods" on page 6-48.

Using Multiple Conflict Resolution Methods

Indicating multiple conflict resolution methods for a column group allows Oracle to resolve a conflict in different ways should others fail to resolve the conflict. When trying to resolve a conflict, Oracle executes each group's methods in the order that you list. The algorithm that Oracle uses to resolve update conflicts is as follows.

1. Starting with the first column group, the receiving master site examines each field in the group to determine if it has changed and, if so, if there is a conflict between the old, new, and current values.
2. If no conflict occurred, Oracle can continue to the next column group. If a conflict occurred, Oracle calls the conflict resolution method with the lowest assigned sequence number for the column group.
3. If the conflict resolution method successfully resolves the conflict, Oracle holds the appropriate values for the columns pending determination of status.
4. If the method cannot resolve the conflict, Oracle continues with the next method until it can resolve the conflict or when no more methods are available.
5. After evaluating all column groups (including the shadow column group) and successfully resolving any conflicts, Oracle stores the new values for the columns.

6. If Oracle is unable to resolve any conflict using assigned methods, the receiving site logs the entire transaction as an error transaction in the site's replication catalog and does not change the values in the local row.

You should use multiple conflict resolutions methods for the following reasons:

- To employ alternative conflict resolution methods when the preferred method cannot resolve a conflict.
- To receive automatic notification of replication conflicts when they occur.

The following sections explain more about each issue.

Note: You can also assign multiple conflict resolution methods to a PRIMARY KEY or UNIQUE constraint to resolve uniqueness conflicts, and to a replicated table to resolve delete conflicts.

Using Multiple Conflict Resolution Methods for Backup In certain situations, the preferred conflict resolution method that you set for a column group, constraint, or table might not always succeed. If this is at all possible, you should specify a sequence of one or more alternative methods to increase the possibility that Oracle can perform conflict resolution without the need for manual resolution.

Some system-defined conflict resolution methods cannot guarantee successful resolution of conflicts in all circumstances. For example, the latest timestamp update conflict resolution method uses a special timestamp column to determine and apply the most recent change. In the unlikely event that the row at the originating site and the row at another site change at precisely the same second, the latest timestamp method cannot resolve the conflict because Oracle stores time related information at the granularity of a second. If you declare a backup update conflict resolution method such as site priority, Oracle might be able to resolve the conflict automatically.

Using Multiple Conflict Resolution Methods for Notification Another reason to use multiple conflict resolution routines is to call a user-defined method that records conflict information or notifies you when Oracle cannot resolve a conflict. For example, you might decide to configure a PRIMARY KEY constraint to call a custom conflict notification method first and then resolve conflicts using the append site name uniqueness conflict resolution method.

Additional Information: For more information about conflict notification, "User-Defined Conflict Notification Methods" on page 6-52.

Overview of Conflict Resolution Configuration

If you decide that conflict resolution is necessary in your advanced replication system, you must first design your conflict resolution strategy and then implement it when you create replicated tables. The following sections provide you with an overview of the steps necessary to complete each stage of conflict resolution configuration.

Design and Preparation Guidelines for Conflict Resolution

Use the following guidelines to design and prepare for a conflict resolution strategy.

- Analyze your data to determine which *column groups* are appropriate for update conflict resolution, and which *conflict resolution methods* are appropriate for each column group. If you cannot avoid uniqueness and delete conflicts, you must also plan for these types of conflict resolution.
- Some of Oracle's prebuilt update conflict resolution methods require unique preparatory steps before use. For example:
  - When you use the earliest or latest timestamp update conflict resolution methods, the replicated table must have a DATE column that Oracle can use for timestamp comparison. Additionally, you might want to add a timestamp maintenance trigger to the table to address time synchronization issues. For more information about timestamp resolution, see "Earliest and Latest Timestamp" on page 6-20.
  - If any column groups in a replication table will use site priority or priority groups for conflict resolution, define the priority levels for each site or value. For more information about configuring priority groups and site priority groups, see "Using Priority Groups for Update Conflict Resolution" and "Using Site Priority for Update Conflict Resolution", respectively.
- If necessary, prepare for conflict notification methods.
  - Create a table to hold conflict notification information at each master site.
  - Create the PL/SQL procedure to record conflict notification in the table.
  - Add the custom defined conflict resolution methods to the package or add methods to automate e-mail notification (optional).

  For more information about configuring conflict notification, see "User-Defined Conflict Notification Methods" on page 6-52.

Implementing Conflict Resolution

After planning, use Oracle Replication Manager and Oracle's replication management API to configure conflict resolution for the replicated tables in a master group. In general, these steps include the following:

1. Suspend replication activity for the master group.
2. Configure conflict resolution for the replicated tables in the master group at the master definition site. For example, when configuring update conflict resolution for a table, use Replication Manager to create necessary column groups and assign update conflict resolution methods to the groups.
3. Regenerate replication support for the replicated tables or for all objects in the master group after you finish configuring conflict resolution.
4. Resume replication activity for the master group once you are finished with all modifications and test your conflict resolution configuration.

The following sections explain how to configure update, uniqueness, and delete conflict resolution.

Configuring Update Conflict Resolution

In a typical advanced replication environment, uniqueness and delete conflicts are not possible, and update conflicts, therefore, require the most attention during system design and configuration. Oracle's advanced replication facility uses *column groups* to detect and resolve update conflicts. The following sections explain how to configure column groups for a replicated table and associate update conflict resolution methods for column groups.

Creating a Column Group

After adding a table to a master group at the master definition site and while replication activity is suspended for the group, you can configure column groups for the table and establish conflict resolution.

1. Expand the database node that is the master definition site for the target master table.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Expand the master group that contains the table that you want to create a column group for.

5. Expand the **Replication Objects** node.
6. Expand the **Tables** node.
7. Expand the schema that contains the table that you want to create a column group for.
8. Select the table that you want to create a column group for.

   The table property sheet will appear in the right pane of the Replication Manager user interface.
9. Press the **Conflict Resolution** tab.
10. Press the **Create** button to display the **Create Column Group** dialog.
11. Enter a name for your column group in the **Group Name** field.
12. Select the columns that you want to add to your column group from the **Available Columns** list (press the <SHIFT> key to select a range of columns or press the <CTRL> key to individually select multiple columns).
13. Press the **Right Arrow** to add the selected columns to the **Columns** list.
14. If necessary, enter any user comments or notes in the **Remark** field.
15. Click **OK** to create the new column group.

API Equivalent: DBMS_REPCAT.MAKE_COLUMN_GROUP

Adding and Removing Columns in a Column Group

While replication activity is suspended for a master group, you can add or remove the columns in a column group for a table.

1. Expand the database node that is the master definition site for the target master table.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Expand the master group that contains table which includes the column group that you want to modify.
5. Expand the **Replication Objects** node.
6. Expand the **Tables** node.
7. Expand the schema that contains the table which includes the column group that you want to modify.

8. Select the table that contains the column group that you want to modify.

   The table property sheet will appear in the right pane of the Replication Manager user interface.

9. Press the **Conflict Resolution** tab.
10. Select the column group that you want to modify from the **Column Groups** list.
11. Press the **Edit** button to the right of the **Column Groups** list.
12. Select the columns that you want to remove from your column group from the **Columns** list (press the <SHIFT> key to select a range of columns or press the <CTRL> key to individually select multiple columns).
13. Press the **Left Arrow** to move the selected columns back to the **Available Columns** list.
14. If necessary, enter any user comments or notes in the **Remark** field.
15. Press the **OK** button to complete your modifications.

API Equivalent: DBMS_REPCAT.ADD_GROUPED_COLUMN, DBMS_REPCAT.DROP_GROUPED_COLUMN

Dropping a Column Group

While replication activity is suspended for a master group, you can drop a column group for a table.

1. Expand the database node that is the master definition site for the target master table.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Expand the master group that contains table which includes the column group that you want to drop.
5. Expand the **Replication Objects** node.
6. Expand the **Tables** node.
7. Expand the schema that contains the table which includes the column group that you want to drop.

8. Select the table that contains the column group that you want to drop.

   The table property sheet will appear in the right pane of the Replication Manager user interface.
9. Press the **Conflict Resolution** tab.
10. Select the column group that you want to drop from the **Column Groups** list.
11. Press the **Remove** button to the right of the **Column Groups** list.

API Equivalent: DBMS_REPCAT.DROP_COLUMN_GROUP

Managing a Group's Update Conflict Resolution Methods

While replication activity is suspended for a master group, you can use Oracle Replication Manager to assign, remove, and order the update conflict resolution methods for a column group of a replicated table.

1. Expand the database node that is the master definition site for the target master table.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Expand the master group that contains table which includes the column group that you want to manage.
5. Expand the **Replication Objects** node.
6. Expand the **Tables** node.
7. Expand the schema that contains the table which includes the column group that you want to manage.
8. Select the table that contains the column group that you want to manage.

   The table property sheet will appear in the right pane of the Replication Manager user interface.
9. Press the **Conflict Resolution** tab.
10. Select the column group that you want to manage from the **Column Groups** list.

At this point, you can assign, remove, or order the update conflict resolution methods for the selected column group. The following sections explain each procedure.

Note: You must suspend replication activity before creating or editing conflict resolution for a table. Furthermore, all changes that you make using the **Conflict** Resolution page are immediately committed to the replication environment.

Assigning an Update Conflict Resolution Method

To assign a new update conflict resolution method to the selected column group, press the **Add** button in the **Select Column Group Resolution Methods** object group to display the **Add Update Resolution Method** dialog and add a new update conflict resolution method for the target column group.

Note: Certain update conflict resolution methods require that you complete some preparatory work before assigning them to a column group (for example, priority groups). See the sections later in this chapter that discuss each type of conflict resolution method and any special requirements necessary to use them.

API Equivalent: DBMS_REPCAT.ADD_UPDATE_RESOLUTION

Removing an Update Conflict Resolution Method

To remove an update conflict resolution method from the selected column group, select the method to remove and then press the **Remove** button in the **Select Column Group Resolution Methods** object group.

API Equivalent: DBMS_REPCAT.DROP_UPDATE_RESOLUTION

Ordering a Column Group's Update Conflict Resolution Methods

To order or reorder the application of conflict resolution methods for the selected column group, press the **Reorder** button. When the Reorder Resolution Methods dialog box appears, Select a resolution method and use the arrow keys to reorder the resolution methods. Press the **OK** button when you have completed your resolution order modifications.

Prebuilt Update Conflict Resolution Methods

The following sections explain Oracle’s prebuilt methods that you can use to resolve update conflicts, including:

- Additive and average.
- Minimum and maximum.
- Earliest and latest timestamp.
- Overwrite and discard.
- Priority groups and site priority.

The following sections explain each prebuilt update conflict resolution method in detail.

Note: The conflict resolution methods that you assign need to ensure data convergence and provide results that are appropriate for how your business uses the data. For complete information about data convergence and Oracle’s prebuilt conflict resolution methods, see "Guaranteeing Data Convergence" on page 6-38.

Additive and Average

The *additive* and *average* methods work with column groups consisting of a single numeric column only.

- The additive method adds the difference between the old and new values at the originating site to the current value at the destination site.

  ```
  current value = current value + (new value - old value)
  ```

  The additive conflict resolution method provides convergence for any number of master sites.

- The average conflict resolution method averages the new column value from the originating site with the current value at the destination site.

  ```
  current value = (current value + new value)/2
  ```

  The average method cannot guarantee convergence if your replicated environment has more than one master. This method is useful for an environment with a single master site and multiple updateable snapshots.

Minimum and Maximum

When the advanced replication facility detects a conflict with a column group and calls the *minimum* value conflict resolution method, 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 select the minimum value conflict resolution method.

If the new value of the designated column is *less than* the current value, the column group values from the originating site are applied at the destination site (assuming that all other errors were successfully resolved for the row). If the new value of the designated column is greater than the current value, the conflict is resolved by leaving the current values of the column group unchanged.

Note: If the two values for the designated column are the same (for example, if the designated column was not the column causing the conflict), the conflict is not resolved, and the values of the columns in the column group remain unchanged. Designate a backup conflict resolution method to be used for this case.

The *maximum* value method is the same as the minimum value method, except that the values from the originating site are only applied if the value of the designated column at the originating site is *greater than* the value of the designated column at the destination site.

There are no restrictions on the datatypes of the columns in the column group. Convergence for more than one master site is only guaranteed if:

- For the maximum method, the column value is always increasing.
- For the minimum method, the column value is always decreasing.

Note: You should not enforce an always-increasing restriction by using a CHECK constraint because the constraint could interfere with conflict resolution.

Earliest and Latest Timestamp

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

The following example demonstrates an appropriate application of the latest timestamp update conflict resolution method:

1. A customer in Phoenix calls the local salesperson and updates her address information.
2. After hanging up the phone, the customer realizes that she gave the local salesperson the wrong postal code.
3. The customer tries to call the local salesperson with the correct postal code, but the salesperson cannot be reached.
4. The customer calls the headquarters, which is located in New York. The New York site, rather than the Phoenix site, correctly updates the address information.
5. The network connecting New York headquarters with the local Phoenix sales site goes down temporarily.
6. When the New York/Phoenix network connection comes back up, Oracle sees two updates for the same address, and detects a conflict at each site.
7. Using the *latest timestamp* method, Oracle selects the most recent update, and applies the address with the correct postal code.

The earliest timestamp method applies the changes from the site with the earliest timestamp, and the latest timestamp method applies the changes from the site with the latest timestamp.

Note: When you use a timestamp conflict resolution method, you should designate a backup method, such as *site priority*, to be called if two sites have the same timestamp.

Other Configuration Issues When you use timestamp resolution, you must carefully consider how time is measured on the different sites managing replicated data. For example, if a replicated environment crosses time zones, applications that use the system should convert all timestamps to a common time zone such as Greenwich Mean Time (GMT). Furthermore, if two sites in a system do not have their system clocks synchronized reasonably well, timestamp comparisons might not be accurate enough to satisfy application requirements.

There are two ways to maintain timestamp columns if you use the EARLIEST or LATEST timestamp update conflict resolution methods.

- Each application can include logic to synchronize timestamps.
- You can create a trigger for a replicated table to synchronize timestamps automatically for all applications.

A clock counts seconds as an increasing value. Assuming that you have properly designed your timestamping mechanism and established a backup method in case two sites have the same timestamp, the *latest* timestamp method (like the maximum value method) guarantees convergence. The *earliest* timestamp method, however, *cannot* guarantee convergence for more than one master.

Additional Information: For an example of a timestamp and site maintenance trigger, "Sample Timestamp and Site Maintenance Trigger" on page 6-32.

Overwrite and Discard

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

For example, if you have a single master site that you expect to be used primarily for queries, with all updates being performed at the snapshot sites, you might select the overwrite method. The overwrite and discard methods are also useful if:

- Your primary concern is data convergence.
- You have a single master site.
- There is no particular business rule for selecting one update over the other.
- You have multiple master sites and you supply a notification facility to notify the person who ensures that data is correctly applied, instead of logging the conflict in the DEFERROR view and leaving the resolution to your local database administrator.

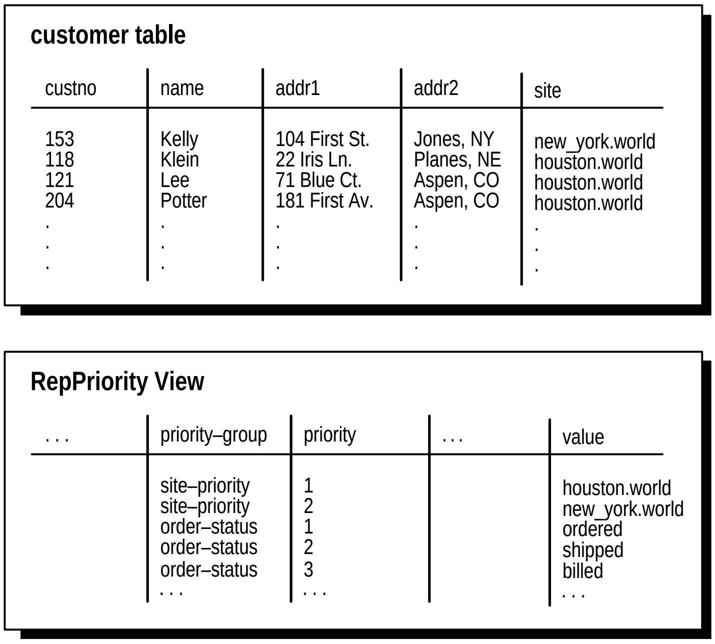
The overwrite method replaces the current value at the destination site with the new value from the originating site. Conversely, the discard method ignores the new value from the originating site.

Priority Groups and Site Priority

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

As shown in Figure 5–1, the REPPRIORITY view displays the priority level assigned to each priority group member (value that the "priority" column can contain). You must specify a priority for all possible values of the "priority" column.

Figure 6–1 Using Priority Groups



customer table

custno	name	addr1	addr2	site
153	Kelly	104 First St.	Jones, NY	new_york.world
118	Klein	22 Iris Ln.	Planes, NE	houston.world
121	Lee	71 Blue Ct.	Aspen, CO	houston.world
204	Potter	181 First Av.	Aspen, CO	houston.world
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

RepPriority View

. . .	priority–group	priority	. . .	value
	site–priority	1		houston.world
	site–priority	2		new_york.world
	order–status	1		ordered
	order–status	2		shipped
	order–status	3		billed
	. . .	. . .		. . .

The REPPRIORITY view displays the values of all priority groups defined at the current location. In the example shown in Figure 5–1, there are two different priority groups, site-priority and order-status. The *CUSTOMER* table is using the site-priority priority group.

Before you use Replication Manager to select the priority group method of update conflict resolution, you must designate which column in your table is the "priority" column.

Additional Information: To learn how to configure priority groups for update conflict resolution, "Using Priority Groups for Update Conflict Resolution" on page 6-23.

Site priority is a special kind of priority group. With site priority, the "priority" column you designate is automatically updated with the global database name of the site where the update originated. The REPPRIORITY view displays the priority level assigned to each database site. Site priority can be useful if one site is considered to be more likely to have the most accurate information. For example, in Figure 5–1, the New York site (priority value = 2) is corporate headquarters, while the Houston site (priority value = 1) is an updateable snapshot at a sales office. Therefore, the headquarters office is considered more likely than the sales office to have the most accurate information about the credit that can be extended to each customer.

Note: The priority-group column of the REPPRIORITY view shows both the site-priority group and the order-status group.

When you are using site priority, convergence with more than one master is not guaranteed. However, you can guarantee convergence with more than one master when you are using priority groups if the value of the "priority" column is always increasing. That is, the values in the priority column correspond to an ordered sequence of events; for example: ordered, shipped, billed.

Similar to priority groups, you must complete several preparatory steps before using Replication Manager to select site priority conflict resolution for a column group.

Additional Information: To learn how to configure site priority, "Using Site Priority for Update Conflict Resolution" on page 6-29.

Using Priority Groups for Update Conflict Resolution

To use the priority group method to resolve update conflicts, you must complete some special steps using Oracle's replication management API before using Replication Manager to assign the priority group resolution method to a column group. First, you must create a priority group. To create a priority group, do the following:

1. Define the name of the priority group and the datatype of the values in the group.
2. Define the priority level for each possible value of the "priority" column. This information is displayed in the REPPRIORITY view.

A single priority group can be used by multiple tables. Therefore, the name that you select for your priority group must be unique within a master group. The column corresponding to this priority group can have different names in different tables.

You must indicate which column in a table is associated with a particular priority group when you add the priority group conflict resolution method for the table. The priority group must therefore contain all possible values for all columns associated with that priority group.

For example, suppose that you have a replicated table, INVENTORY, with a column of type VARCHAR2, STATUS, that could have three possible values: ORDERED, SHIPPED, and BILLED. Now suppose that you want to resolve update conflicts based upon the value of the STATUS column. After suspending replication activity for the associated master group ACCT, complete the following steps at the master definition site to configure priority group resolution for the INVENTORY table.

1. Use Replication Manager to create a column group for the INVENTORY table that includes the STATUS column.
2. Create and populate the STATUS priority group that is associated with the master group ACCT. To do this, use the following API calls:

```
DBMS_REPCAT.DEFINE_PRIORITY_GROUP(
            gname                => 'acct',
            pgroup               => 'status',
            datatype              =>'varchar2');
DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
           gname                 => 'acct',
          pgroup               => 'status',
            value               => 'ordered',
            priority            => 1);
DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
            sname               => 'acct',
            pgroup              => 'status',
            value               => 'shipped',
            priority            => 2);
DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
            sname               => 'acct',
            pgroup              => 'status',
            value               => 'billed',
            priority            => 3);
```

3. Use Replication Manager to designate the PRIORITY GROUP conflict resolution method for the target column group.

> **Note:** Before creating or managing a priority group, use Replication Manager to suspend replication activity for the corresponding master group at the master definition site. After managing priority groups in any way, regenerate replication support for the associated replicated table before resuming replication activity for the master group.

The next several sections describe more about managing priority groups.

Creating a Priority Group

Use the DEFINE_PRIORITY_GROUP procedure in the DBMS_REPCAT package to create a new priority group for a master group, as shown in the following example:

```
DBMS_REPCAT.DEFINE_PRIORITY_GROUP(
            gname      => 'acct',
            pgroup     => 'status',
            datatype   => 'varchar2');
```

This example creates a priority group called STATUS for the ACCT master group. The members of this priority group have values of type VARCHAR2.

Additional Information: See "DEFINE_PRIORITY_GROUP" procedure in the *Oracle8i Replication API Reference* book for details.

Adding Members to a Priority Group

There are several different procedures in the DBMS_REPCAT package for adding members to a priority group. These procedures are of the form ADD_PRIORITY_*type*, where *type* is equivalent to the datatype that you specified when you created the priority group:

- ADD_PRIORITY_CHAR
- ADD_PRIORITY_VARCHAR2
- ADD_PRIORITY_NUMBER
- ADD_PRIORITY_DATE
- ADD_PRIORITY_RAW
- ADD_PRIORITY_NCHAR
- ADD_PRIORITY_NVARCHAR2

The specific 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.

The following example adds the value SHIPPED to the STATUS priority group:

```
DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
            gname     => 'acct',
            pgroup    => 'status',
            value     => 'shipped',
            priority    => 2);
```

Additional Information: The parameters for the ADD_PRIORITY_*datatype* procedures are described in Table 10–77 , and the exceptions are listed in Table 10–78 .

Altering the Value of a Member

There are several different procedures in the DBMS_REPCAT package for altering the value of a member of a priority group. These procedures are of the form ALTER_PRIORITY_*type*, where *type* is equivalent to the datatype that you specified when you created the priority group:

- ALTER_PRIORITY_CHAR
- ALTER_PRIORITY_VARCHAR2
- ALTER_PRIORITY_NUMBER
- ALTER_PRIORITY_DATE
- ALTER_PRIORITY_RAW
- ALTER_PRIORITY_NCHAR
- ALTER_PRIORITY_NVARCHAR2

The procedure that you must call is determined by the datatype of your "priority" column. Because a priority group member consists of a priority associated with a particular value, these procedures enable you to change the value associated with a given priority level.

The following example changes the recognized value of items at priority level 2 from SHIPPED to IN_SHIPPING:

```
DBMS_REPCAT.ALTER_PRIORITY_VARCHAR2(
               gname        => 'acct',
               pgroup       => 'status',
               old_value    => 'shipped',
               new_value      => 'in_shipping');
```

Additional Information: See the "ALTER_PRIORITY_*datatype*" procedures in the *Oracle8i Replication API Reference* book for details.

Altering the Priority of a Member

Use the ALTER_PRIORITY procedure in the DBMS_REPCAT package to alter the priority level associated with a given priority group member. Because a priority group member consists of a priority associated with a particular value, this procedure lets you raise or lower the priority of a given column value. Members with higher priority values are given higher priority when resolving conflicts.

The following example changes the priority of items marked as IN_SHIPPING from level 2 to level 4:

```
DBMS_REPCAT.ALTER_PRIORITY(
             gname            => 'acct',
             pgroup           => 'status',
             old_priority     => 2,
             new_priority     => 4);
```

Additional Information: The parameters for the ALTER_PRIORITY procedure are described in Table 10–87 , and the exceptions are listed in Table 10–88 .

Dropping a Member by Value

There are several different procedures in the DBMS_REPCAT package for dropping a member of a priority group by value. These procedures are of the form DROP_PRIORITY_*type*, where *type* is equivalent to the datatype that you specified when you created the priority group:

- DROP_PRIORITY_CHAR
- DROP_PRIORITY_VARCHAR2

- DROP_PRIORITY_NUMBER
- DROP_PRIORITY_DATE
- DROP_PRIORITY_RAW
- DROP_PRIORITY_NCHAR
- DROP_PRIORITY_NVARCHAR2

The procedure that you must call is determined by the datatype of your "priority" column.

In the following example, IN_SHIPPING is no longer a valid state for items in the STATUS priority group:

```
DBMS_REPCAT.DROP_PRIORITY_VARCHAR2(
          gname        => 'acct',
          pgroup       => 'status',
          value          => 'in_shipping');
```

Additional Information: The parameters for the DROP_PRIORITY_*datatype* procedures are described in Table 10–138, and the exceptions are listed in Table 10–139.

Dropping a Member by Priority

Use the DROP_PRIORITY procedure in the DBMS_REPCAT package to drop a member of a priority group by priority level.

In the following example, IN_SHIPPING (which was assigned to priority level 4) is no longer a valid state for items in the STATUS priority group:

```
DBMS_REPCAT.DROP_PRIORITY(
          gname         => 'acct',
          pgroup        => 'status',
          priority_num    => 4);
```

Additional Information: The parameters for the DROP_PRIORITY procedure are described in Table 10–136 , and the exceptions are listed in Table 10–137 .

Dropping a Priority Group

Use the DROP_PRIORITY_GROUP procedure in the DBMS_REPCAT package to drop a priority group for a given master group. For example, the following call drops the STATUS priority group:

```
DBMS_REPCAT.DROP_PRIORITY_GROUP(
          gname      => 'acct',
          pgroup       => 'status');
```

Attention: Before dropping a priority group, you remove the priority group update resolution method from all column groups that depend on the priority group. Query the REPRESOLUTION view to determine which column groups depend on a priority group.

Additional Information: The parameters for the DROP_PRIORITY_GROUP procedure are described in Table 10–140, and the exceptions are listed in Table 10–141.

Using Site Priority for Update Conflict Resolution

Site priority is a specialized form of priority groups. Thus, many of the procedures associated with site priority behave similarly to the procedures associated with priority groups.

If you chose to use the site priority method to resolve update conflicts, you must first create a site priority group before you can use Replication Manager to add this conflict resolution method to a column group. Creation of a site priority group consists of two steps.

1. Define the name of the site priority group.
2. Add each site to the site priority group and define its priority level. This information is displayed in the REPPRIORITY view.

In general, you need only one site priority group for a master group. This site priority group can be used by any number of replicated tables. The next several sections describe how to manage site priority groups.

Pre and Post Steps for Managing Site Priority

When configuring or managing site priority, keep in mind the following important order dependent operations.

- Before creating or managing a site priority group, use Replication Manager to suspend replication activity for the corresponding master group.
- You must manage site priority groups from the master definition site of the corresponding master group.
- After managing site priority groups in any way, use Replication Manager to regenerate replication support for the associated replicated table. Then you can use Replication Manager to resume replication activity for the master group.

Creating a Site Priority Group

Use the DEFINE_SITE_PRIORITY procedure in the DBMS_REPCAT package to create a new site priority group for a master group, as shown in the following example:

```
DBMS_REPCAT.DEFINE_SITE_PRIORITY(
            gname     => 'acct',
            name        => 'site');
```

This example creates a site priority group called SITE for the ACCT object group.

Additional Information: The parameters for the DEFINE_SITE_PRIORITY procedure are described in Table 10–124, and the exceptions are listed in Table 10–125.

Adding a Site to the Group

Use the ADD_SITE_PRIORITY_SITE procedure in the DBMS_REPCAT package to add a new site to a site priority group, as shown in the following example:

```
DBMS_REPCAT.ADD_SITE_PRIORITY_SITE(
           gname       => 'acct',
           name        => 'site',
           site        => 'hq.widgetek.com',
           priority      => 100);
```

This example adds the HQ site to the SITE group and sets its priority level to 100.

Note: The highest priority is given to the site with the highest priority value. Priority values do not have to be consecutive integers.

Additional Information: The parameters for the ADD_SITE_PRIORITY_SITE procedure are described in Table 10–79, and the exceptions are listed in Table 10–80.

Altering the Priority Level of a Site

Use the ALTER_SITE_PRIORITY procedure in the DBMS_REPCAT package to alter the priority level associated with a given site, as shown in the following example:

```
DBMS_REPCAT.ALTER_SITE_PRIORITY(
            gname        => 'acct',
            name         => 'site',
            old_priority => 100,
            new_priority   => 200);
```

This example changes the priority level of a site in the SITE group from 100 to 200.

> **Note:** The highest priority is given to the site with the highest priority value. Priority values do not have to be consecutive integers.

Additional Information: The parameters for the ALTER_SITE_PRIORITY procedure are described in Table 10–91, and the exceptions are listed in Table 10–92.

Altering the Site Associated with a Priority Level

Use the ALTER_SITE_PRIORITY_SITE procedure in the DBMS_REPCAT package to alter the site associated with a given priority level, as shown in the following example:

```
DBMS_REPCAT.ALTER_SITE_PRIORITY_SITE(
           gname         => 'acct',
           name          => 'site',
           old_site      => 'hq.widgetek.com',
           new_site        => 'hq.widgetworld.com);
```

This example changes the global database name of the HQ site to HQ.WIDGETWORLD.COM, while its priority level remains the same.

Additional Information: The parameters for the ALTER_SITE_PRIORITY_ SITE procedure are described in Table 10–93, and the exceptions are listed in Table 10–94.

Dropping a Site by Site Name

Use the DROP_SITE_PRIORITY_SITE procedure in the DBMS_REPCAT package to drop a given site, by name, from a site priority group, as shown in the following example:

```
DBMS_REPCAT.DROP_SITE_PRIORITY_SITE(
           gname    => 'acct',
           name     => 'site',
           site       => 'hq.widgetek.com');
```

This example drops the HQ site from the SITE group.

Additional Information: The parameters for the DROP_SITE_PRIORITY_SITE procedure are described in Table 10–144, and the exceptions are listed in Table 10–145.

Dropping a Site by Priority Level

Use the DBMS_REPCAT.DROP_PRIORITY procedure described on page 5 - 27 to drop a site from a site priority group by priority level.

Dropping a Site Priority Group

Use the DROP_SITE_PRIORITY procedure in the DBMS_REPCAT package to drop a site priority group for a given master group, as shown in the following example:

```
DBMS_REPCAT.DROP_SITE_PRIORITY(
           gname   => 'acct',
           name      => 'site');
```

In this example, SITE is no longer a valid site priority group.

Attention: Before calling this procedure, you must call the DROP_UPDATE_RESOLUTION procedure for any column groups in the master group that are using the SITE PRIORITY conflict resolution method with this site priority group. You can determine which column groups are affected by querying the REPRESOLUTION view.

Additional Information: The parameters for the DROP_SITE_PRIORITY procedure are described in Table 10–142, and the exceptions are listed in Table 10–143.

Sample Timestamp and Site Maintenance Trigger

In either a trigger or in your application, you must implement the logic necessary to maintain the timestamp and site information. The following example trigger considers clock synchronization problems, but needs to be modified if the application crosses time zones.

Because the example trigger uses one of the generated procedures to check whether or not the trigger should actually be fired, it is necessary to generate replication support for the corresponding CUSTOMERS table before creating the trigger. This will also allow transactions on the CUSTOMERS table to be propagated.

```
dbms_repcat.generate_replication_support(sname => 'SALES',
                                         oname => 'CUSTOMERS',
                                        type  => 'TABLE');
```

Now you can define the trigger:

```
CREATE OR REPLACE TRIGGER sales.t_customers
  BEFORE INSERT OR UPDATE ON sales.customers
  FOR EACH ROW
DECLARE
  timestamp$x DATE := SYSDATE;
  site$x VARCHAR2(128) := dbms_reputil.global_name;
BEGIN
 -- Don't fire if a snapshot refreshing;
 -- Don't fire if a master and replication is turned off
 IF (NOT (dbms_snapshot.i_am_a_refresh)
 AND dbms_reputil.replication_is_on) THEN
  IF NOT dbms_reputil.from_remote THEN
    IF INSERTING THEN
       -- set site and timestamp columns.
       :new."TIMESTAMP" := TIMESTAMP$X;
       :new."SITE" := SITE$X;
    ELSIF UPDATING THEN
        IF(:old."ADDR1" = :new."ADDR1" OR
          (:old."ADDR1" IS NULL AND :new."ADDR1" IS NULL)) AND
          (:old."ADDR2" = :new."ADDR2" OR
          (:old."ADDR2" IS NULL AND :new."ADDR2" IS NULL)) AND
          (:old."FIRST_NAME" = :new."FIRST_NAME" OR
          (:old."FIRST_NAME" IS NULL AND  :new."FIRST_NAME"
                                               IS NULL)) AND
          (:old."LAST_NAME" = :new."LAST_NAME" OR
          (:old."LAST_NAME" IS NULL AND :new."LAST_NAME" IS NULL)) AND
          (:old."SITE" = :new."SITE" OR
          (:old."SITE" IS NULL AND :new."SITE" IS NULL)) AND
          (:old."TIMESTAMP" = :new."TIMESTAMP" OR
          (:old."TIMESTAMP" IS NULL AND :new."TIMESTAMP" IS NULL)) THEN
          -- column group was not changed; do nothing
          NULL;
        ELSE
          -- column group was changed; set site and timestamp columns.
          :new."SITE" := SITE$X;
          :new."TIMESTAMP" := TIMESTAMP$X;
          -- consider time synchronization problems;
          -- previous update to this row may have originated from a site
          -- with a clock time ahead of the local clock time.
          IF :old."TIMESTAMP" IS NOT NULL AND
             :old."TIMESTAMP" > :new."TIMESTAMP" THEN
             :new."TIMESTAMP" := :old."TIMESTAMP" + 1 / 86400;
          ELSIF :old."TIMESTAMP" IS NOT NULL AND
             :old."TIMESTAMP" = :new."TIMESTAMP" AND
            (:old."SITE" IS NULL OR :old."SITE" != :new."SITE") THEN
            :new."TIMESTAMP" := :old."TIMESTAMP" + 1 / 86400;
          END IF;
        END IF;
      END IF;
    END IF;
  END IF;
END;
```

Next, use Replication Manager to add the trigger to the master group that contains the replicated table and resume replication activity for the master group.

Configuring Uniqueness Conflict Resolution

In a typical advanced replication environment, you should try to avoid the possibility of uniqueness conflicts. However, if uniqueness conflicts must be addressed, you can assign one or more conflict resolution methods to a PRIMARY KEY or UNIQUE constraint in a replicated table to resolve uniqueness conflicts when they occur. The following sections explain how to configure a replicated table and associate uniqueness conflict resolution methods for PRIMARY KEY and UNIQUE constraints.

Assigning a Uniqueness Conflict Resolution Method

To assign a uniqueness conflict resolution method to a constraint, use the ADD_ UNIQUE_RESOLUTION procedure of the DBMS_REPCAT package. For example, the following statement assigns the APPEND_SEQUENCE uniqueness conflict resolution method to the C_CUST_NAME constraint of the CUSTOMERS table:

```
DBMS_REPCAT.ADD_UNIQUE_RESOLUTION (
                   sname                 => 'acct',
                   oname                 => 'customers',
                   constraint_name       => 'c_cust_name',
                   sequence_no           =>  1,
                   method                => 'APPEND SEQUENCE',
                   comment               => 'Resolve Conflict',
                   parameter_column_name => 'last_name');
```

Additional Information: The parameters for the ADD_UNIQUE_RESOLUTION procedure are described in Chapter 10.

Removing a Uniqueness Conflict Resolution Method

To remove a uniqueness conflict resolution method from a constraint, use the DROP_ UNIQUE_RESOLUTION procedure of the DBMS_REPCAT package. For example, the following statement drops the uniqueness conflict resolution method assigned in the previous example:

```
DBMS_REPCAT.DROP_UNIQUE_RESOLUTION (
     sname              => 'acct',
     oname              => 'customers',
     constraint_name    => 'c_cust_name',
     sequence_no        =>  1 );
```

Additional Information: See the "DROP_UNIQUE_RESOLUTION" procedure in the *Oracle8i Replication API Reference* book for details.

Prebuilt Uniqueness Resolution Methods

Oracle provides three prebuilt methods for resolving uniqueness conflicts:

- Append the global name of the originating site to the column value from the originating site.
- Append a generated sequence number to the column value from the originating site.
- Discard the row value from the originating site.

The following sections explain each uniqueness conflict resolution method in detail.

Note: Oracle's prebuilt uniqueness conflict resolution methods do not actually converge the data in a replicated environment; they simply provide techniques for resolving constraint violations. When you use one of Oracle's uniqueness conflict resolution methods, you should also use a notification mechanism to alert you to uniqueness conflicts when they happen and then manually converge replicated data, if necessary. For more information about data convergence, see "Guaranteeing Data Convergence".

Append Site Name/Sequence

The *append site name* and *append sequence* methods work by appending a string to a column that is generating a DUP_VAL_ON_INDEX exception. Although these methods allow the column to be inserted or updated without violating a unique integrity constraint, they do not provide any form of convergence between multiple master sites. The resulting discrepancies must be manually resolved; therefore, these methods are meant to be used with some form of a notification facility**.**

Note: Both append site name and append sequence can be used on character columns only.

These methods can be useful when the availability of the data may be more important than the complete accuracy of the data. To allow data to be available as soon as it is replicated

- Select append site name or append sequence.
- Use a notification scheme to alert the appropriate person to resolve the duplication, instead of logging a conflict.

When a uniqueness conflict occurs, the *append site name* method appends the global database name of the site originating the transaction to the replicated column value. The name is appended to the first period (.). For example, HOUSTON.WORLD becomes HOUSTON.

Similarly, the *append sequence* method appends a generated sequence number to the column value. The column value is truncated as needed. If the generated portion of the column value exceeds the column length, the conflict method does not resolve the error.

Discard

The *discard uniqueness* conflict resolution method resolves uniqueness conflicts by simply discarding the row from the originating site that caused the error. This method does not guarantees convergence with multiple masters and should be used with a notification facility.

Unlike the append methods, the discard uniqueness method minimizes the propagation of data until data accuracy can be verified.

Configuring Delete Conflict Resolution

In a typical advanced replication environment, you should try to avoid the possibility of delete conflicts. However, if the possibility of delete conflicts must be addressed, you can create your own delete conflict resolution methods and then assign them to a replicated table. The following sections explain how to configure a replicated table and associate user-defined delete conflict resolution methods to the table.

Assigning a Delete Conflict Resolution Method

To assign a user-defined delete conflict resolution method to a replicated table, use the ADD_ DELETE_RESOLUTION procedure of the DBMS_REPCAT package. For example, the following statement assigns the CUSTOMERS_DELETE_M1 user-defined function as a delete conflict resolution method for the CUSTOMERS table:

```
DBMS_REPCAT.ADD_DELETE_RESOLUTION (
     sname                 => ’acct’,
     oname                 => ’customers’,
     sequence_no           => 1,
     parameter_column_name  => ’last_name’,
     function_name         => ’customers_delete_m1’ );
```

Additional Information: The parameters for the ADD_DELETE_RESOLUTION procedure are described in Chapter 10.

Removing a Delete Conflict Resolution Method

To remove a delete conflict resolution method from a replicated table, use the DROP_ DELETE_RESOLUTION procedure of the DBMS_REPCAT package. For example, the following statement drops the delete conflict resolution method assigned in the previous example:

```
DBMS_REPCAT.DROP_DELETE_RESOLUTION (
     sname                  => 'acct',
     oname                  => 'customers',
     sequence_no            => 1 );
```

Additional Information: The parameters for the DROP_DELETE_RESOLUTION procedure are described in Chapter 10.

Guaranteeing Data Convergence

Data convergence is a requirement in an advanced replication system. Data convergence happens when all replication sites ultimately have the same values for a given row. When you configure an advanced replication system, the conflict resolution strategy you design must guarantee data convergence. Table 6–1 summarizes Oracle's prebuilt update conflict resolution methods and in which types of configurations they guarantee data convergence between multiple master sites and their associated snapshot sites.

Note: Oracle's prebuilt uniqueness conflict resolution methods do not ensure data convergence in any type of replicated environment. Therefore, you should configure conflict notification along with uniqueness conflict resolution and manually converge data, if necessary.

Table 6–1 Data Convergence Properties of Update Conflict Resolution Methods

Resolution Methods	One Master Site	Any Number of Master Sites
MINIMUM	YES	YES (column values must always decrease)
MAXIMUM	YES	YES (column values must always increase)
EARLIEST TIMESTAMP	YES (with backup method)	NO
LATEST TIMESTAMP	YES (with backup method)	YES (with backup method)
PRIORITY GROUP	YES	YES (with ordered update values)
SITE PRIORITY	YES	NO
OVERWRITE	YES	NO
DISCARD	YES	NO
AVERAGE	YES	NO
ADDITIVE	YES	YES

If you have more than one master site, the *overwrite, discard, average, earliest timestamp,* and *site priority* methods cannot guarantee data convergence; consequently, these methods should only be used in conjunction with a notification facility. Furthermore, network failures and infrequent pushing of the deferred remote procedure call (RPC) queue increase the likelihood of non-convergence for these methods.

However, the *overwrite, discard, average, earliest timestamp,* and *site priority* methods are perfect for mass deployment environments that use a single master site with many updateable snapshots.

Avoiding Ordering Conflicts

Ordering conflicts can occur in advanced replication configurations with three or more master sites. If propagation to master site X is blocked for any reason, updates to replicated data can continue to be propagated among other master sites. When propagation resumes, these updates may be propagated to site X in a different order than they occurred on the other masters, and these updates may conflict. By default, the resulting conflicts will be recorded in the error log and can be re-executed after the transactions they depend upon are propagated and applied. Whenever possible, however, it is best to avoid or automatically resolve ordering conflicts. For example, you should select conflict resolution routines that ensure convergence in multimaster configurations where ordering conflicts are possible.

The example in Table 6–2 shows how having three master sites can lead to ordering conflicts. Master Site A has priority 30; Master Site B has priority 25; and Master Site C has priority 10; *x* is a column of a particular row in a column group that is assigned the *site-priority* conflict resolution method. The highest priority is given to the site with the highest priority value. Priority values can be any Oracle number and do not have to be consecutive integers.

Table 6–2 Example: Ordering Conflicts With Site Priority

Time	Action	Site A	Site B	Site C
1	All sites are up and agree that x = 2.	2	2	2
2	Site A updates x = 5.	5	2	2
3	Site C becomes unavailable.	5	2	down
4	Site A pushes update to Site B. Site A and Site B agree that x = 5. Site C is still unavailable. The update transaction remains in the queue at Site A.	5	5	down
5	Site C becomes available with x = 2. Sites A and B agree that x = 5.	5	5	2
6	Site B updates x = 5 to x = 7.	5	7	2
7	Site B pushes the transaction to Site A. Sites A and B agree that x = 7. Site C still says x = 2.	7	7	2
8	Site B pushes the transaction to Site C. Site C says the old value of x = 2; Site B says the old value of x = 5. Oracle detects a conflict and resolves it by applying the update from Site B, which has a higher priority level (25) than Site C (10). All site agree that x = 7.	7	7	7
9	Site A successfully pushes its transaction (x = 5) to Site C. Oracle detects a conflict because the current value at Site C (x = 7) does not match the old value at Site A (x = 2). Site A has a higher priority (30) than Site C (10). Oracle resolves the conflict by applying the outdated update from Site A (x = 5). Because of this ordering conflict, the sites no longer converge.	7	7	5

You can guarantee convergence when using priority groups if you require that the flow of ownership be ordered. For example, the workflow model dictates that information flow one-way through a three-step sequence:

1. From the ORDERING site.
2. to the SHIPPING site.
3. to the BILLING site.

If the billing site receives a change to a row from the ordering site after the billing site received a change to that row from the shipping site, the billing site ignores the out-of-order change because the change from shipping has a higher priority.

Suggestion: To help determine which conflict resolution method to use, make a diagram or time-action table (such as Table 6–2) to help uncover any potential loopholes in your conflict resolution methodology.

Minimizing Data Propagation for Update Conflict Resolution

To detect and resolve an update conflict for a row, the propagating site must send a certain amount of data about the new and old versions of the row to the receiving site. Depending on your environment, the amount of data that Oracle propagates to support update conflict detection and resolution can be different.

For example, when you create a replicated table and all participating sites are Oracle8 or greater databases, you can choose minimize the amount of data that must be communicated to determine conflicts for each changed row in the table. In this case, Oracle propagates:

- The old value of the primary key and each column in the column group (the value before the modification).
- The new value of each updated column in the column group.

Note: For an inserted row, the row has no *old* value. For a deleted row, the row has no *new* value.

In general, you should choose to minimize data propagation in Oracle8 or greater replication environments to reduce the amount of data that Oracle needs to transmit across the network. As a result, you can help to improve overall system performance.

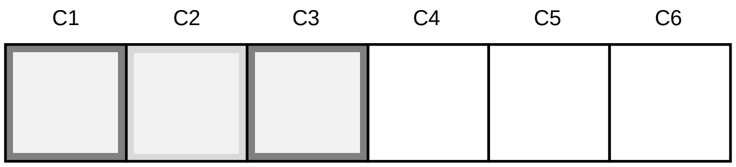
Alternatively, when a replicated environment uses both Oracle7 and Oracle8 or greater sites, you cannot minimize the communication of row data for update conflict resolution. In this case, Oracle must propagate the entire old and new versions of each changed row to perform conflict resolution.

When you use Replication Manager to generate support for replicated tables, you can minimize data propagation by enabling the **Minimize Communications** setting of the **Edit Replication Object property** sheet. When using the replication API, you can minimize data propagation by setting the `min_communication` parameter to TRUE in the following DBMS_REPCAT procedures:

- CREATE_SNAPSHOT_REPOBJECT
- GENERATE_REPLICATION_SUPPORT
- GENERATE_REPLICATION_TRIGGER
- GENERATE_SNAPSHOT_SUPPORT

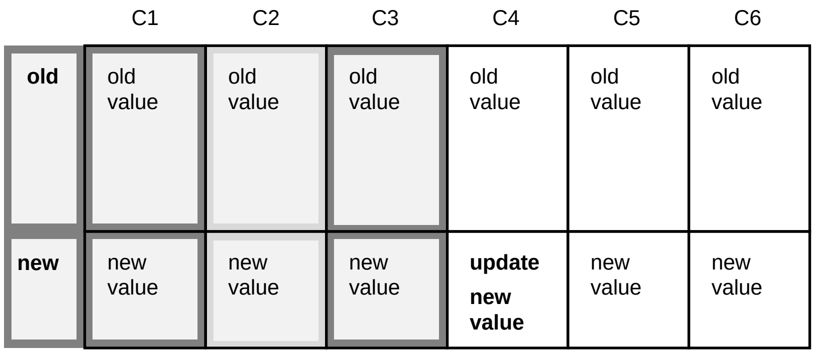
Minimizing Communication Examples

In the replicated table below, columns 1 and 3 together compose the primary key. There are two column groups, columns 1 - 3 and columns 4 - 6.



column group 1 = C1, C2, & C3
column group 2 = C4, C5, & C6
primary key = C1 & C3

If you disable the **Minimize Communication** setting when generating replication support for the table, Oracle sends six old values (C1 -C6) and six new values (C1 - C6) for any update. For example, if you update column C4,



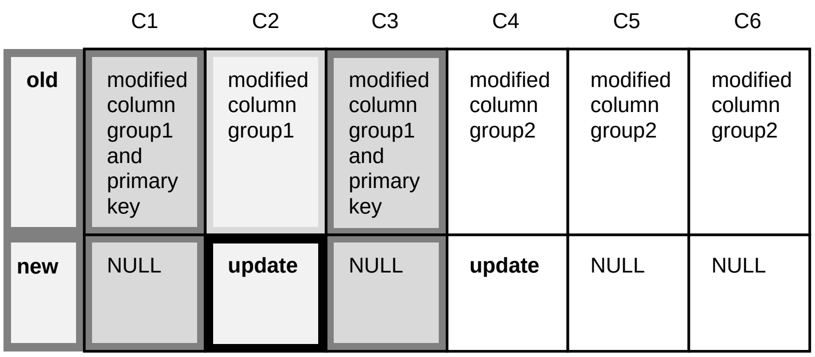
Alternatively, if you enable the **Minimize Communication** setting when generating replication support for the table, Oracle minimizes communication. For example, when you update **column C4,** Oracle sends:

- Old values for the primary key.
- Old values for the modified column group.
- NULLs for old values not in the primary key and the modified column group.
- NULL new values for all columns not updated.
- Actual new values for updated columns.

	C1	C2	C3	C4	C5	C6
old	primary key	NULL	primary key	modified column group	modified column group	modified column group
new	NULL	NULL	NULL	**update new value**	NULL	NULL

If you update columns **C2 and C4**, Oracle sends:

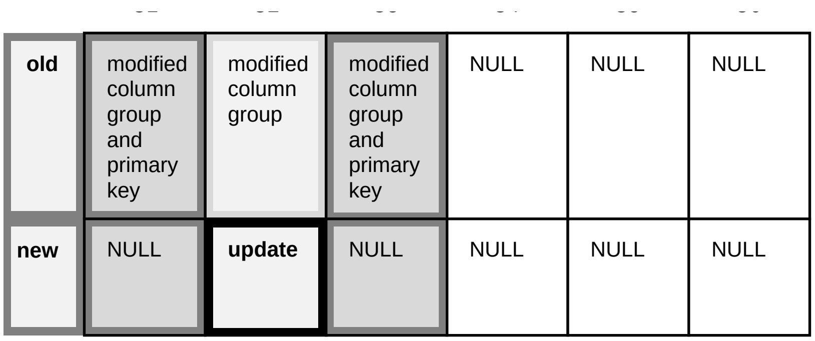
- Old values for both modified column groups.
- Old values for the primary key columns.
- New values for the two updates.
- NULLs for the other four new values.



	C1	C2	C3	C4	C5	C6
old	modified column group1 and primary key	modified column group1	modified column group1 and primary key	modified column group2	modified column group2	modified column group2
new	NULL	**update**	NULL	**update**	NULL	NULL

If you update **column 2,** Oracle sends:

- Old values for the primary key.
- Old values for members of the modified column group.
- The new values for the updated column.
- NULLs for the other five new values.



old	modified column group and primary key	modified column group	modified column group and primary key	NULL	NULL	NULL
new	NULL	**update**	NULL	NULL	NULL	NULL

Further Reducing Data Propagation

If you have minimized propagation using the method described above, you can further reduce data propagation in some cases by using the DBMS_REPCAT. SEND_AND_COMPARE_OLD_VALUES procedure to send old values only if they are needed to detect and resolve conflicts. For example, the latest timestamp conflict detection and resolution method does not require old values for non-key and non-timestamp columns.

Suggestion: Further minimizing propagation of old values is particularly valuable if you are replicating LOB datatypes and do not expect conflicts on these columns.

Attention: You must ensure that the appropriate old values are propagated to detect and resolve anticipated conflicts. User-supplied conflict resolution procedures must deal properly with NULL old column values that are transmitted. Using SEND_AND_COMPARE_OLD_VALUES to further reduce data propagation reduces protection against unexpected conflicts.

To further reduce data propagation execute the following procedure:

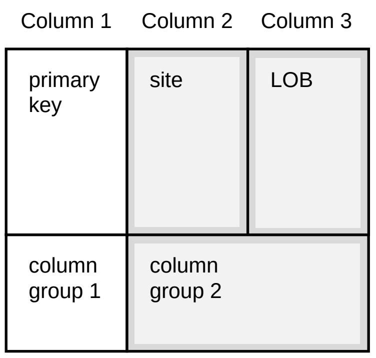
```
DBMS_REPCAT.VARCHAR2SDBMS_REPCAT.SEND_AND_COMPARE_OLD_VALUES(
    sname        IN    VARCHAR2,
    oname        IN    VARCHAR2,
    column_list  IN    VARCHAR2 |
    column_table IN    DBMS_REPCAT.VARCHAR2s
    operation    IN    VARCHAR2 := 'UPDATE',
     send          IN     BOOLEAN  :=  TRUE);
```

After executing this procedure, you must generate replication support again with `min_communication` set to TRUE for this change to take effect.

Note: The operation parameter allows you to decide whether or not to transmit old values for non-key columns when rows are deleted and/or when non-key columns are updated. If you do not send the old value, 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.

The specified behavior for old column values is exposed in two columns in the REPCOLUMN view: COMPARE_OLD_ON_DELETE ('Y' or 'N') and COMPARE_OLD_ON_UPDATE ('Y' or 'N').

The following example shows how you can further reduce data propagation by using SEND_AND_COMPARE_OLD_VALUES. Consider a table called 'SCOTT.REPORTS' with 3 columns. Column 1 is the primary key and is in its own column group (column group 1). Column 2 and column 3 are in a second column group (column group 2).



The conflict resolution strategy for the second column group is site priority. Column 2 is a VARCHAR2 column containing the site name. Column 3 is a LOB column. Whenever you update the LOB, you must also update column 2 with the global name of the site at which the update occurs. Because there are no triggers for piecewise updates to LOBs, you must explicitly update column 2 whenever you do a piecewise update on the LOB.

Suppose you generate replication support for SCOTT.REPORTS with `min_communication` set to TRUE and then use an UPDATE statement to modify column 2 (the site name) and column 3 (the LOB). The deferred remote procedure call (RPC) contains the new value of the site name and the new value of the LOB because they were updated. The deferred RPC also contains the old value of the primary key (column 1), the old value of the site name (Column 2), and the old value of the LOB (Column 3).

Note: The conflict detection and resolution strategy does not require the old value of the LOB. Only column C2 (the site name) is required for both conflict detection and resolution. Sending the old value for the LOB could add significantly to propagation time.

To ensure that the old value of the LOB is not propagated when either column C2 or column C3 is updated, make the following call:

```
  dbms_repcat.send_and_compare_old_values(
            sname         => 'SCOTT',
            oname         => 'REPORTS'
            column_list   => 'C3',
            operation     => 'UPDATE',
            send          => FALSE);
```

You must generate replication support for SCOTT.REPORTS again with `min_communication` set to TRUE for this change to take effect. Suppose you subsequently use an UPDATE statement to modify column 2 (the site name) and column 3 (the LOB). The deferred RPC contains the old value of the primary key (column 1), the old and new values of the site name (column 2), and just the new value of the LOB (column 3). The deferred RPC contains NULLs for the new value of the primary key and the old value of the LOB.

Additional Information: See the "SEND_AND_COMPARE_OLD_VALUES" procedure in the *Oracle8i Replication API Reference* book for details.

User-Defined Conflict Resolution Methods

Oracle allows you to write your own conflict resolution or notification methods. A user-defined conflict resolution method is a PL/SQL function that returns either TRUE or FALSE. TRUE indicates that the method has successfully resolved all conflicting modifications for a column group. If the method cannot successfully resolve a conflict, it should return FALSE. Oracle continues to evaluate available conflict resolution methods, in sequence order, until either a method returns TRUE or there are no more methods available.

If the conflict resolution method raises an exception, Oracle stops evaluation of the method, and, if any other methods were provided to resolve the conflict (with a later sequence number), Oracle does not evaluate them.

Conflict Resolution Method Parameters

The parameters needed by a user-defined conflict resolution method are determined by the type of conflict being resolved (unique, 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.

Attention: Type checking of parameter columns in user-defined conflict resolution methods is not performed until you regenerate replication support for the associated replicated table.

Resolving Update Conflicts

For update conflicts, a user-defined function should accept the following values for each column in the column group:

- Old column value from the initiating site. The mode for this parameter is IN. This value should not be changed.
- New column value from the initiating site. The mode for this parameter is IN OUT. If the function can resolve the conflict successfully, it should modify the new column value as needed.
- Current column value from the receiving site. The mode for this parameter is IN.

The old, new, and current values for a column are received consecutively. The final argument to the conflict resolution method should be a Boolean flag. If this flag is FALSE, it indicates that you have updated the value of the IN OUT parameter, new, and that you should update the current column value with this new value. If this flag is TRUE, it indicates that the current column value should not be changed.

Resolving Uniqueness Conflicts

Uniqueness conflicts can occur as the result of an INSERT or UPDATE. Your uniqueness conflict resolution method should accept the new column value from the initiating site in IN OUT mode for each column in the column group. The final parameter to the conflict resolution method should be a Boolean flag.

If the method can resolve the conflict, it should modify the new column values so that Oracle can insert or update the current row with the new column values. Your function should set the Boolean flag to TRUE if it wants to discard the new column values, and FALSE otherwise.

Because a conflict resolution method cannot guarantee convergence for uniqueness conflicts, a user-defined uniqueness resolution method should include a notification mechanism.

Resolving Delete Conflicts

Delete conflicts occur when you successfully delete from the local site, but the associated row cannot be found at the remote site (for example, because it had been updated). For delete conflicts, the function should accept old column values in IN OUT mode for the entire row. The final parameter to the conflict resolution method should be a BOOLEAN flag.

If the conflict resolution method can resolve the conflict, it modifies the old column values so that Oracle can delete the current row that matches all old column values. Your function should set the Boolean flag to TRUE if it wants to discard these column values, and FALSE otherwise.

If you perform a delete at the local site and an update at the remote site, the remote site detects the delete conflict, but the local site detects an unresolvable update conflict. This type of conflict cannot be handled automatically. The conflict will raise 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.

Additional Information: See "Avoiding Delete Conflicts" on page 7-20.

Restrictions

You should avoid the following types of SQL commands in user-defined conflict resolution methods. Use of such commands can result in unpredictable results.

- Data Definition Language statements (for example, CREATE, ALTER, DROP).
- Transaction control statements (for example, COMMIT, ROLLBACK).
- Session control (for example, ALTER SESSION).
- System control (for example, ALTER SYSTEM).

Example User-Defined Conflict Resolution Method

The following examples show user-defined methods that are variations on the standard MAXIMUM and ADDITIVE prebuilt conflict resolution methods. Unlike the standard methods, these custom functions can handle nulls in the columns used to resolve the conflict.

Maximum User Function

```
-- User function similar to MAXIMUM method.
-- If curr is null or curr < new, use new values.
-- If new is null or new < curr, use current values.
-- If both are null, no resolution.
-- Does not converge with > 2 masters, unless
-- always increasing.

FUNCTION max_null_loses(old                 IN    NUMBER,
                        new                 IN  OUT  NUMBER,
                        cur                 IN    NUMBER,
                        ignore_discard_flag OUT    BOOLEAN)
  RETURN BOOLEAN IS
BEGIN
   IF (new IS NULL AND cur IS NULL) OR new = cur THEN
       RETURN FALSE;
   END IF;
   IF new IS NULL THEN
       ignore_discard_flag := TRUE;
   ELSIF cur IS NULL THEN
       ignore_discard_flag := FALSE;
   ELSIF new < cur THEN
       ignore_discard_flag := TRUE;
   ELSE
       ignore_discard_flag := FALSE;
   END IF;
   RETURN TRUE;
END max_null_loses;
```

Additive User Function

```
-- User function similar to ADDITIVE method.
-- If old is null, old = 0.
-- If new is null, new = 0.
-- If curr is null, curr = 0.
-- new = curr + (new - old) -> just like ADDITIVE method.

FUNCTION additive_nulls(old                 IN    NUMBER,
                        new                 IN  OUT  NUMBER,
                        cur                 IN    NUMBER,
                        ignore_discard_flag OUT    BOOLEAN)
   RETURN BOOLEAN IS
   old_val NUMBER := 0.0;
   new_val NUMBER := 0.0;
   cur_val NUMBER := 0.0;
BEGIN
   IF old IS NOT NULL THEN
      old_val := old;
   END IF;
   IF new IS NOT NULL THEN
      new_val := new;
   END IF;
   IF cur IS NOT NULL THEN
      cur_val := cur;
   END IF;
   new := cur_val + (new_val - old_val);
   ignore_discard_flag := FALSE;
   RETURN TRUE;
END additive_nulls;
```

User-Defined Conflict Notification Methods

A conflict notification method is a user-defined function that provides conflict notification rather than or in addition to conflict resolution. For example, you can write your own conflict notification methods to log conflict information in a database table, send an email message, or page an administrator. After you write a conflict notification method, you can assign it to a column group (or constraint) in a specific order so that Oracle notifies you when a conflict happens, before attempting subsequent conflict resolution methods, or after Oracle attempts to resolve a conflict but cannot do so.

To configure a replicated table with a user-defined conflict notification mechanism, you must complete the following steps:

1. Create a conflict notification log.
2. Create the user-defined conflict notification method in a package.

The following sections explain each step.

Creating a Conflict Notification Log

When configuring a replicated table to use a user-defined conflict notification method, the first step is to create a database table that can record conflict notifications. You can create a table to log conflict notifications for one or many tables in a master group.

To create a conflict notification log table at all master sites, use the replication execute DDL facility. For more information, "Executing DDL Within a Master Group" on page 7-2. 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 whic the
                             --- conflict occurred
 table_owner   VARCHAR2(30), --- owner of the table
 conflict_type VARCHAR2(6)   --- INSERT, DELETE or UNIQUE
)
```

Creating a Conflict Notification Package

To create a conflict notification method, you must define the method in a PL/SQL package and then replicate the package as part of a master group along with the associated replicated table.

A conflict notification method can perform conflict notification only, or both conflict notification and resolution. If possible, you should always use one of Oracle's prebuilt conflict resolution methods to resolve conflicts. When a user-defined conflict notification method performs only conflict notification, assign the user-defined method to a column group (or constraint) along with conflict resolution methods that can resolve conflicts.

Note: If Oracle cannot ultimately resolve a replication conflict, Oracle rolls back the entire transaction, including any updates to a notification table. If notification is necessary independent of transactions, you can design a notification mechanism to use the Oracle DBMS_PIPES package or the database interface to Oracle Office.

Sample Conflict Notification Package

The following package and package body perform a simple form of *conflict notification* by logging uniqueness conflicts for a CUSTOMERS table into the previously defined CONF_REPORT table.

Note: This example of *conflict notification* does not resolve any conflicts. You should either provide a method to resolve conflicts (for example, *discard* or *overwrite*), or provide a notification mechanism that will succeed (for example, using e-mail) even if the error is not resolved and the transaction is rolled back. With simple modifications, the following user-defined conflict notification method can take more active steps. For example, instead of just recording the notification message, the package can use the DBMS_OFFICE utility package to send an Oracle Office email message to an administrator.

```
CREATE OR REPLACE PACKAGE notify AS
 -- Report uniqueness constraint violations on CUSTOMERS table
 FUNCTION customers_unique_violation (
   first_name        IN OUT VARCHAR2,
   last_name         IN OUT VARCHAR2,
   discard_new_values IN OUT BOOLEAN)
 RETURN BOOLEAN;
END notify;
/

CREATE OR REPLACE PACKAGE BODY notify AS
 -- Define a PL/SQL table to hold the notification message
 TYPE message_table IS TABLE OF VARCHAR2(80) INDEX BY BINARY_INTEGER;

 PROCEDURE report_conflict (
   conflict_report IN MESSAGE_TABLE,
   report_length   IN NUMBER,
   conflict_time   IN DATE,
   conflict_table  IN VARCHAR2,
   table_owner     IN VARCHAR2,
   conflict_type   IN VARCHAR2) IS
 BEGIN
   FOR idx IN 1..report_length LOOP
     BEGIN
       INSERT INTO sales.conf_report
         (line, txt, timestamp, table_name, table_owner, conflict_type)
         VALUES (idx, SUBSTR(conflict_report(idx),1,80), conflict_time,
                 conflict_table, table_owner, conflict_type);
     EXCEPTION WHEN others THEN NULL;
     END;
   END LOOP;
 END report_conflict;

 -- This is the conflict resolution method that will be called first when
 -- a uniqueness constraint violated is detected in the CUSTOMERS table.
 FUNCTION customers_unique_violation (
   first_name  IN OUT VARCHAR2,
   last_nameIN OUT VARCHAR2,
   discard_new_valuesIN OUT BOOLEAN)
  RETURN BOOLEAN IS
   local_node  VARCHAR2(128);
   conf_report MESSAGE_TABLE;
   conf_time   DATE := SYSDATE;
  BEGIN
  -- Get the global name of the local site
```

```
    BEGIN
      SELECT global_name INTO local_node FROM global_name;
    EXCEPTION WHEN others THEN local_node := '?';
    END;
  -- Generate a message for the DBA
  conf_report(1) := 'UNIQUENESS CONFLICT DETECTED IN TABLE CUSTOMERS ON ' ||
                    TO_CHAR(conf_time, 'MM-DD-YYYY HH24:MI:SS');
  conf_report(2) := '  AT NODE ' || local_node;
  conf_report(3) := 'ATTEMPTING TO RESOLVE CONFLICT USING ' ||
                    'APPEND SEQUENCE METHOD';
  conf_report(4) := 'FIRST NAME: ' || first_name;
  conf_report(5) := 'LAST NAME:  ' || last_name;
  conf_report(6) := NULL;
  --- Report the conflict
  report_conflict(conf_report, 5, conf_time, 'CUSTOMERS',
                'OFF_SHORE_ACCOUNTS', 'UNIQUE');
  --- Do not discard the new column values. They are still needed by
  --- other conflict resolution Methods
  discard_new_values := FALSE;
  --- Indicate that the conflict was not resolved.
    RETURN FALSE;
  END customers_unique_violation;
END notify;
/
```

Viewing Conflict Resolution Information

Oracle provides replication catalog (REPCAT) views that you can use to determine what conflict resolution methods are being used by each of the tables and column groups in your replicated environment. Each view has three versions: USER_*, ALL_*, SYS.DBA_*. The views available include the following:

REPRESOLUTION_ METHOD	Lists all of the available conflict resolution methods.
REPCOLUMN_GROUP	Lists all of the column groups defined for the database.
REPGROUPED_COLUMN	Lists all of the columns in each column group in the database.
REPPRIORITY_GROUP	Lists all of the priority groups and site priority groups defined for the database.
REPPRIORITY	Lists the values and corresponding priority levels for each priority or site priority group.

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.
REPRESOLUTION	Shows more specific information about the conflict resolution method used to resolve conflicts on each object.
REPPARAMETER_ COLUMN	Shows which columns are used by the conflict resolution methods to resolve a conflict.

Additional Information: See "Data Dictionary Views" in the *Oracle8i Replication API Reference* book for details.

7

Administering a Replicated Environment

This chapter describes how to administer your replicated database environment. The topics include the following:

- Advanced Management of Master and Snapshot Groups.
- Monitoring an Advanced Replication System.
- Database Backup and Recovery in Replication Systems.
- Auditing Successful Conflict Resolution.
- Determining Differences Between Replicated Tables.
- Updating The Comments Fields in Views.

Advanced Management of Master and Snapshot Groups

Chapters 3 and 4 in this book discuss the most commonly performed procedures that you use to manage master and snapshot groups in an advanced replication environment. The following sections explain several less commonly used administrative procedures that involve the management of master and snapshot groups.

Executing DDL Within a Master Group

Replication Manager lets you propagate one or more SQL DDL statements to some or all of the master sites in a master group. This option lets you execute unique DDL that is not specifically supported within Oracle's replication management API. For example, you might want to create rollback segments and users that are necessary to support a replication environment.

Warning: Do not execute DDL that could damage global database integrity in a multimaster environment. For example, do not execute DDL statements to alter a replication object at any site.

To execute DDL at selected master sites in a master group:

1. Expand the database node that is the master definition site for the master group where you want to execute the DDL at.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Select the target master group that you want to execute the DDL at.

   The property page for the selected master group will appear in the right pane of the Replication Manager user interface.
5. Press the **Operations** tab.
6. Press the **Execute DDL** button.
7. Enter the DDL to be executed in the **Enter DDL Text** field.
8. Select the master sites where you want the DDL executed at (press the <SHIFT> key to select a range of master sites or press the <CTRL> key to individually select multiple master sites).
9. Press the **Execute Now** button to execute the specified DDL at the selected master sites.

API Equivalent: DBMS_REPCAT.EXECUTE_DDL

Relocating a Master Group's Definition Site

If the master definition site of a master group becomes unavailable or you simply want to relocate the master destination site for the group:

1. Expand the database node that contains the target master group.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Select the target master group that you want define a new master definition site for.

   The property page for the selected master group will appear in the right pane of the Replication Manager user interface.
5. Press the **Master Sites** tab.
6. Click **Change** to change the groups master definition site.
7. Use the **Change Masterdef** dialog to change the group's master definition site.

When you relocate the master definition site for a master group, you can choose to notify:

- All other master sites.
- The current master definition site—uncheck this setting when the current master definition site is unavailable.

API Equivalent: DBMS_REPCAT.RELOCATE_MASTERDEF

Changing a Snapshot Group's Master Site

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

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

In this example, the master site for the ACCT object group is changed to the DBS2 master site.

You must call this procedure at the snapshot site whose master site you want to change. The new database must be a master site in the replicated environment.

When you call this procedure, Oracle uses the new master to perform a full refresh of each snapshot in the local snapshot group.

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

Additional Information: See the "SWITCH_SNAPSHOT_ MASTER" procedure in the *Oracle8i Replication API Reference* book for details.

Monitoring an Advanced Replication System

Oracle uses its internal system of deferred transactions and job queues to propagate changes among the sites in an advanced replication system. It is important that you monitor regularly the internal workings of a replication environment to ensure that it is running smoothly. The following sections explain how you can use Oracle Replication Manager to view and manage administration requests, deferred transactions, error transactions, and job queues.

Managing Administration Requests

An administration request is a specific call to a procedure or function in Oracle's replication management API. For example, when you use Replication Manager to create a new master group, Replication Manager completes the task by making a call to the DBMS_REPCAT.CREATE_MASTER_REPGROUP procedure. All DDL changes to replication groups and the objects within generate administration requests. Many top-level administration requests generate additional replication management API calls to complete the request.

Administration requests are inserted by the master definition site in the administration request queues at all master sites as one distributed transaction. Each master site has a scheduled job, DO_DEFERRED_REPCAT_ ADMIN, that executes requests simultaneously at each site. As administration requests are processed, each site reports back to the master definition site. Oracle removes requests that complete successfully from the administration request queue at the master definition site. However, if any errors are encountered, the administration request remains in the master definition site's administration request queue with an error status.

Replication Manager allows you view and update the status of administration requests. The following sections explain more about how to manage administration requests for a master group in a multimaster advanced replication environment.

Displaying Administration Requests

To display the administration requests for a master group at a master site:

1. Expand the database node that you want to view the administrative requests for.
2. Expand the **Configuration** node.
3. Expand the **Master Groups** node.
4. Select the master group that you want to view the administrative requests for.
5. Press the **Operations** tab.

   All administrative requests for the selected master group will be displayed in the Administrative Requests list. The following is displayed for each administrative request:

   - The corresponding replication management API call for the request.
   - The current status of the request (for example, ready, error, awaiting callback, and so on).
   - The source database for the administration request.
   - Any errors associated with the administration request.

Replication Manager does not automatically update the display of the detail panel. To obtain a more current list of pending administration requests, refresh the display by selecting another tab and then reselecting the **Operations** tab.

Additional Information: You can also view administration requests by querying the REPCATLOG data dictionary view.

Applying Administration Requests As long as each instance in an Oracle advanced replication facility has one or more SNP processes, each server 1fgruns a job at a regular interval to execute all administration requests. If you do not want to wait for Oracle to execute administration requests, use Replication Manager to apply all pending administration requests manually.

1. Press the **Apply** button to apply the displayed administrative requests.

   - If you want to apply the administrative requests at all sites participating in the selected master group, enable the **As All Sites** checkbox before pressing the **Apply** button.

Note: Oracle automatically attempts to apply all administration requests synchronously among all master sites.

API Equivalent: DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN

Deleting Administration Requests Sometimes it is necessary to remove pending administration requests for a master group. For example, certain administration requests might return errors. Even after you resolve the corresponding error situation, administration requests remain in the server's queue unless you manually purge them.

To remove selected administration requests for a master group:

1. Select the administrative request or requests that you want to delete (press the <SHIFT> key to select a range of requests or press the <CTRL> key to individually select multiple requests).
2. Press the **Selected** button to delete the selected administrative requests.

To remove all administration requests for a master group:

1. Press the **All** button to delete all administrative requests for the selected master group.

API Equivalent: DBMS_REPCAT.PURGE_MASTER_LOG

More about Administration Request Mechanisms

When you use Replication Manager or make a call to a procedure in the DBMS_ REPCAT pac1fgage to administer an advanced replication system, Oracle uses its internal mechanisms to broadcast the request using synchronous replication. If a synchronous broadcast fails for any reason, Oracle returns an error message and rolls back the encompassing transaction.

When an Oracle Server receives an administration request, it records the request in the REPCATLOG view and the corresponding DDL statement in a child table of the REPCATLOG view. When you view administration requests for a master group at a master site, you might observe requests that are awaiting a callback from another master site. Whenever you use Replication Manager to create an administration request for a replication group, Oracle automatically inserts a job into the local job queue, if one does not already exist for the group. This job periodically executes the procedure DO_DEFERRED_REPCAT_ADMIN. Whenever you synchronously broadcast a request, Oracle attempts to start this job immediately in order to apply the replicated changes at each master site.

Assuming that Oracle does not encounter any errors, DO_DEFERRED_REPCAT_ADMIN will be run whenever a background process is available to execute the job. The initialization parameter JOB_QUEUE_INTERVAL determines how often the background process wakes up. You can experience a delay if you do not have enough background processes available to execute the outstanding jobs.

Note: When JOB_QUEUE_PROCESSES = 0 at a site, you must apply administration requests manually for all groups at the site. See "Applying Administration Requests" for more information.

For each call of DO_DEFERRED_REPCAT_ADMIN at a master site, the site checks the REPCATLOG view to see if there are any requests that need to be performed. When one or more administration requests are present, Oracle applies the request and updates any local views as appropriate. This event can occur asynchronously at each master site.

DO_DEFERRED_REPCAT_ADMIN executes the local administration requests in the proper order. When DO_DEFERRED_REPCAT_ADMIN is executed at a master that is not the master definition site, it does as much as possible. Some asynchronous activities such as populating a replicated table require communication with the master definition site. If this communication is not possible, DO_DEFERRED_REPCAT_ADMIN stops executing administration requests to avoid executing requests out of order. Some communication with the master definition site, such as the final step of updating or deleting an administration request at the master definition site, can be deferred and will not prevent DO_DEFERRED_REPCAT_ADMIN from executing additional requests.

The success or failure of an administration request at each master site is noted in the REPCATLOG view at each site. For each master group, Replication Manager displays the corresponding status of each administration request. Ultimately, each master site propagates the status of its administration requests to the master definition site. If a request completes successfully at a master site, Oracle removes the callback for the site from the REPCATLOG view at the master definition site. If the event completes successfully at all sites, all entries in the REPCATLOG view at all sites, including the master definition site, will be removed.

By synchronously broadcasting the change, Oracle ensures that all sites are aware of the change, and thus are capable of remaining in synch. By allowing the change to be applied at the site at a future point in time, Oracle provides you with the flexibility to choose the most appropriate time to apply changes at a site.

If an object requires automatically generated replication support, you must regenerate replication support after altering the object. Oracle then activates the internal triggers and regenerates the packages to support replication of the altered object at all master sites.

Note: Although the DDL must be successfully applied at the master definition site in order for these procedures to complete without error, this does not guarantee that the DDL is successfully applied at each master site. Replication Manager displays the status of all administration requests. Additionally, the REPCATLOG view contains interim status and any asynchronous error messages generated by the request.

Any snapshot sites that are affected by a DDL change are updated the next time you perform a refresh of the snapshot site. While all master sites can communicate with one another, snapshot sites can communicate only with their associated master site.

If you must alter the shape of a snapshot as the result of a change to its master, you must drop and re-create the snapshot.

Diagnosing Problems with Administration Requests

After you administer an advanced replication environment, you should take a look at the administration requests to make sure that everything is running smoothly. While monitoring your system, you might find that there are problems related to administration requests. The following list provides you with some troubleshooting ideas for administration request problems.

- If administration requests are backed up at a site and not because of an error condition, ensure that there is not a problem with the local job running DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN. See "Diagnosing Problems with Jobs" for more information about diagnosing job queue problems.

Note: Additionally, check the LOG_USER column in the DBA_JOBS view to ensure that the replication job is being run on behalf of the replication administrator. Check the USERID column of the DBA_REPCATLOG view to ensure that the replication administrator was the user that submitted the request. DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN only performs those administrative requests submitted by the user that calls this procedure.

- Ensure the relevant databases are running and communication is possible.
- Ensure that the necessary private database links are available for the advanced replication facility, and that the user designated in the CONNECT TO clause (generally the replication administrator) has the necessary privileges. If you used the Replication Manager setup wizard to setup your sites, you should not have any problems.

Managing Deferred Transactions

An advanced replication system using asynchronous data propagation uses an internal system of deferred transactions and job queues to push changes from one site to another. Replication Manager lets you view and manage deferred and error transactions queued at each server in an advanced replication system.

Displaying Deferred Transactions

To display a summary list of a replication site's deferred transactions by their outgoing destination:

1. Expand the database node that contains the deferred transactions that you want to view.
2. Expand the **Administration** node.
3. Expand the **Deferred Transactions by Dest** node.
4. Select the destination of the deferred transactions that you want to view.

   The deferred transaction property sheet will appear in the right pane of the Replication Manager user interface.
5. Press the **Transactions** tab.

All deferred transactions for the selected destination will be displayed in the body of the **Transactions** tab. Displayed information includes transaction ID, # of Calls, Delivery Order, and the Start Time.

To display the properties for an individual deferred transaction:

1. Expand the destination node that contains the deferred transaction that you want to view.
2. Select the individual deferred transaction.

   The general characteristics of the selected deferred transaction will be displayed in the right pane of the Replication Manager user interface.

To display the deferred calls for an individual deferred transaction:

1. Expand the target deferred transaction node.
2. Select the Deferred Call node.

   The transaction's Call #, Schema, Package, Procedure, and # of Arguments will be displayed in the right pane of the Replication Manager user interface.

Note: Replication Manager does not automatically update the display of the detail panel. To obtain a more current list of pending deferred transactions, refresh the display.

Additional Information: You can also view deferred transactions by querying the DEFCALL, DEFCALLDEST, DEFLOB, DEFSCHEDULE, DEFTRAN, and DEFTRANDEST views.

Executing Deferred Transactions

Each instance in an Oracle advanced replication facility that has one or more SNP processes and the necessary scheduled links automatically runs a job at a regular interval to execute deferred transactions automatically at targeted destinations. If you do not want to wait for Oracle to execute a deferred transaction, use Replication Manager to manually push all pending deferred transactions for a particular destination.

1. Expand the database node that contains the deferred transactions that you want to execute.
2. Expand the **Administration** node.

3. Expand the **Deferred Transactions by Dest** node.
4. Select the destination of the deferred transactions that you want to view.

   The deferred transaction property sheet will appear in the right pane of the Replication Manager user interface.
5. Press the **Transactions** tab.

   All deferred transactions for the selected destination will be displayed in the body of the **Transactions** tab. Displayed information includes transaction ID, # of Calls, Delivery Order, and the Start Time.
6. Press the **Push Transactions** button to execute the deferred transactions.

   The **Push Options** dialog box will appear.
7. Make any necessary adjustments to the **Push Options** dialog box and press the **Push** button.

API Equivalent: DBMS_DEFER_SYS.PUSH

Deleting Deferred Transactions

In some cases, you might know that a deferred transaction in the queue at your local site will cause an error transaction at the receiving site if pushed to the site. In such cases, it might be appropriate to delete the deferred transaction from the local queue to prevent an error transaction from happening.

Warning: See "Managing Error Transactions" for more information about error transactions and when to delete deferred transactions.

To delete an individual deferred transaction:

1. Expand the database node that contains the deferred transactions that you want to delete.
2. Expand the **Administration** node.
3. Expand the **Deferred Transactions by Dest** node.
4. Select the destination of the deferred transactions that you want to delete.

   The deferred transaction property sheet will appear in the right pane of the Replication Manager user interface.
5. Press the **Transactions** tab.

   All deferred transactions for the selected destination will be displayed in the body of the **Transactions** tab. Displayed information includes transaction ID, # of Calls, Delivery Order, and the Start Time.

6. Select the deferred transaction or transactions that you want to delete (press the <SHIFT> key to select a range of transactions or press the <CTRL> key to individually select multiple transactions).
7. Press the **Delete Selected** button.

You have several other options available when deleting deferred transactions:

- Enable the **Delete to all destinations** checkbox to delete the same transactions in deferred transaction queue for other destinations.
- Press the **Delete All** button to delete all displayed deferred transactions.

API Equivalent: DBMS_DEFER_SYS.DELETE_TRAN

Managing Error Transactions

When Oracle pushes a deferred transaction from a snapshot or master site to another master site, Oracle ensures that the transaction is not removed from the local queue until it has been successfully propagated to the remote site. However, a transaction can be successfully propagated to a master site without being successfully applied at the site. An error in applying a deferred transaction may be the result of a database problem, such as a lack of available space in a table that you are attempting to update, or may be the result of an unresolvable replication conflict. If an error occurs, Oracle performs the following actions at the receiving master site:

- Rolls back the transaction.
- Logs the error transaction in the receiving site’s replication catalog.

Displaying Error Transactions

You should frequently check each site in a replicated environment for error transactions and then resolve them. To display a summary list of a replication site’s local error transactions:

1. Expand the database node that contains the errors that you want to view.
2. Expand the **Administration** node.
3. Select the **Local Errors** node.

   All local errors will be displayed in the right pane of Replication Manager.

To resolve an error transaction properly, you need specific information about what the transaction is attempting to perform. Deferred transactions consist of a series of deferred remote procedure calls that must be applied in a given order to maintain transaction consistency.

To display more information about an individual error transaction:

1. Select the individual error transaction.

   The local error property sheet will be displayed in the right pane of the Replication Manager user interface.

To display the deferred calls for an individual error transaction:

1. Expand the individual error transaction.

   Review the transaction's deferred calls and call destinations, and the transaction's destinations.

Resolving Error Transactions

Once you have determined the cause of an error transaction, you may need to perform one or more of the following actions at the destination site after fixing the error:

- Re-execute the error transaction.
- Delete the error transaction from the local site.

Executing Error Transactions

When you have resolved the problem that caused an error transaction, you can re-execute the error transaction. If the error transaction executes successfully, Oracle automatically removes the transaction from the local site's replication catalog.

To re-execute an error transaction:

1. Right-click on the individual error transaction.
2. Select **Re-execute Transaction**.

API Equivalent: DBMS_DEFER_SYS.EXECUTE_ERROR.

When Oracle re-executes an error transaction at the receiving site, Oracle executes the transaction in the *security context of the original receiver*. If the original receiver is no longer a valid user (that is, if the user was dropped), the deferred transaction can be re-executed *under the security context of a different user* using the DBMS_ DEFER_ SYS.EXECUTE_ERROR_AS_USER procedure. See this procedure, which is described in the *Oracle8i Replication API Reference* book.

Attention: If you re-execute a single error transaction, Oracle does not commit the transaction, even if it executes successfully. If you are satisfied with the results of the transaction, you should issue the SQL command COMMIT WORK. If EXECUTE_ERROR re-executes multiple transactions, each transaction is committed as it completes.

Deleting Error Transactions

Sometimes, you must manually resolve an error transaction (in other words, not re-execute the transaction to resolve it). When you have resolved the problem that caused an error transaction, you should delete the error transaction. To delete an error transaction:

1. Right-click on the individual error transaction and select **Delete**.

API Equivalent: DBMS_DEFER_SYS.DELETE_ERROR.

Managing Local Jobs

Oracle runs jobs in each site's job queue to complete certain tasks automatically that are necessary to manage an advanced replication environment. For example, when you use Replication Manager to create and enable a scheduled link to a remote site, the local server places a job in its job queue that Oracle periodically runs to push local changes to a remote master.

Replication Manager has several features that you can use to view and manage the job queues of each server in an advanced replication system. The following sections explain more about viewing and managing a server's local jobs.

Displaying a Site's Local Jobs

Oracle creates a job in a site's local job queue on behalf of the following operations:

- When you create a scheduled link, Oracle creates a job to push deferred transactions to the target remote master site. The PL/SQL for this type of job will include the API call DBMS_DEFER_SYS.PUSH.
- When you schedule purging of a site's deferred transaction queue, Oracle creates a job to perform this operation. The PL/SQL for this type of job will include the API call DBMS_DEFER_SYS.PURGE.
- When you create a master group, Oracle creates a job to push corresponding administration requests to the other master sites that manage the group (one job per destination). The PL/SQL for this type of job will include the API call DBMS_REPCAT.DO_DEFERRED_ REPCAT_ADMIN.
- When you create a refresh group, Oracle creates a job to perform regular refreshes for the group. The PL/SQL for this type of job will include the API call DBMS_REFRESH.REFRESH.

To display a summary list of a replication site's local job queue:

1. Expand the database node that you want to view the local jobs for.
2. Expand the **Administration** node.
3. Select the **Local Jobs** node.

The detail panel of Replication Manager lists summary information for all local jobs.

To display the properties for an individual local job:

1. Expand the **Local Jobs** node.
2. Select the individual job that you want to view the details for.

The pages of the **Job** property sheet let you view the various properties of the job, including:

- The job's next execution date and execution interval.
- The job's PL/SQL text.
- The most recent execution date for the job.
- The current status of the job (normal or broken).
- The number of failed executions for the job.
- The user environment under which the job executes.

Editing the Properties of a Local Job In certain situations, you might want to edit the properties of a local job. For example, the default execution interval for a master group's administration requests is 10 minutes; the only way to edit this setting is to edit the scheduling properties for the corresponding job. To edit the properties of a local job:

1. Select the target job.

   The properties of the selected job will be displayed in the right pane of the Replication Manager user interface.

The **Edit Job** property sheet lets you edit the following properties of a job:

- The job's next execution date and execution interval.
- The job's PL/SQL text.

Manually Running a Job Rather than wait for the next execution date for a job, you can force a job to run. To run a job immediately:

1. Select the target job.

   The local job property sheet will be displayed in the right pane of the Replication manager user interface.

2. Press the **Run Now** button.

Breaking and Enabling a Job In certain cases, you might need to temporarily disable (break) a job and then later enable the job again. Or, you might need to enable a broken job after fixing a problem the prevented the job from running properly. For example, if you accidentally drop a database link upon which a job depends, the job will fail and eventually break (after 16 successive failures). After you recreate the necessary database link, you can then enable the broken job. To break or enable (make normal) a local job:

1. Select the target job.

   The local job property sheet will be displayed in the right pane of the Replication manager user interface.

2. Enable the **Make Broken** checkbox.
3. Press the **Apply** button.

Diagnosing Problems with Jobs

Many operations in an advanced replication environment rely on jobs to perform work. While monitoring your system, you might find that there are problems related to jobs. You should use Replication Manager to monitor local jobs regularly.

If a job fails to execute, first check to see the status of the job. If the job is broken, check the alert log and trace files for error information and fix the error. If the job is not broken (normal), Oracle should ultimately re-execute it.

If the job is OK but has never executed, there may be a problem with the availability of SNP background processes. Check the initialization parameter JOB_QUEUE_PROCESSES to determine the number of background processes available and JOB_QUEUE_INTERVAL to determine how frequently each background processes wakes up. You can also query the DBA_JOBS_RUNNING view to explore what jobs these processes are currently running (you may have a problem with a runaway job), and the alert log and trace file can provide you with additional information about potential problems with the background process.

Database Backup and Recovery in Replication Systems

Databases using advanced replication are distributed databases. Follow the guidelines for distributed database backups outlined in the *Oracle8i Administrator's Guide* when creating backups of advanced replication databases. Follow the guidelines for coordinated distributed recovery in the *Oracle8i Administrator's Guide* when recovering an advanced replication database.

If you fail to follow the coordinated distributed recovery guidelines, there is no guarantee that your advanced replication databases will be consistent. For example, a restored master site may have propagated different transactions to different masters. You may need to perform extra steps to correct for an incorrect recovery operation. One such method is to drop and recreate all replicated objects in the recovered database.

Recommendation: Remove pending deferred transactions and deferred error records from the restored database, and resolve any outstanding distributed transactions *before* dropping and recreating replicated objects. If the restored database was a master definition site for some replicated environments, you should designate a new master definition site before dropping and creating objects. Any snapshots mastered at the restored database should be fully refreshed, as well as any snapshots in the restored database.

To provide continued access to your data, you may need to change master definition sites (assuming the database being recovered was the master definition site), or remaster snapshot sites (assuming their master site is being recovered).

Performing Checks on Imported Data

After performing an export/import of a replicated object or an object used by the advanced replication facility (for example, the REPSITES view), you should run the REPCAT_IMPORT_CHECK procedure in the DBMS_REPCAT package.

In the following example, the procedure checks the objects in the ACCT replicated object group at a snapshot site to ensure that they have the appropriate object identifiers and status values:

```
DBMS_REPCAT.REPCAT_IMPORT_CHECK( gname    =>   'acct',
                                 master   =>   FALSE);
```

Additional Information: See the "REPCAT_IMPORT_CHECK" procedure in the *Oracle8i Replication API Reference* book for details.

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 REPRESOLUTION_STATISTICS view. This view is updated only if you have chosen to turn on conflict resolution statistics gathering for the table involved in the conflict.

Gathering Conflict Resolution Statistics

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

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

Additional Information: See the "REGISTER_STATISTICS" procedure in the *Oracle8i Replication API Reference* book for details.

Viewing Conflict Resolution Statistics

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

Additional Information: See the "REPRESOLUTION_STATISTICS" view and the "DEFERROR" view in the *Oracle8i Replication API Reference* book for details.

Canceling Conflict Resolution Statistics

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

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

Additional Information: See the "CANCEL_STATISTICS" procedure in the *Oracle8i Replication API Reference* book for details.

Deleting Statistics Information

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

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

```
DBMS_REPCAT.PURGE_STATISTICS(sname      =>  'acct_rec',
                             oname      =>  'emp',
                             start_date =>   '01-JAN-95',
                              end_date   =>   '31-MAR-95);
```

Additional Information: See the "PURGE_STATISTICS" procedure in the *Oracle8i Replication API Reference* book for details.

Determining Differences Between Replicated Tables

When administering a replicated environment, you may periodically want to check 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 release 7.3 or higher:

DIFFERENCES

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

RECTIFY

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

To restore equivalency between all copies of a replicated table, you should 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, you need to create two tables to hold the results of the comparison.

   You must create one table that can hold the data for the columns being compared. For example, if you decide to compare the EMPNO, SAL, and BONUS columns of the EMPLOYEE table, your CREATE statement would need to be similar to the one shown below.

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

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

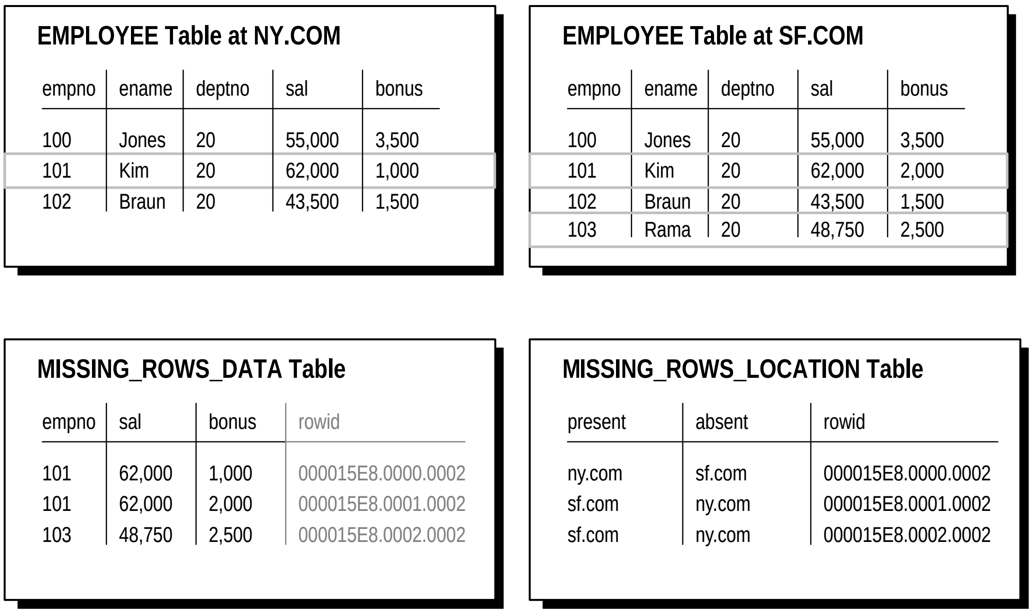
   ```
   CREATE TABLE missing_rows_location
   (
     present     VARCHAR2(128),
     absent      VARCHAR2(128),
     r_id        ROWID
   )
   ```

4. Suspend replication activity for the object group containing the tables that you want to compare. Although suspending replication activity for the group is not a requirement, rectifying tables that were not quiesced first can result in inconsistencies in your data.
5. At the site containing the "reference" table, call the DBMS_RECTIFIER_DIFF.DIFFERENCES procedure. For example, if you wanted to compare the EMPLOYEE tables at the New York and San Francisco sites, your procedure call would look similar to the following:

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

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

Figure 7–1 Determining Differences Between Replicas



EMPLOYEE Table at NY.COM

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

EMPLOYEE Table at SF.COM

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

MISSING_ROWS_DATA Table

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

MISSING_ROWS_LOCATION Table

present	absent	rowid
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. Now you can rectify the table at the "comparison" site to be equivalent to the table at the "reference" site by calling the DBMS_RECTIFIER_DIFF.RECTIFY procedure as shown in the following example:

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

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

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

7. After you have successfully executed the RECTIFY procedure, your missing rows tables should be empty. You can now 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. You may now resume replication activity for the master group.

Managing Snapshot Logs

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

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

Altering Snapshot Logs

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

Required Privileges Only the owner of the master table or a user with the SELECT privilege for the master table can alter a snapshot log.

Altering Snapshot Log Storage Parameters To alter a snapshot log's storage parameters, use the Storage and Options pages of the Snapshot Log property sheet or an equivalent ALTER SNAPSHOT LOG statement. For example:

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

Altering a Snapshot Log to Add Filter Columns To add new filter columns to a snapshot log, use the SQL command ALTER SNAPSHOT LOG. For example:

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

Managing Snapshot Log Space

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

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

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

- One snapshot is not configured for automatic refreshes and has not been manually refreshed for a long time.
- One snapshot has an infrequent refresh interval, such as every year (365 days).

- A network failure has prevented an automatic refresh of one or more of the snapshots based on the master table.
- A network or site failure has prevented a master from becoming aware that a snapshot has been dropped.

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

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

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

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

Additional Information: See the DBMS_SNAPSHOT. PURGE_LOG procedure in the *Oracle8i Replication API Reference* book for details.

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

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

1. Acquire an exclusive lock on the master table to prevent updates from occurring during the following process.

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

2. Using a second database session, copy the rows in the snapshot log (in other words, the MLOG$ base table) to a temporary table.

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

3. Using the second session, truncate the log using the SQL command TRUNCATE.

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

4. Using the second session, reinsert the old rows so that you do not have to perform a complete refresh of the dependent snapshots.

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

5. Using the first session, release the exclusive lock on the master table.

   ```
   ROLLBACK;
   ```

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

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

Reorganizing Master Tables that Have Snapshot Logs

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

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

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

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

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

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

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

Examples The following two statements preserve snapshot log information when the master table is truncated:

```
TRUNCATE TABLE tablename PRESERVE SNAPSHOT LOG;
TRUNCATE TABLE tablename;
```

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

```
TRUNCATE TABLE tablename PURGE SNAPSHOT LOG
```

Methods of Reorganizing a Database Table Oracle provides four table reorganization methods that preserve the capability for fast refresh; these appear under the following headings. Other reorg methods require an initial complete refresh to enable subsequent fast refreshes.

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

Method 1 for Reorganizing Table t

1. Call dbms_snapshot.begin_table_reorganization for table t.
2. Rename table t to t_old.
3. Create table t as select * from t_old.
4. Call dbms_snapshot.end_table_reorganization for new table t.

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

Ensure that no transaction is issued against the reorganized table between calling dbms_snapshot.begin_table_reorganization and dbms_snapshot.end_table_reorganization.

Method 2 for Reorganizing Table t

1. Call dbms_snapshot.begin_table_reorganization for table t.
2. Export table t.
3. Truncate table t with PRESERVE SNAPSHOT LOG option.
4. Import table t using conventional path.
5. Call dbms_snapshot.end_table_reorganization for new table t.

Warning: When you truncate master tables as part of a reorg, you must use the PRESERVE SNAPSHOT LOG clause of the truncate table DDL.

Ensure that no transaction is issued against the reorganized table between calling dbms_snapshot.begin_table_reorganization and dbms_snapshot.end_table_reorganization.

Method 3 for Reorganizing Table t

1. Call dbms_snapshot.begin_table_reorganization for table t.
2. Export table t.
3. Rename table t to t_old.
4. Import table t using conventional path.
5. Call dbms_snapshot.end_table_reorganization for new table t.

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

Ensure that no transaction is issued against the reorganized table between calling dbms_snapshot.begin_table_reorganization and dbms_snapshot.end_table_reorganization.

Method 4 for Reorganizing Table t

1. Call dbms_snapshot.begin_table_reorganization for table t.
2. Select contents of table t to a flat file.
3. Rename table t to t_old.
4. Create table t with the same shape as t_old.
5. Run SQL*Loader using conventional path.
6. Call dbms_snapshot.end_table_reorganization for new table t.

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

Ensure that no transaction is issued against the reorganized table between calling dbms_snapshot.begin_table_reorganization and dbms_snapshot.end_table_reorganization.

Dropping Snapshot Logs

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

- All snapshots of a master table have been dropped.
- All snapshots of a master table are to be completely refreshed, not fast refreshed.

To drop a snapshot log, complete the following:

1. Expand the database node that contains the snapshot log that you want to delete.
2. Expand the **Configuration** node.
3. Expand the **Snapshot Logs** node.
4. Expand the schema that contains the snapshot log that you want to delete.
5. Right-click on the snapshot log that you want to delete and select Remove.
6. Press the **Yes** button to confirm the deletions.

You can also execute the equivalent DROP SNAPSHOT LOG SQL statement in SQL*Plus:

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

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

Troubleshooting Common Problems

The following sections contain troubleshooting guidelines for managing an advanced replication environment.

Diagnosing Problems with Database Links

If you think a database link is not functioning properly, you can drop and recreate it using Oracle Enterprise Manager, Oracle Server Manager, or another tool.

- Make sure that the scheduled interval is what you want.
- Make sure that the scheduled interval is not shorter than the required execution time.

If you used a connection qualifier in a database link to a given site, the other sites that link to that site must have the exact same connection qualifier. For example, if you create a database link as follows:

```
CREATE DATABASE LINK myethernet CONNECT TO repsys IDENTIFIED BY secret
USING 'connect_string_myethernet'
```

All the sites, whether masters or snapshots, associated with the `myethernet` database link must include the `'connect_string_myethernet'` connect string.

Diagnosing Problems with Master Sites

There are a number problems that might arise in a multimaster advanced replication system. The next few sections discuss some problems and ways to troubleshoot them.

Replicated Objects Not Created at New Master Site

If you add a new master site to a master group, and the appropriate objects are not created at the new site, try the following:

- Ensure that the necessary private database links exist between the new master site and the existing master sites. If you used the Replication Manager setup wizard to setup your sites, you should not have any problems. You must have links both to the new site from each existing site, and from the new site to each existing site.
- Make sure that the administration requests at all sites have completed successfully. If requests have not been executed yet, you can manually execute pending administration requests to complete the operation immediately.

DDL Changes Not Propagated to Master Site

If you create a new master group object or alter the definition of a master group object at the master definition site and the modification is not propagated to a master site, first ensure that the administration requests at all sites have completed successfully. If requests are pending execution, you can manually execute them to complete the operation immediately.

When you execute DDL statements through the replication API, Oracle executes the statements on behalf of the user who submits the DDL. When a DDL statement applies to an object in a schema other than the submitter's schema, the submitter needs appropriate privileges to execute the statement. In addition, the statement must explicitly name the schema. For example, assume that you, the replication administrator, supply the following as the DDL_TEXT parameter to the DBMS_ REPCAT.CREATE_MASTER_ REPOBJECT procedure:

```
CREATE TABLE scott.new_emp AS SELECT * FROM hr.emp WHERE ...;
```

Because each table name contains a schema name, this statement works whether the replication administrator is SCOTT, HR, or another user, as long as the administrator has the required privileges.

Suggestion: Qualify the name of every schema object with the appropriate schema.

DML Changes Not Asynchronously Propagated to Other Sites

If you make an update to your data at a master site, and that change is not properly asynchronously propagated to the other sites in your replicated environment, try the following:

- Use Replication Manager to check whether the corresponding deferred transaction has been pushed to the destination. If not, you can also check to see how much longer it will be before the scheduled link pushes the queue to the destination site. If you do not want to wait for the next scheduled push across a link, you can execute deferred transaction manually. See "Executing Deferred Transactions" on page 7-10.
- If a scheduled link's interval has passed and corresponding deferred transactions have not been pushed, check the corresponding job for the link. See "Displaying a Site's Local Jobs" on page 7-14.
- Even after propagating a deferred transaction to a destination, it might not execute because of an error. To learn more about error transactions, see "Managing Error Transactions" on page 7-12.

DML Cannot be Applied to Replicated Table

If you receive the DEFERRED_RPC_QUIESCE exception when you attempt to modify a replicated table, one or more master groups at your local site are "quiescing" or "quiesced". To proceed, your replication administrator must resume replication activity for the group.

Bulk Updates and Constraint Violations

A single update statement applied to a replicated table can update zero or more rows. The update statement causes zero or more update requests to be queued for deferred execution, one for each row updated. This distinction is important when constraints are involved, because Oracle effectively performs constraint checking at the end of each statement. While a bulk update might not violate a uniqueness constraint, for example, some equivalent sequence of individual updates might violate uniqueness.

If the ordering of updates is important, update one row at a time in an appropriate order. This lets you define the order of update requests in the deferred RPC queue.

Re-Creating a Replicated Object

If you add an object such as a package, procedure, or view to a master group, the status of the object must be VALID. If the status of an object is INVALID, recompile the object, or drop and recreate the object before adding it to a master group.

Unable to Generate Replication Support for a Table

When you generate replication support for a table, Oracle activates an internal trigger at the local site. If the table will be propagating changes asynchronously, this trigger uses the DBMS_DEFER package to build the calls that are placed in the local deferred transaction queue. EXECUTE privileges for most of the packages involved with advanced replication, such as DBMS_REPCAT and DBMS_DEFER, need to be granted to replication administrators and users that own replicated objects. The Replication Manager setup wizard and the DBMS_REPCAT_ADMIN package performs the grants needed by the replication administrators for many typical replication scenarios. When the owner of a replicated object is not a replication administrator, however, you must explicitly grant EXECUTE privilege on DBMS_DEFER to the object owner.

Problems with Replicated Procedures or Triggers

If you discover an unexpected unresolved conflict, and you were mixing procedural and row-level replication on a table, carefully review the procedure to ensure that the replicated procedure did not cause the conflict. Ensure that ordering conflicts between procedural and row-level updates are not possible. Check if the replicated procedure locks the table in EXCLUSIVE mode before performing updates (or uses some other mechanism of avoiding conflicts with row-level updates). Check that row-level replication is disabled at the start of the replicated procedure and re-enabled at the end. Ensure that row-level replication is re-enabled even if exceptions occur when the procedure executes. In addition, check to be sure that the replicated procedure executed at all master sites. You should perform similar checks on any replicated triggers that you have defined on replicated tables.

Diagnosing Problems with the Deferred Transaction Queue

If deferred transactions at a site are not being pushed to their destinations, there can be several reasons for the problem. The following sections explain some possible causes.

Check Jobs for Scheduled Links

When you create a scheduled link, Oracle adds a corresponding job to the site's job queue. If you have scheduled a link to push deferred transactions at a periodic interval, and you encounter a problem, you should first be certain that you are not experiencing a problem with the job queue.

Distributed Transaction Problems

When Oracle pushes a deferred transaction to a remote site using serial propagation, it uses a distributed transaction to ensure that the transaction has been properly committed at the remote site before the transaction is removed from the queue at the local site. For information on diagnosing problems with distributed transactions (two-phase commit), see the book *Oracle8i Distributed Database Systems*.

Incomplete Database Link Specifications

If you notice that transactions are not being pushed to a given remote site, you may have a problem with how you have specified the destination for the transaction. When you create a scheduled link, you must provide the full database link name. If you use Replication Manager, you should not have any problems.

Incorrect Replication Catalog Views

Having the wrong view definitions can lead to erroneous deferred transaction behavior. The DEFCALLDEST and DEFTRANDEST views are defined differently in CATDEFER.SQL and CATREPC.SQL. The definitions in CATREPC.SQL should be used whenever advanced replication is used. If CATDEFER.SQL is ever (re)loaded, ensure that the view definitions in CATREPC.SQL are subsequently loaded.

Diagnosing Problems with Snapshots

There are a number problems that might happen with snapshot sites in an advanced replication system. The next few sections discuss some problems and ways to troubleshoot them.

Problems Creating Replicated Objects at Snapshot Site

If you unsuccessfully attempt to create a new object at a snapshot site, try the following:

- For an updatable snapshot, check that the associated master table has a snapshot log.
- Make sure that you have the necessary privileges to create the object. See "Privileges" on page 3-27 for more information.
- If you are trying to add an existing snapshot to a snapshot group, try recreating the snapshot when you add it to the group.

Troubleshooting Refresh Problems

The following sections explain several common snapshot refresh problems.

Common Problems Several common factors can prevent the automatic refresh of a group of snapshots:

- The lack of an SNP background process at the snapshot database.
- An intervening network or server failure.
- An intervening server shutdown.

When a snapshot refresh group is experiencing problems, ensure that none of the above situations is preventing Oracle from completing group refreshes.

Automatic Refresh Retries When Oracle fails to refresh a group automatically, the group remains due for its refresh to complete. Oracle will retry an automatic refresh of a group with the following behavior:

- Oracle retries the group refresh first one minute later, then two minutes later, four minutes later, and so on, with the retry interval doubling with each failed attempt to refresh the group.
- Oracle does not allow the retry interval to exceed the refresh interval itself.
- Oracle retries the automatic refresh up to sixteen times.

If after sixteen attempts to refresh a refresh group Oracle continues to encounter errors, Oracle considers the group broken. The General page of the Refresh Group property sheet in Schema Manager indicates when a refresh group is broken. You can also query the BROKEN column of the USER_REFRESH and USER_REFRESH_CHILDREN data dictionary views to see the current status of a refresh group.

The errors causing Oracle to consider a snapshot refresh group broken are recorded in a trace file. After you correct the problems preventing a refresh group from refreshing successfully, you must refresh the group manually. Oracle then resets the broken flag so that automatic refreshes can happen again.

Additional Information: The name of the snapshot trace file is of the form SNP*n*, where *n* is platform specific. See your platform-specific Oracle documentation for the name on your system.

Snapshots Continually Refreshing If you encounter a situation where Oracle continually refreshes a group of snapshots, check the group's refresh interval. Oracle evaluates a group's automatic refresh interval before starting the refresh. If a group's refresh interval is less than the amount of time it takes to refresh all snapshots in the group, Oracle continually starts a group refresh each time the SNP background process checks the queue of outstanding jobs.

Snapshot Logs Growing Without Bounds If a snapshot log is growing without bounds, check to see whether a network or site failure has prevented a master from becoming aware that a snapshot has been dropped. You may need to purge part of the log as described in "Purging Rows from a Snapshot Log" and unregister the snapshot.

Advanced Troubleshooting of Refresh Problems

If you have a problem refreshing a snapshot, try the following:

- Check the NEXT value in the DBA_SNAPSHOTS view to determine if the refresh has been scheduled.
- If the refresh interval has passed, check the DBA_REFRESH view for the associated job number for the snapshot refresh and then follow the instructions for diagnosing a problem with job queues in "Managing Local Jobs".
- You may also encounter an error if you attempt to define a master detail relationship between two snapshots. You should define master detail relationships only on the master tables by using declarative referential integrity constraints; the related snapshots should then be placed in the same refresh group to preserve this relationship.
- If you encounter a situation where your snapshots are being continually refreshed, you should check the refresh interval that you specified. This interval

is evaluated before the snapshot is refreshed. If the interval that you specify is less than the amount of time it takes to refresh the snapshot, the snapshot will be refreshed each time the SNP background process checks the queue of outstanding jobs.

- If there are any outstanding conflicts recorded at the master site for the snapshots, you can only refresh the snapshots by setting the parameter REFRESH_AFTER_ERRORS to TRUE. This parameter can be set when you create or alter a snapshot refresh group. (There is a corresponding parameter for Replication Manager property sheets.)
- If your snapshot logs are growing too large, see "Managing Snapshot Logs" on page 7-22.
- Snapshots in the same refresh groups will have their rows updated in a single transaction. Such a transaction can be very large, requiring either a large rollback segment at the snapshot site (with the rollback segment specified to be used during refresh) or you will need to use more frequent refreshes to reduce the transaction size.
- If Oracle error ORA-12004 occurs, the master site may have run out of rollback segments when trying to maintain the snapshot log, or the snapshot log may be out of date. For example, the snapshot log may have been purged or recreated. See "Managing Snapshot Logs" on page 7-22.
- Complete refreshes of a single table internally use the TRUNCATE feature to increase speed and reduce rollback segment requirements. However, until the snapshot refresh is complete, users may temporarily see no data in the snapshot. Refreshes of multiple snapshots (for example, refresh groups) do not use the TRUNCATE feature.
- Reorganization of the master table (for example, to reclaim system resources) should TRUNCATE the master table to force ROWID snapshots to do complete refreshes, otherwise, the snapshots will have incorrect references to master table ROWIDs. For more information, see "Reorganizing Master Tables that Have Snapshot Logs" on page 7-25.

Registering a Snapshot at its Master Site

At the master site, an Oracle database automatically registers information about the snapshots based on the master tables. The following sections explain more about Oracle’s snapshot registration mechanism.

Viewing Information about Registered Snapshots You can use the General page of the Snapshot Log property sheet in Schema Manager to list the snapshots associated with a snapshot log. For additional information, you can query the DBA_ REGISTERED_SNAPSHOTS data dictionary view to list the following information about a remote snapshot:

- The owner, name, and database that contains the snapshot.
- The snapshot's defining query.
- Other snapshot characteristics, such as its refresh method (fast or complete).

You can also join the DBA_REGISTERED_SNAPSHOT view with the DBA_ SNAPSHOT_LOGS view at the master site to obtain the last refresh times for each snapshot. Administrators can use this information to monitor snapshot activity from master sites and coordinate changes to snapshot sites if a master table needs to be dropped, altered, or relocated.

Internal Mechanisms Oracle automatically registers a read-only snapshot at its master database when you create the snapshot, and unregisters the snapshot when you drop it.

Note: Oracle7 master sites cannot register snapshots.

Caution: Oracle cannot guarantee the registration or unregistration of a snapshot at its master site during the creation or drop of the snapshot, respectively. If Oracle cannot successfully register a snapshot during creation, Oracle completes snapshot registration during a subsequent refresh of the snapshot. If Oracle cannot successfully unregister a snapshot when you drop the snapshot, the registration information for the snapshot persists in the master database until it is manually unregistered.

Manual Snapshot Registration If necessary, you can maintain registration manually. Use the REGISTER_SNAPSHOT and UNREGISTER_SNAPSHOT procedures of the DBMS_SNAPSHOT package at the master site to add, modify, or remove snapshot registration information.

Additional Information: The REGISTER_SNAPSHOT and UNREGISTER_SNAPSHOT procedures are described in chapter 9, "Data Dictionary Views" in the *Oracle8i Replication API Reference* manual.

Updating The Comments Fields in Views

There are several procedures in the DBMS_REPCAT package that allow you to update the comment information in the various 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	Additional Information

REPGROUP	`COMMENT_ON_REPGROUP(`
	`gname IN VARCHAR2,`
	`Comment IN VARCHAR2)`
		The parameters for the COMMENT_ ON_REPGROUP procedure are described in the *Oracle8i Replication API Reference* book.

REPOBJECT	`COMMENT_ON_REPOBJECT(`
	`sname IN VARCHAR2,`
	`oname IN VARCHAR2,`
	`type IN VARCHAR2,`
	`comment IN VARCHAR2)`
		The parameters for the COMMENT_ON_REPOBJECT procedure are described in the *Oracle8i Replication API Reference* book.

REPSITES	`COMMENT_ON_REPSITES(`
	`gname IN VARCHAR2,`
	`master IN VARCHAR,`
	`comment IN VARCHAR2)`
		The parameters for the COMMENT_ ON_REPSITES procedure are described in the *Oracle8i Replication API Reference* book.

REPCOLUMN_ GROUP	`COMMENT_ON_COLUMN_GROUP(`
	`sname IN VARCHAR2,`
	`oname IN VARCHAR2,`
	`column_group IN VARCHAR2,`
	`comment IN VARCHAR2)`
		The parameters for the COMMENT_ ON_COLUMN_GROUP procedure are described in the *Oracle8i Replication API Reference* book.

REPPRIORITY_ GROUP	`COMMENT_ON_PRIORITY_GROUP(`
	`gname IN VARCHAR2,`
	`pgroup IN VARCHAR2)`
	`comment IN VARCHAR2)`
		The parameters for the COMMENT_ ON_PRIORITY_GROUP procedure are described in the *Oracle8i Replication API Reference* book.

REPPRIORITY_ GROUP (site priority group)	`COMMENT_ON_SITE_PRIORITY(`
	`gname IN VARCHAR2,`
	`name IN VARCHAR2,`
	`comment IN VARCHAR2)`
		The parameters for the COMMENT_ ON_SITE_PRIORITY procedure are described in the *Oracle8i Replication API Reference* book.

Table 7–1 Updating Comments in Advanced Replication Facility Views

View	DBMS_REPCAT Procedure	Additional Information
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 the *Oracle8i Replication API Reference* book.
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 the *Oracle8i Replication API Reference* book.
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 the *Oracle8i Replication API Reference* book.

8 Advanced Techniques

This chapter describes several advanced techniques that you can use in implementing an Oracle replicated database environment:

- Using Procedural Replication.
- Using Synchronous Data Propagation.
- Designing for Survivability.
- Snapshot Cloning and Offline Instantiation.
- Security Setup for Multimaster Replication.
- Security Setup for Snapshot Replication.
- Avoiding Delete Conflicts.
- Using Dynamic Ownership Conflict Avoidance.
- Modifying Tables without Replicating the Modifications.

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 replicated 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 per 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 "Avoiding Delete Conflicts".

Restrictions on Procedural Replication

All parameters for a replicated procedure must be IN parameters; OUT and IN / OUT modes are not supported. The datatypes supported for these parameters are: NUMBER, DATE, VARCHAR2, CHAR, ROWID, RAW, BLOB, CLOB, NCHAR, NVARCHAR, and NCLOB.

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 will help 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 "Modifying Tables without Replicating the Modifications".
- Only one replicated procedure should be executed at a time, as described in the next section, "Serialization of Transactions".
- Deferred transactions should be propagated serially. For more information, see "Guidelines for Scheduled Links".
- 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).

Serialization of Transactions

Serial execution ensures that your data remains consistent. The advanced 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, 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 will be 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 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

Additional Information: See "Disabling the Advanced Replication Facility".

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, 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 boolean parameters determine where the procedure gets executed. When CALL_LOCAL is TRUE, the procedure is executed locally. When CALL_REMOTE is TRUE, the procedure will ultimately be executed at all other master sites in the replicated 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 FALSE, and CALL_REMOTE is TRUE.

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.

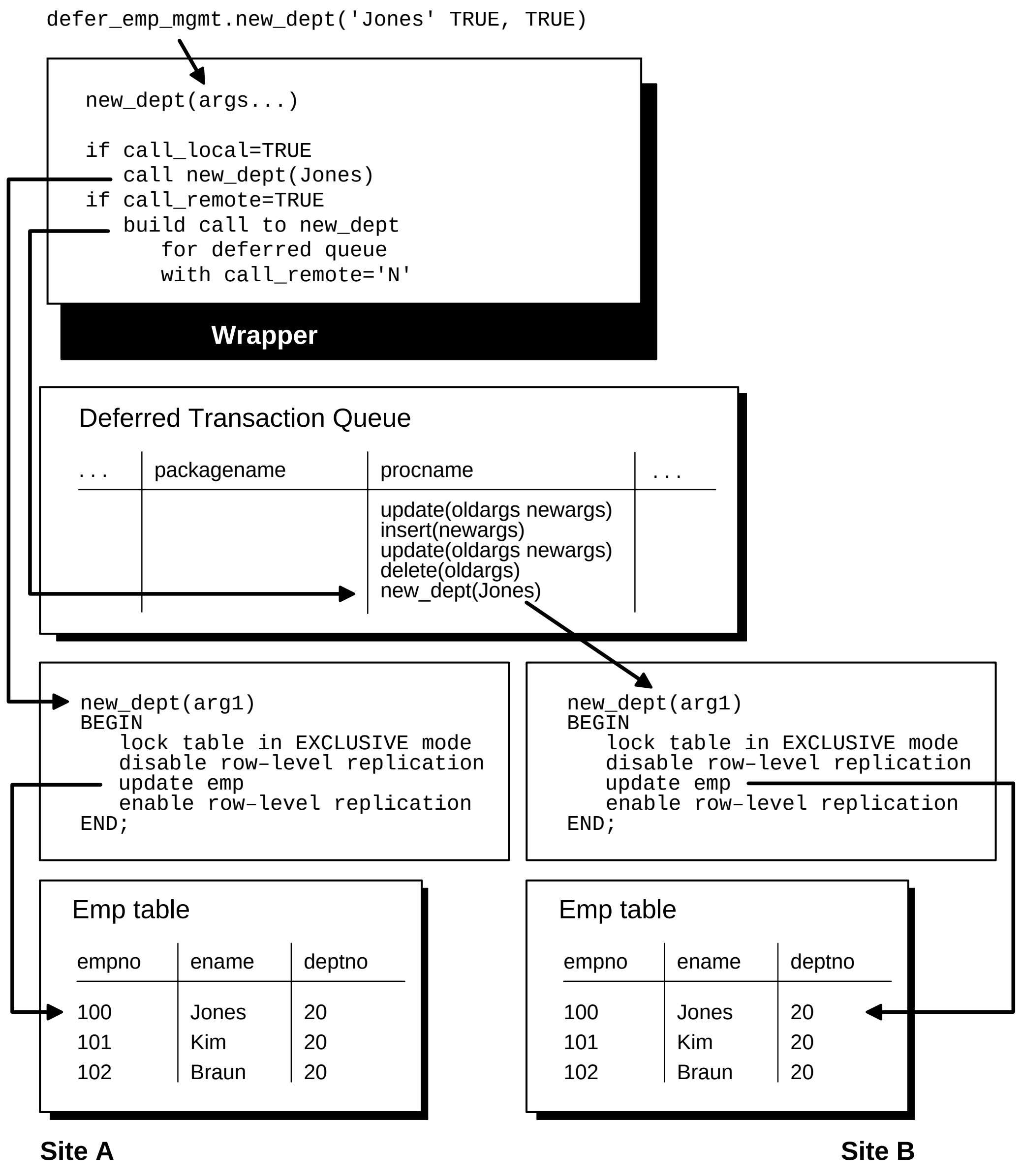
For example, suppose you create the package EMP_MGMT containing the procedure NEW_DEPT, which takes one argument, ENAME. To replicate this package to all master sites in your system, you can use Replication Manager to add the package to a master group and then generate replication support for the object. See "Managing Master Groups" for more information about managing master groups and replicated objects using Replication Manager. After completing these steps, an application can call procedure in the replicated package as follows:

```
defer_emp_mgmt.new_dept( ename        => 'Jones',
                        call_local    => TRUE,
                        call_remote  => TRUE);
```

As shown in Figure 8–1, 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 replicated environment with static partitioning of data ownership (that is, if you are not preventing row-level replication), the replication facility will preserve the order of operations at the remote node, since both row-level and procedural replication use the same asynchronous queue.

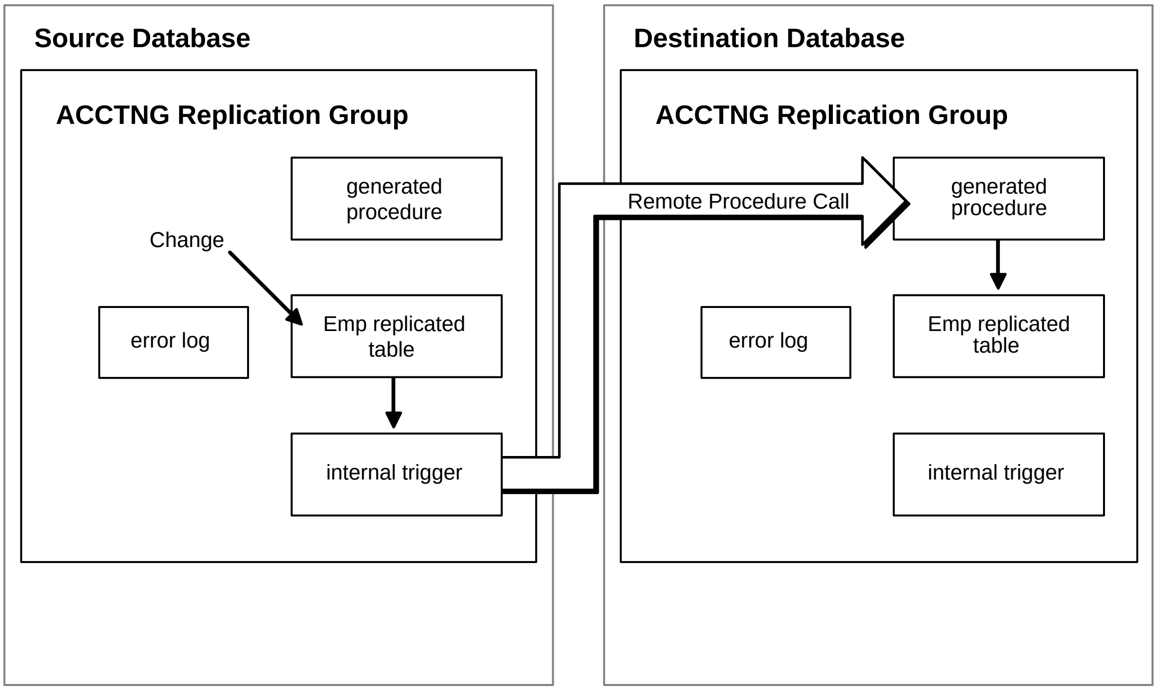
Figure 8–1 Asynchronous Procedural Replication



Using Synchronous Data Propagation

Asynchronous data propagation is the normal configuration for advanced replication environments. However, Oracle also supports synchronous data propagation for applications with special requirements. *Synchronous data propagation* occurs when an application updates a local replica of a table, and within the same transaction also updates all other replicas of the same table. Consequently, synchronous data replication is also called *real-time data replication*. Use synchronous replication only when applications require that replicated sites remain continuously synchronized.

Figure 8–2 Synchronous Data Replication Mechanisms

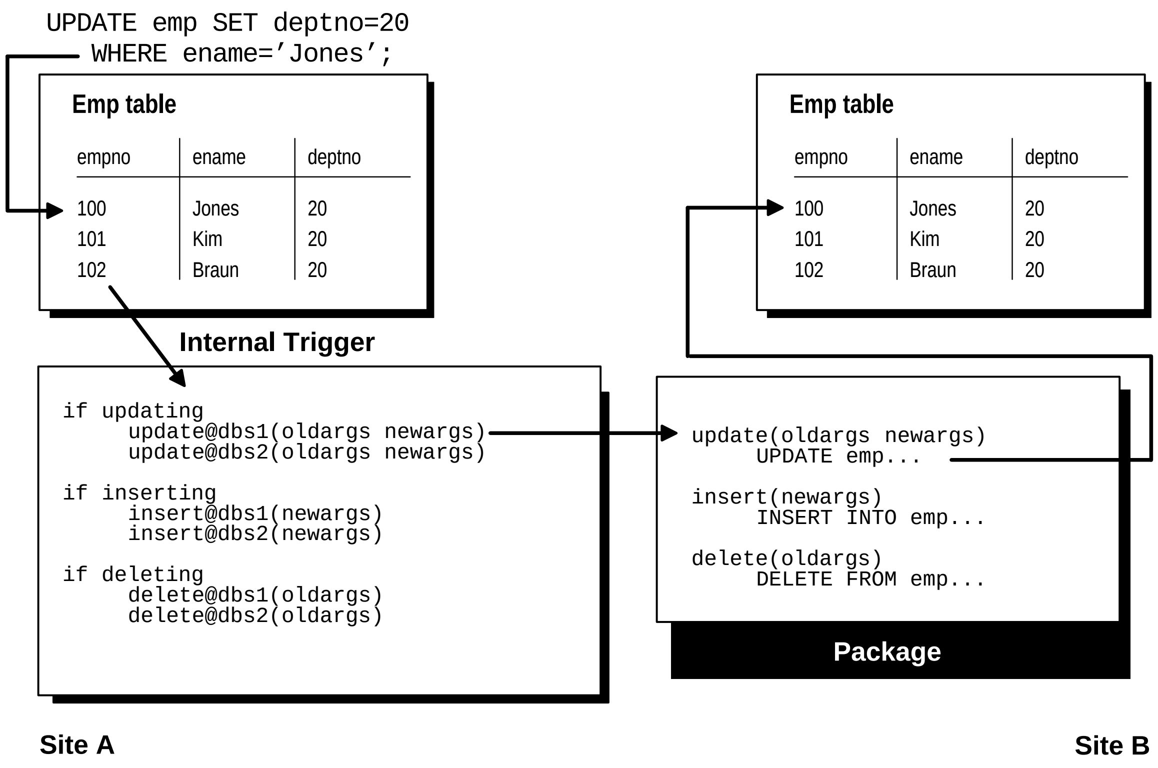


As Figure 8–2 shows, Oracle uses the same system of internal database triggers to generate RPCs that asynchronously replicate data-level changes to other replication sites to support synchronous, row-level data replication. However, Oracle does not defer the execution of such RPCs. Instead, data replication RPCs execute within the boundary of the same transaction that modifies the local replica. Consequently, a data-level change must be possible at all sites that manage a replicated table or else a transaction rollback occurs.

Understanding Synchronous Data Propagation

As shown in Figure 8–3, whenever an application makes a DML change to a local replicated table and the replication group is using synchronous row-level replication, the change is synchronously propagated to the other master sites in the replicated environment using internal triggers. When the application applies a local change, the internal triggers issue calls to generated procedures at the remote master sites *in the security context of the replication propagator*. Oracle ensures that all distributed transactions either commit or rollback in the event of a failure. See the discussion of distributed updates in the book *Oracle8i Distributed Database Systems*.

Figure 8–3 Propagating Changes using Synchronous Row-Level Replication


```
UPDATE emp SET deptno=20
   WHERE ename='Jones';
```

empno	ename	deptno
100	Jones	20
101	Kim	20
102	Braun	20

```
if updating
     update@dbs1(oldargs newargs)
     update@dbs2(oldargs newargs)

if inserting
     insert@dbs1(newargs)
     insert@dbs2(newargs)

if deleting
     delete@dbs1(oldargs)
     delete@dbs2(oldargs)
```

empno	ename	deptno
100	Jones	20
101	Kim	20
102	Braun	20

```
update(oldargs newargs)
     UPDATE emp...

insert(newargs)
     INSERT INTO emp...

delete(oldargs)
     DELETE FROM emp...
```

Restrictions

Because of the locking mechanism used by synchronous replication, deadlocks can occur. When an application performs a synchronous update to a replicated table, Oracle first locks the local row and then uses an AFTER ROW trigger to lock the corresponding remote row. Oracle releases the locks when the transaction commits at each site.

Note: A replication system that uses real-time propagation of replication data is highly dependent on system and network availability because it can function only when all sites in the system are concurrently available.

Destination of Synchronously Replicated Transactions

The necessary remote procedure calls to support synchronous replication are included in the internal trigger for each object. When you generate replication support for a replicated object, Oracle activates the triggers at all master sites to add the necessary remote procedure calls for the new site. Conversely, when you remove a master site from a master group, Oracle removes the calls from the internal triggers.

Conflict Detection

If all sites of a master group communicate synchronously with one another, applications should never experience replication conflicts. However, if even one site is sending changes asynchronously to another site, applications can experience conflicts at any site in the replicated environment.

If the change is being propagated synchronously, an error is raised and a rollback will be required. If the change is propagated asynchronously, Oracle automatically detects the conflicts and either logs the conflict or, if you designate an appropriate resolution method, resolves the conflict.

Additional Information: See Chapter 6, "Conflict Resolution".

Adding New Sites to an Advanced Replication Environment

When you add a new master or snapshot site for to a replication group in an advanced replication environment, Replication Manager allows you to select the data propagation mode (method) for the new site.

- A new master site’s data propagation mode determines how the site both sends changes to and receives changes from all other master sites.
- A new snapshot site’s data propagation mode determines how it sends changes to it’s master site.

See Chapter 2, "Using Multimaster Replication" and Chapter 5, "Directly Create Snapshot Environment" for more information about adding master and snapshot sites to an advanced replication environment, respectively.

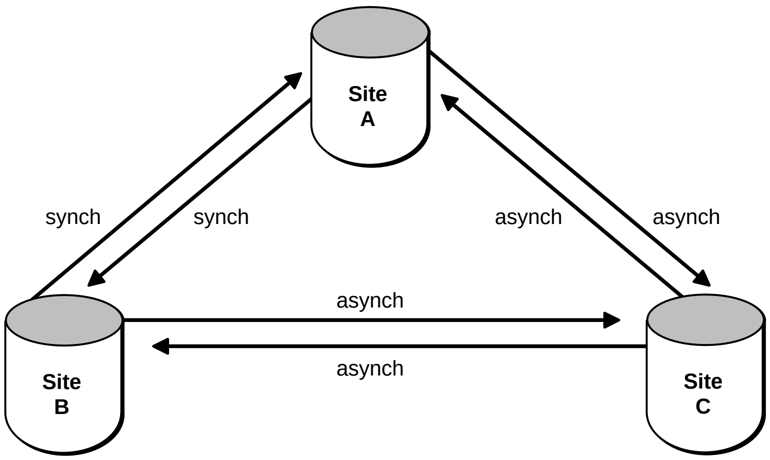
Understanding Mixed-Mode Multimaster Systems

In some situations, you might decide to have a mixed-mode environment in which some master sites propagate a master group’s changes asynchronously and others propagate changes synchronously. The order in which you add new master sites to a group with different data propagation modes can be important.

For example, suppose that you have three master sites: A, B, and C. If you first create site A as the master definition site, and then add site B with a synchronous propagation mode, site A will send changes to site B synchronously and site B will send changes to site A synchronously. There is no need to concern yourself with the scheduling of links at either site, because neither site is creating deferred transactions.

Now suppose that you create master site C with an asynchronous propagation mode. The propagation modes are now as illustrated in Figure 8–4.

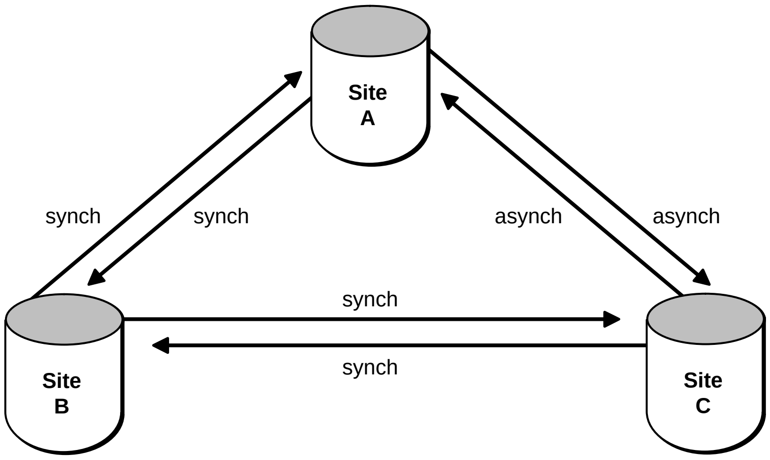
Figure 8–4 Selecting a Propagation Mode



You must now schedule propagation of the deferred transaction queue from site A to site C, from site B to site C, and from site C to sites A and B.

As another example, consider what would happen if you created site A as the master definition site, then added site C with an asynchronous propagation mode, then added site B with a synchronous propagation mode? Now the propagation modes would be as shown in Figure 8–5.

Figure 8–5 Ordering Considerations



Each time that you add a new master site to a mixed-mode multimaster system, consider how the addition will affect the data propagation modes to and from existing sites.

Tip: You can view the data propagation modes between master group sites in a multimaster system quickly by using a Replication Manager destination map. See "Displaying a Destination Map for a Master Group" for more information about master group destination maps.

Altering a Master Site's Data Propagation Mode

To change the data propagation mode from one master site to another in a master group, use the destination map for the group in Replication Manager.

1. Suspend replication activity for the master group.
2. Right-click the link that you want to alter.

3. Click **Make Asynchronous** or **Make Synchronous**.
4. Resume replication activity for the master group.

API Reference: DBMS_REPCAT.ALTER_MASTER_PROPAGATION

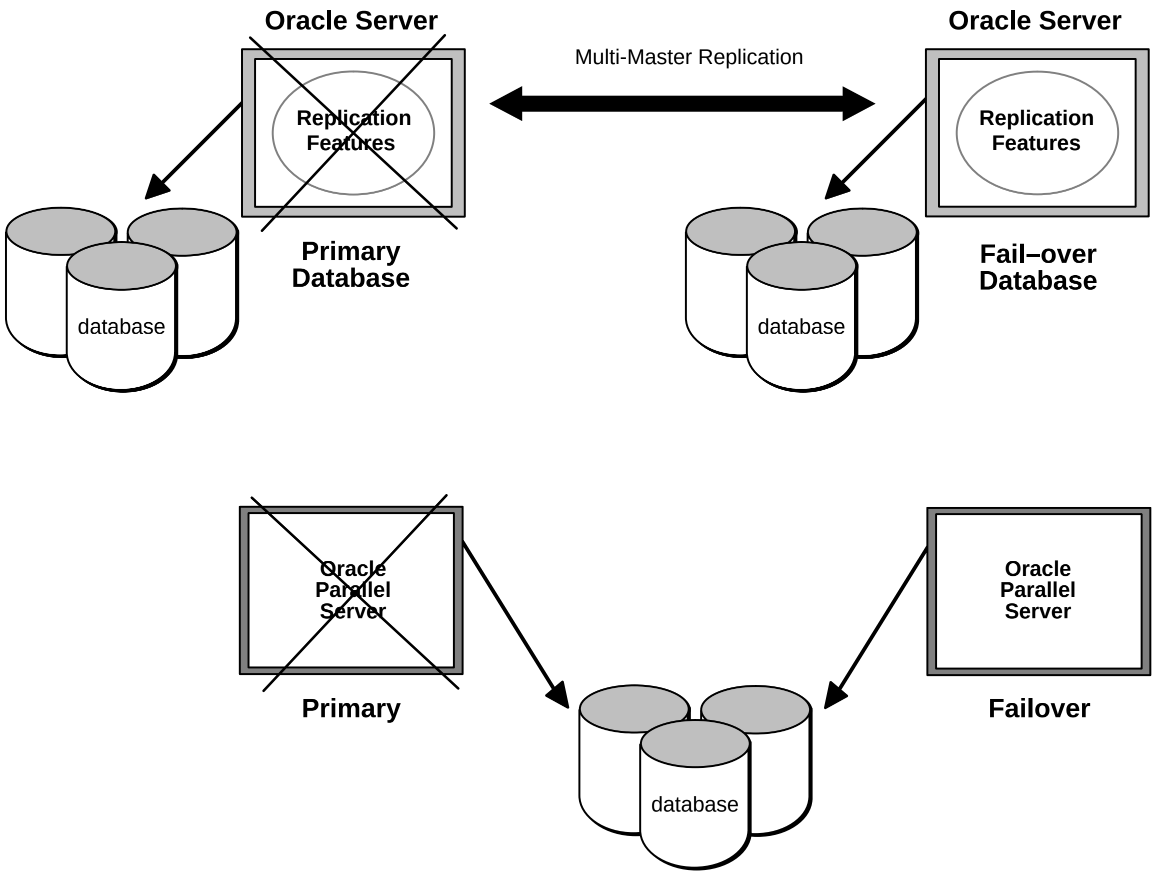
After you switch the propagation mode between one or more master sites in a master group:

- It is not necessary to regenerate replication support for any master group objects in Oracle8 or greater databases.
- You must regenerate replication support for all master group objects in Oracle7 databases.
- If you switch from synchronous to asynchronous, be sure to schedule the propagation and purging for deferred transaction queues at the affected sites.

Designing for Survivability

Survivability provides the capability to continue running applications despite system or site failures. Survivability allows you to run applications on a fail-over system, accessing the same, or very nearly the same, data as these systems accessed on the primary system when it failed. As shown in Figure 8–6, the Oracle Server provides two different technologies for accomplishing survivability: the Oracle Parallel Server and the advanced replication facility.

Figure 8–6 Survivability Methods: Advanced Replication vs. Parallel Server



Oracle Parallel Server versus Advanced Replication

The Oracle Parallel Server supports fail-over to surviving systems when a system supporting an instance of the Oracle Server fails. The Oracle Parallel Server requires a cluster or massively parallel hardware platform, and thus is applicable for protection against processor system failures in the local environment where the cluster or massively parallel system is running.

In these environments, the Oracle Parallel Server is the ideal solution for survivability — supporting high transaction volumes with no lost transactions or data inconsistencies in the event of an instance failure. If an instance fails, a surviving instance of the Oracle Parallel Server automatically recovers any incomplete transactions. Applications running on the failed system can execute on the fail-over system, accessing all data in the database.

The Oracle Parallel Server does not, however, provide survivability for site failures (such as flood, fire, or sabotage) that render an entire site, and thus the entire cluster or massively parallel system, inoperable. To provide survivability for site failures, you can use the advanced replication facility to maintain a replica of a database at a geographically remote location.

Should the local system fail, the application can continue to execute at the remote site. Advanced replication, however, cannot guarantee the protection of all transactions. Also, special care must be taken to prevent data inconsistencies when recovering the primary site.

Note: You can also configure a standby-database to protect an Oracle database from site failures. For more information about Oracle's standby database feature, see the *Oracle8i Backup and Recovery Guide*.

Designing for Survivability

If you choose to use the advanced replication facility for survivability, you should consider the following issues:

- The advanced replication facility must be able to keep up with the transaction volume of the primary system. This is application specific, but generally much lower than the throughput supported if you are using the Oracle Parallel Server.
- If a failure occurs at the primary site, recently committed transactions at the primary site may not have been asynchronously propagated to the fail-over site yet. These transactions will appear to be lost.
- These "lost" transactions must be dealt with when the primary site is recovered.

  Suppose, for example, you are running an order-entry system that uses replication to maintain a remote fail-over order-entry system, and the primary system fails.

  At the time of the failure, there were two transactions recently executed at the primary site that did not have their changes propagated and applied at the fail-over site. The first of these was a transaction that entered a new order, and the second was a transaction that cancelled an existing order.

In the first case, someone may notice the absence of the new order when processing continues on the fail-over system, and re-enter it. In the second case, the cancellation of the order may not be noticed, and processing of the order may proceed; that is, the canceled item may be shipped and the customer billed.

What happens when you restore the primary site? If you simply push all of the changes executed on the fail-over system back to the primary system, you will encounter conflicts.

Specifically, there will be duplicate orders for the item originally ordered at the primary system just before it failed. Additionally, there will be data changes resulting from the transactions to ship and bill the order that was originally canceled on the primary system.

You must carefully design your system to deal with these situations. The next section explains this process.

Implementing a Survivable System

Oracle's advanced replication facility can be used to provide survivability against site failures by using multiple replicated master sites. You must configure your system using one of the following methods. These methods are listed in order of increasing implementation difficulty.

- The fail-over site is used for read access only. That is, no updates are allowed at the fail-over site, even when the primary site fails.
- After a failure, the primary site is restored from the fail-over site using export/import, or via full backup.
- Full conflict resolution is employed for all data/transactions. This requires careful design and implementation. You must ensure proper resolution of conflicts that can occur when the primary site is restored, such as duplicate transactions.
- Provide your own special applications-level routines and/or procedures to deal with the inconsistencies that occur when the primary site is restored, and the queued transactions from the active fail-over system are propagated and applied to the primary site.

Snapshot Cloning and Offline Instantiation

By default, Oracle builds and populates replicas when you:

- Create a snapshot in a basic replication environment.
- Add master site or snapshot site to an advanced replication environment.

When building a large replicated environment, the amount of data necessary to build and populate replicas throughout the system can generate an excessive amount of network traffic. To avoid saturating the network with the data necessary to build a large replicated environment, Oracle lets you perform offline instantiation of new sites in both basic and advanced replication systems. The following sections explain how to clone snapshots in a basic replication environment and offline instantiate master and snapshot sites in an advanced replication environment.

Snapshot Cloning for Basic Replication Environments

To reduce the network overhead associated with the creation of the same set of snapshots in many databases, you can perform snapshot "cloning" by following these steps:

1. Create the snapshots and corresponding refresh groups that you want to clone in a database separate from the database that contains the master tables. The database in which you create snapshots of the master table data is the template snapshot database for the cloning process. See Chapter 3, "Snapshot Concepts & Architecture" for more information about creating snapshots.
2. At the template snapshot database, export the schemas containing the snapshots and refresh groups that you want to clone to other snapshot databases. The Export utility does not support the export of individual snapshots. Therefore, you must export snapshots at the schema level.
3. Prepare all other snapshot databases for snapshots of the master database. If you are not familiar with setting up the accounts and database links necessary to prepare a snapshot database, see "Prepare for Snapshots" on page 3-24.

Note: To make sure that you have prepared a snapshot database properly, connect to each snapshot schema in the database. Next, execute the defining queries of the proposed snapshots to ensure that they execute without error. Additionally, check to make sure that each new snapshot database has enough free space to hold the new snapshots.

4. At each new snapshot database, import the snapshot schemas that were exported from the template snapshot database. This creates and populates the same snapshots that were exported from the template snapshot database.
5. Perform a fast refresh of all fast-refreshable snapshots. Doing so registers the new snapshots in the master database.

Offline Instantiation of a Master Site in an Advanced Replication System

Offline instantiation of a master site lets you add the new master site to a master group while limiting the amount of downtime required for existing sites in the multimaster replication system. Offline instantiation of a master site is useful primarily for systems with very large databases where the time required to transfer the data through network links to a new site would be prohibitive.

Assuming that you are using a default replication configuration that uses asynchronous row-level replication, the steps necessary to create a new master site using offline instantiation are as follows:

1. Prepare the new master site with the Replication Manager setup wizard. At the new master site, make sure to create the schemas that will contain objects of the master group. If you are not sure how to prepare a master site, see "The Replication Setup Wizard" and "Adding a Master Site to a Master Group" for more information.
2. Synchronize the master sites that replicate the master group. This process includes pushing of all outstanding administration requests at each master site, resolving any administration requests that have an error status, resolving error transactions at each master site, and so on.
3. Suspend replication activity for the master group.

   Warning: Do not resume replication activity or do other replication administration for the master group until the new master site appears at the master definition site. Otherwise, changes that you make at any site will not be propagated to the new site, and you might have problems synchronizing the group's data.
4. From the group's master definition site, call the replication management API procedure DBMS_OFFLINE_OG.BEGIN_INSTANTIATION with the parameters *gname* (the name of the master group) and *new_site* (the name of the new master site). This procedure adds the new site to the master group.
5. Export the individual objects in the master group from any master site. When a master group contains replication objects that originate from more than one schema, make sure to export the objects from *each* schema.
6. From the master definition site, call the replication management API procedure DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS with the parameters *gname* (the name of the master group) and *new_site* (the name of the new master site).

At this point, normal non-administrative activities can resume at the existing master sites. However, replication activity for the group remains suspended at the new site.

7. Transfer the export file to the new master site.
8. At the new master site, call the replication management API procedure DBMS_OFFLINE_OG.BEGIN_LOAD with the parameters *gname* (the name of the master group) and *new_site* (the name of the new master site). This procedure ensures that applications cannot update master group objects while you import master group data in the next step, disables internal replication triggers, and disables propagation of the deferred RPC queue from all other master sites to the new site.
9. Use the Import utility to import the data from the Export file.
10. When the import is complete at the new master site, call the replication management API procedure DBMS_OFFLINE_OG.END_LOAD (*gname*, *new_site*) to enable the new site.
11. Return to the master definition site and call the procedure DBMS_OFFLINE_OG.END_INSTANTIATION with the parameters *gname* (the name of the master group) and *new_site* (the name of the new master site). This procedure resumes replication activity at the new site.

Additional Information: See the book *Oracle8i Utilities* to learn about the Import and Export utilities.

Please note the following:

- When you add multiple new sites, you must perform the above steps for each new site. You cannot instantiate a number of new sites as a group.
- Be careful to call the listed procedures from the appropriate site. Failure to do so could cause unexpected results.

Offline Instantiation of a Snapshot Site in an Advanced Replication System

Offline instantiation of a snapshot site in an advanced replication environment is useful when you need to create a large snapshot site, and the time required to transfer replication data through network to the new site would be prohibitive.

Use the following steps to create a snapshot site using offline instantiation:

At the Master Site **1.** Before you begin, create a snapshot log for each master table.

2. Create a snapshot of each master group table. Create each snapshot in the same schema as its master table. To accomplish this, each snapshot's name must be different from its corresponding master table. Additionally, the snapshot's defining query must use a loopback database link to reference the master table. For example, to create a snapshot EMPLOYEE of the EMP table in the same database DBS1:

   ```
   CREATE SNAPSHOT sales.employee AS SELECT * FROM sales.emp@dbs1
   ```

   Attention: Before creating snapshots, make sure that the master database has ample storage space.

3. As schema owner, use Export to export all schemas that contain the snapshots.
4. Drop the snapshots at the master site that were created for offline instantiation.
5. Transfer the Export file(s) to the new snapshot site.

At a New Snapshot Site

1. Using the Replication Manager setup wizard, prepare the new snapshot site to communicate with the master site. If you are not familiar with this process, see "Setting Up Snapshot Site" on page 5-2.
2. Using the Replication Manager snapshot group wizard, create an empty snapshot group to correspond with the master group. Do not choose to replicate any objects of the master group.
3. For each schema and snapshot, use the procedure DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (*gname, sname, master_site, snapshot_oname*) to create empty snapshots in the specified schema and object group at the new snapshot site, as well as all the necessary supporting objects.
4. Import only the snapshot base tables from the Export file(s).
5. When the import is complete, for each schema and snapshot, use the procedure DBMS_OFFLINE_SNAPSHOT.END_LOAD (*gname, sname, snapshot_oname*).

Additional Information: See the book *Oracle8i Utilities.*

Security Setup for Multimaster Replication

Nearly all users should find it easiest to use the Replication Manager setup wizard when configuring multimaster replication security. However, for certain cases you may need to use the replication management API to perform these setup operations.

To configure a replication environment, the database administrator must connect with DBA privileges to grant the necessary privileges to the replication administrator.

First set up user accounts at each master site with the appropriate privileges to configure and maintain the replication environment and to propagate and apply replicated changes. You must also define links for users at each master site.

In addition to the end users who will 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 8–1 on on page 8-21 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 will also need to grant certain privileges directly.

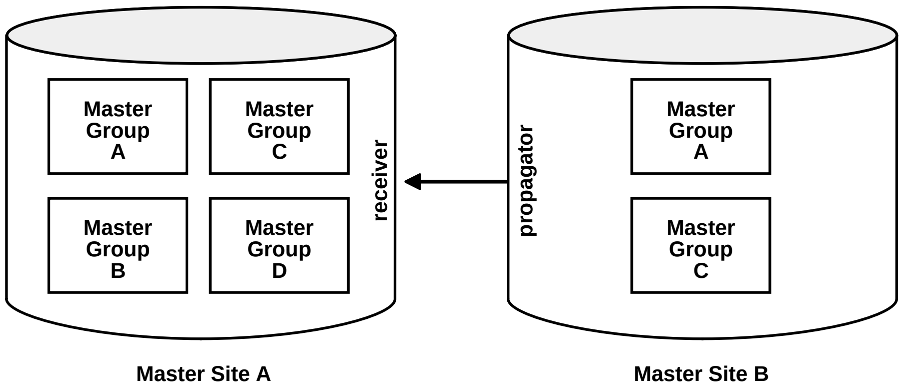
Trusted vs. Untrusted Security

In addition to the different users, you also need to determine which type of security model you will implement: trusted or untrusted. With a trusted security model, the receiver has access to ALL local master groups. Since 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 8–7. Even though only Master Groups A and C exist at Master Site B, the propagator has access to Master Groups A, B, C, and D at Master Site A because the trusted security model has been used. While this greatly increases the flexibility of database administration (due to the mobility of remote database administration), it also increases the chances of a malicious user at a remote site viewing or corrupting data at the master site.

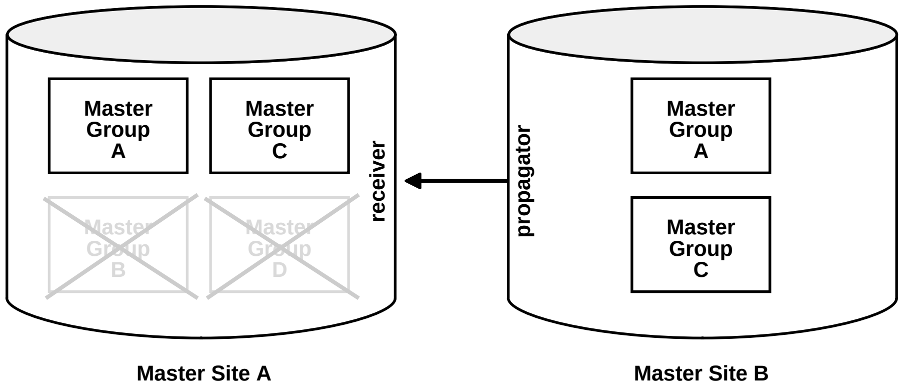
Regardless of the security model used, Oracle8*i* automatically grants the appropriate privileges for objects as they are added to or removed from a replicated environment.

Figure 8–7 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, will only be able to access the specified master groups that are local to the receiver. Figure 8–8 illustrates an untrusted security model. Since 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 propagators access at Master Site A.

Figure 8–8 Untrusted Security: Multimaster Replication



Typically, master sites are considered trusted and therefore the trusted security model is used. If, however, your remote master site(s) are untrusted, you may want to use the untrusted model and assign your receiver limited privileges (a site might be considered untrusted, for example, if a consulting shop performs work for multiple customers). Use the appropriate API call listed for the receiver in Table 8–1 to assign the different users the appropriate privileges.

Table 8–1 Required User Accounts

User	Privileges
global replication administrator	DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA
schema-level replication administrator	DBMS_REPCAT_ADMIN.GRANT_ADMIN_SCHEMA
propagator	DBMS_DEFER_SYS.REGISTER_PROPAGATOR

receiver	See the "DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP" procedure in the *Oracle8i Replication API Reference* book 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 will need to 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 will need to call

DBMS_DEFER_SYS.SCHEDULE_PUSH multiple times at each site, once for each remote location.

A sample script for setting up multimaster replication between HQ.WORLD and SALES.WORLD is shown below:

```
/*--- Create global replication administrator at HQ ---*/
connect system/manager@hq.world
create user repadmin identified by repadmin
execute dbms_repcat_admin.grant_admin_any_schema(username => 'repadmin')

/*--- Create global replication administrator at Sales ---*/
connect system/manager@sales.world
create user repadmin identified by repadmin
execute dbms_repcat_admin.grant_admin_any_schema(username => 'repadmin')

/*--- Create single user to act as both propagator and receiver at HQ ---*/
connect system/manager@hq.world
create user prop_rec identified by prop_rec
/*--- Grant privileges necessary to act as propagator ---*/
execute dbms_defer_sys.register_propagator(username => 'prop_rec')
/*--- Grant privileges necessary to act as receiver ---*/
execute dbms_repcat_admin.register_user_repgroup(
        username => 'prop_rec',
        privilege_type => 'receiver',
        list_of_gnames => NULL)

/*--- Create single user to act as both propagator and receiver at Sales ---*/
connect system/manager@sales.world
create user prop_rec identified by prop_rec
/*--- Grant privileges necessary to act as propagator ---*/execute
dbms_defer_sys.register_propagator(username => 'prop_rec')
/*--- Grant privileges necessary to act as receiver ---*/
execute dbms_repcat_admin.register_user_repgroup(
        username => 'prop_rec',
        privilege_type => 'receiver',
        list_of_gnames => NULL)

/*--- Create public link from HQ to Sales with necessary USING clause ---*/
connect system/manager@hq.world
create public database link sales.world using sales.world

/*--- Create private repadmin to repadmin link ---*/
connect repadmin/repadmin@hq.world
create database link sales.world connect to repadmin identified by repadmin
```

```
/*--- Schedule replication from HQ to Sales ---*/
execute dbms_defer_sys.schedule_push(
     destination => 'sales.world',
     interval => 'sysdate + 1/24',
     next_date => sysdate,
     stop_on_error => FALSE,
     parallelism => 1)

/*--- Schedule purge of def tran queue at HQ ---*/
execute dbms_defer_sys.schedule_purge(
     next_date => sysdate,
     interval = 'sysdate + 1',
     delay_seconds => 0,
     rollback_segment =>'')

/*--- Create link from propagator to receiver for scheduled push ---*/
connect prop_rec/prop_rec@hq.world
create database link sales.world connect to prop_rec identified by prop_rec

/*--- Create public link from Sales to HQ with necessary USING clause
---*/connect system/manager@sales.world
create public database link hq.world using hq.world

/*--- Create private repadmin to repadmin link ---*/
connect repadmin/repadmin@sales.world
create database link hq.world connect to repadmin identified by repadmin

/*--- Schedule replication from Sales to HQ ---*/
execute dbms_defer_sys.schedule_push(
     destination => 'hq.world',
     interval => 'sysdate + 1/24',
     next_date => sysdate,
     stop_on_error => FALSE,
     parallelism => 1)

/*--- Schedule purge of def tran queue at Sales ---*/
execute dbms_defer_sys.schedule_purge(
     next_date => sysdate,
     interval = 'sysdate + 1',
     delay_seconds => 0,
     rollback_segment =>'')

/*--- Create link from propagator to receiver for scheduled push ---*/
connect prop_rec/prop_rec@sales.world
create database link hq.world connect to prop_rec identified by prop_rec
```

Security Setup for Snapshot Replication

Nearly all users should find it easiest to use the Replication Manager setup wizard when configuring snapshot replication security. However, for certain specialized cases, you may need to use the replication management API to perform these setup operations.To configure a replication environment, the database administrator must connect with DBA privileges to grant the necessary privileges to the replication administrator.

First set up user accounts at each snapshot site with the appropriate privileges to configure and maintain the replication environment and to propagate replicated changes. You must also define links for these users to the associated master site. You may need to create additional users, or assign additional privileges to users at the associated master site.

In addition to end users who will be accessing replicated objects, there are three special categories of "users" at a snapshot site:

- Replication administrators who are responsible for configuring and maintaining a replication environment.
- Propagators who are responsible for propagating deferred transactions.
- Refreshers who are responsible for pulling down changes to the snapshots from the associated master tables.

Typically, a single user performs each of these functions. However, there may be situations where you need different users performing these functions. For example, snapshots may be created by a snapshot site administrator and refreshed by another end user.

Table 8–2 describes the privileges needed to create and maintain a snapshot site.

Table 8–2 Required Snapshot Site User Accounts

User	Privileges
snapshot site replication administrator	DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA
propagator	DBMS_DEFER_SYS.REGISTER_PROPAGATOR
refresher	CREATE ANY SNAPSHOT ALTER ANY SNAPSHOT

In addition to creating the appropriate users at the snapshot site, you may need to create additional users at the associated master site, as well. Table 8–3 on on page 8-27 describes the privileges need by master site users to support a new snapshot site.

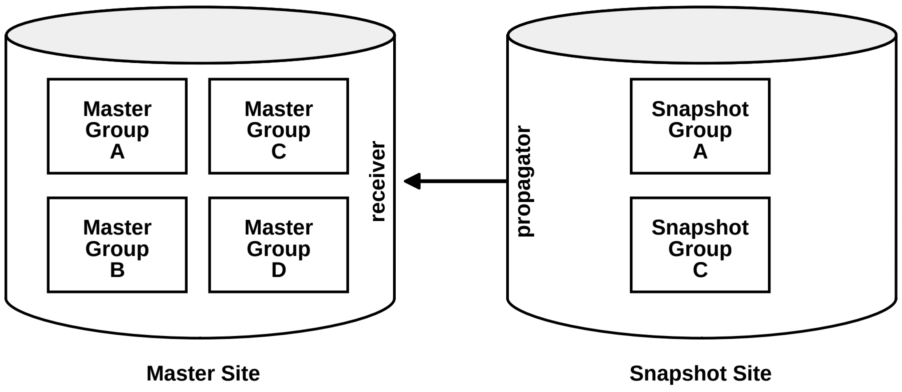
Trusted vs. Untrusted Security

In addition to the different users at the master site, you also need to determine which type of security model you will implement: trusted or untrusted. With a trusted security model, the receiver and proxy snapshot administrator have access to ALL local master groups. Since the proxy snapshot administrator and receiver perform database activities at the local master site on behalf of the snapshot administrator and propagator at the remote snapshot site, the propagator and snapshot administrator also have access to ALL master groups at the master site (remember that a single receiver is used for ALL incoming transactions).

For example, consider the scenario in Figure 8–9. Even though Snapshot Groups A and C exist at the snapshot site (based on Master Groups A and C at the Master Site), the propagator and snapshot administrator have access to Master Groups A, B, C, and D at the Master Site because the trusted security model has been used. While this greatly increases the flexibility of database administration (since the DBA can perform administrative functions at any of these remote sites and have these changes propagated to the master sites), it also increases the chances of a malicious user at a remote site viewing or corrupting data at the master site.

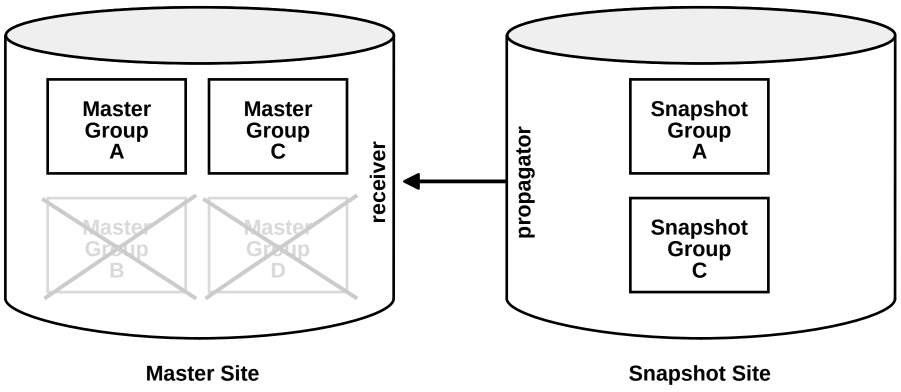
Regardless of the security model used, Oracle8*i* automatically grants the appropriate privileges for objects as they are added to or removed from a replicated environment.

Figure 8–9 Trusted Security: Snapshot Replication



Untrusted security assigns only the privileges to the proxy snapshot administrator and receiver that are required to work with specified master groups; the propagator and snapshot administrator, therefore, will only be able to access these specified master groups at the Master Site. Figure 8–10 illustrates an untrusted security model with snapshot replication. Since the Snapshot Site contains Snapshot Groups A and C, access to only Master Groups A and C are required. Using untrusted security does not allow the propagator or the snapshot administrator at the Snapshot Site to access Master Groups B and D at the Master Site.

Figure 8–10 Trusted Security: Snapshot Replication



Typically, snapshot sites are more vulnerable to security breaches and therefore the untrusted security model is used. There are very few reasons why you would want to use a trusted security model with your snapshot site and it is recommended that you use the untrusted security model with snapshot sites.

One reason you might choose to use a trusted security model is when your snapshot site is considered a master site in every way (security, constant network connectivity, resources) but is a snapshot only because of data partitioning requirements (remember that horizontal or vertical partitioning are not supported in a multimaster configuration).

Use the appropriate API calls listed for the proxy snapshot administrator and receiver in Table 8–3 to assign the different users the appropriate privileges.

Table 8–3 Required Master Site User Accounts

User	Privileges

proxy snapshot site administrator	See the "DBMS_REPCAT_ADMIN.REGISTER_USER_ REPGROUP" procedure in the *Oracle8i Replication API Reference* book for details.
	Trusted:
	DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP
	`privilege => 'proxy_snapadmin'`
	`list_of_gnames => NULL`
	Untrusted:
	DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP
	`privilege => 'proxy_snapadmin'`
	`list_of_gnames => '`*`mastergroupname`*`'`

receiver	See the "DBMS_REPCAT_ADMIN.REGISTER_USER_ REPGROUP" procedure in the *Oracle8i Replication API Reference* book for details.
	Trusted:
	DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP
	`privilege => 'receiver'`
	`list_of_gnames => NULL`
	Untrusted:
	DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP
	`privilege => 'receiver'`
	`list_of_gnames => '`*`mastergroupname`*`'`

proxy refresher	**Trusted**: grant CREATE SESSION grant SELECT ANY TABLE **Untrusted**: grant CREATE SESSION grant SELECT on necessary master tables and snapshot logs

After creating the accounts at both the snapshot and associated master sites, you need to create the following private database links, including username and password, from the snapshot site to the master:

- From the snapshot replication administrator to the proxy snapshot replication administrator.
- From the propagator to the receiver.
- From the refresher to the proxy refresher.

Assuming you have designated a single user account to act as snapshot administrator, propagator, and refresher, you will need to create one link for each snapshot site. You do not need a link from the master site to the snapshot site.

After creating these links, you must call DBMS_DEFER_SYS.SCHEDULE_PUSH and DBMS_DEFER_SYS.SCHEDULE_PURGE at the snapshot site to define how frequently you want to propagate your deferred transaction queue to the associated master site, and how frequently you wish to purge this queue. You must also call DMBS_REFRESH. REFRESH at the snapshot site to schedule how frequently to pull changes from the associated master site.

Avoiding Delete Conflicts

To avoid encountering delete conflicts, you might find it easiest to mark rows as deleted and purge them later. This section outlines a simple technique for purging these marked rows using procedural replication.

Suppose that your database contains the following MAIL_LIST table:

```
Name                Null?      Type
------------------- ---------- --------     --------------
CUSTNO              NOT NULL   NUMBER(4)    PRIMARY KEY
CUSTNAME                       VARCHAR2(10)
ADDR1                          VARCHAR2(30)
ADDR2                          VARCHAR2(30)
CITY                           VARCHAR2(30)
STATE                          VARCHAR2(2)
ZIP                            NUMBER(9)
PHONE                          NUMBER(10)
REMOVE_DATE                   DATE
```

Instead of deleting a customer when he or she requests to be removed from your mailing list, the REMOVE_DATE column would be used to indicate former customers; A NULL value would be used for current customers. After customers request removal from the mailing list, their rows are no longer updated. Such a convention avoids conflicts when the rows are actually deleted sometime later. A view of current customers could be defined as follows:

```
CREATE OR REPLACE VIEW corp.current_mail_list AS
 SELECT custno, custname, addr1, addr2, city, state, zip, phone
  FROM corp.mail_list WHERE remove_date IS NULL;
```

Periodically, perhaps once a year after the holiday sales, the former customers would be purged from the table using the REMOVE_DATE field. Such a delete could be performed using row-level replication just by performing the following delete:

```
DELETE corp.mail_list
  WHERE remove_date IS NOT NULL AND remove_date<'01-JAN-95';
```

However, for a large company with an extensive mail order business, the number of former customers could be quite large resulting in a lot of undesired network traffic and database overhead. Instead, the procedural replication could be used using the following package:

```
CREATE OR REPLACE PACKAGE corp.purge AS
 PROCEDURE remove_cust(purge_date IN DATE);
END;
/
CREATE OR REPLACE PACKAGE BODY corp.purge AS
 PROCEDURE remove_cust(purge_date IN DATE) IS
 BEGIN
  -- turn off row-level replication for set delete
  dbms_reputil.replication_off;
  -- prevent phantom reads
  LOCK TABLE corp.mail_list IN EXCLUSIVE MODE;
  DELETE corp.mail_list 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;
```

The DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT procedure would have been used to generate the DEFER_PURGE package during the initial replication setup. Then, the procedural replication package could be called as follows by a single master site:

```
BEGIN
 defer_purge.remove_cust('14-APR-97','Y');
END;
```

The procedure, PURGE.REMOVE_CUST, would be executed locally and asynchronously executed at each master, resulting in many rows being deleted with only minimal network traffic.

To ensure that there are no outstanding transactions against the rows to be purged, your application should be written to *never* update logically deleted rows and the REMOVE_DATE should be old enough to ensure that the logical delete of the row is propagated before the row is purged. Thus, in the previous example, it is probably not necessary to lock the table in EXCLUSIVE mode; although this is another method of guaranteeing that these rows not be updated during the purge.

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 in Chapter 1. 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, he or she 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 will update 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 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.

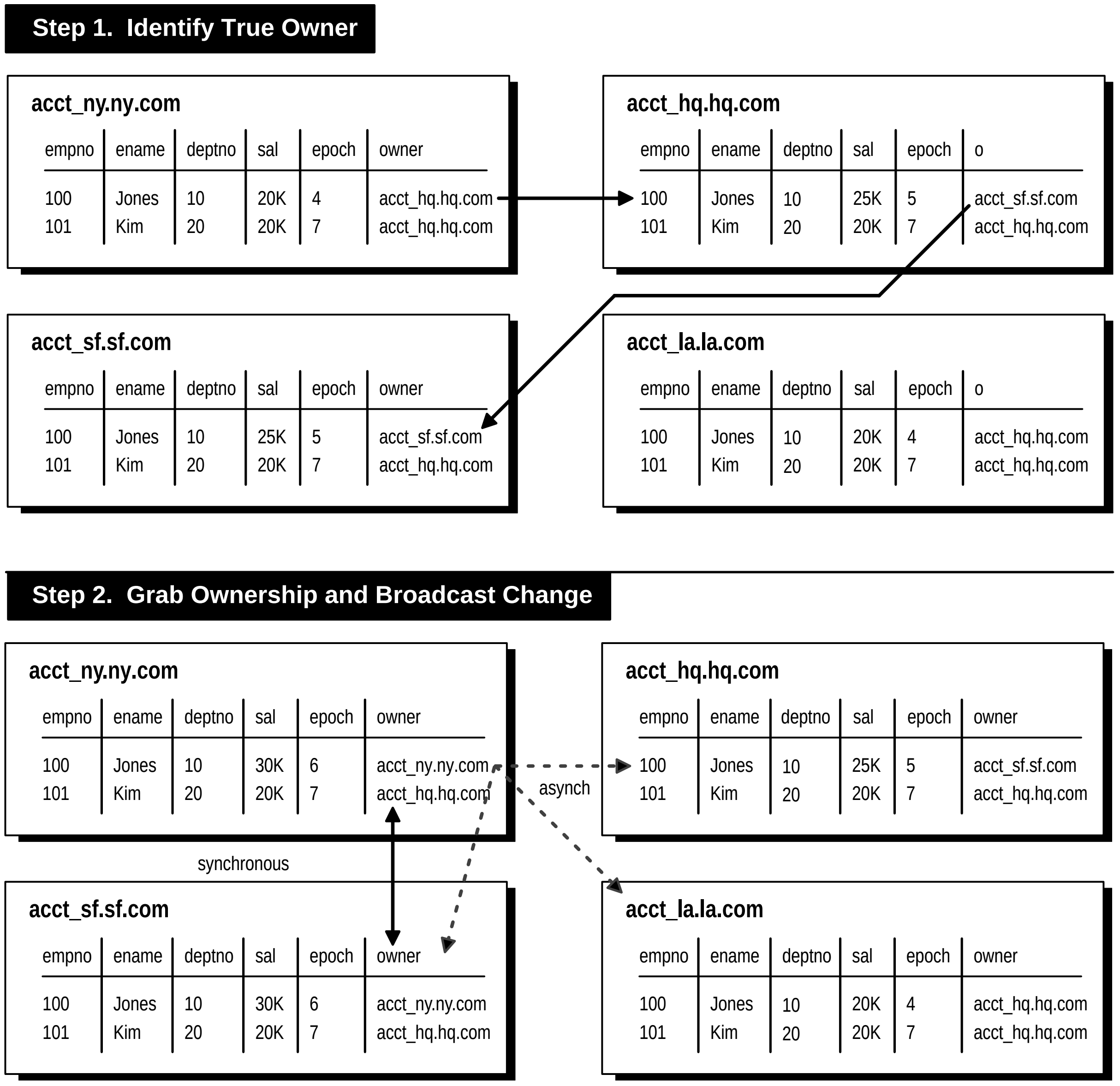
You should design your application to implement token passing for you automatically. You should not allow the owner or epoch columns to be updated outside this application.

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. Grab ownership of the row.
4. Perform the update. (Oracle releases the lock when you commit your transaction.)

For example, Figure 8–11 illustrates how ownership of employee 100 passes from the ACCT_SF database to the ACCT_NY database.

Figure 8–11 Grabbing the Token


Step 1. Identify True Owner
Step 2. Grab Ownership and Broadcast Change
synchronous
asynch

Step 1. Identify True Owner

acct_ny.ny.com

empno	ename	deptno	sal	epoch	owner
100	Jones	10	20K	4	acct_hq.hq.com
101	Kim	20	20K	7	acct_hq.hq.com

acct_hq.hq.com

empno	ename	deptno	sal	epoch	o
100	Jones	10	25K	5	acct_sf.sf.com
101	Kim	20	20K	7	acct_hq.hq.com

acct_sf.sf.com

empno	ename	deptno	sal	epoch	owner
100	Jones	10	25K	5	acct_sf.sf.com
101	Kim	20	20K	7	acct_hq.hq.com

acct_la.la.com

empno	ename	deptno	sal	epoch	o
100	Jones	10	20K	4	acct_hq.hq.com
101	Kim	20	20K	7	acct_hq.hq.com

Step 2. Grab Ownership and Broadcast Change

acct_ny.ny.com

empno	ename	deptno	sal	epoch	owner
100	Jones	10	30K	6	acct_ny.ny.com
101	Kim	20	20K	7	acct_hq.hq.com

acct_hq.hq.com

empno	ename	deptno	sal	epoch	owner
100	Jones	10	25K	5	acct_sf.sf.com
101	Kim	20	20K	7	acct_hq.hq.com

acct_sf.sf.com

empno	ename	deptno	sal	epoch	owner
100	Jones	10	30K	6	acct_ny.ny.com
101	Kim	20	20K	7	acct_hq.hq.com

acct_la.la.com

empno	ename	deptno	sal	epoch	owner
100	Jones	10	20K	4	acct_hq.hq.com
101	Kim	20	20K	7	acct_hq.hq.com

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 pseudo code for this algorithm is shown below:

```
-- Pseudo 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 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 replicated 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 will be propagated to all other sites in the usual manner. When the SF change is propagated to NY, it will be 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 will ignore them because they will 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, will be asynchronously propagated to the other sites in the usual manner.

Modifying Tables without Replicating the Modifications

You may encounter a situation where you need to modify a replicated object, but you do not want this modification replicated to the other sites in the replicated environment. For example, you might want to disable replication in the following situations:

- When you are using procedural replication to propagate a change, you should 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 "Triggers and Replication".

- Sometimes when you manually resolve a conflict, you might not want to replicate this modification to the other copies of the table.

  You might 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 might 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 Replication Manager 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 the Advanced Replication Facility

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.

Attention: 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, you should call REPLICATION_OFF at the start of your procedure, as shown in the following example. This ensures that the advanced 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;
```

Re-enabling the Advanced 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 snapshots. This procedure takes no arguments. Calling REPLICATION_ON sets the internal replication variable to TRUE.

Triggers and Replication

If you have defined a replicated trigger on a replicated table, you may need to ensure that the trigger fires only once for each change that you make. Typically, you will only want the trigger to fire when the change is first made, and you will 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 will fire at each remote site. Any updates performed by the replicated trigger will not be 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.

Enabling/Disabling Replication for Snapshots

To disable all local replication triggers for snapshots at your current site, set the internal refresh variable to TRUE by calling SET_I_AM_A_REFRESH, as shown in the following example:

```
DBMS_SNAPSHOT.SET_I_AM_A_REFRESH(value => TRUE);
```

To re-enable the triggers, set the internal refresh variable to FALSE, as shown below:

```
DBMS_SNAPSHOT.SET_I_AM_A_REFRESH(value => FALSE);
```

To determine the value of the internal refresh variable, call the I_AM_A_REFRESH function as shown below:

```
ref_stat := DBMS_SNAPSHOT.I_AM_A_REFRESH;replication:advanced
techniques<$startrange>;advanced replication:techniques<$startrange>
```

9

Using Deferred Transactions

This chapter describes the following topics:

- Listing Information about Deferred Transactions.
- Creating a Deferred Transaction.
- LOB Storage.

Listing Information about Deferred Transactions

Oracle provides several tables and views for you to use in administering deferred transactions. These views provide information about each deferred transaction, such as the transaction destinations, the deferred calls that make up the transaction, and any errors encountered during attempted execution of the transaction.

Attention: You should not modify these tables directly; use the procedures provided in the DBMS_DEFER and DBMS_DEFER_SYS packages.

These views are briefly described below. For more information, see "Data Dictionary Views" in the *Oracle8i Replication API Reference* book.

Data Dictionary View	Description
DEFCALL	Records all deferred remote procedure calls (RPCs).
DEFCALLDEST	Lists the destinations for each deferred remote procedure call.
DEFDEFAULT DEST	Lists the default destination for deferred remote procedure calls.
DEFERROR	Provides information about transactions that could not be applied.
DEFLOB	Storage for LOB parameters to deferred RPCs.
DEFSCHEDULE	Displays information about when a job is next scheduled to be executed.
DEFTRAN	Records all deferred transactions.
DEFTRANDEST	Lists the destinations for a deferred transaction.

Creating a Deferred Transaction

Every well formed deferred transaction must consist of zero or one DBMS_ DEFER.TRANSACTION calls followed by zero or more well formed deferred remote procedure calls, followed by a SQL COMMIT statement.

Attention: The procedures for which you are building deferred calls must be part of a package. Deferred calls to standalone procedures are not supported.

Every well formed deferred remote procedure call must consist of one DBMS_DEFER.CALL call, followed by zero or more DBMS_DEFER.*datatype*_ARG calls. The number of calls to the appropriate *datatype*_ARG procedures is determined by the value of the ARG_COUNT parameter passed to the CALL procedure.

If you do not call DBMS_DEFER.TRANSACTION to indicate the start of a transaction, Oracle considers your first call to DBMS_DEFER.CALL to be the start of a new transaction.

Security

To create your own deferred transactions, you must have the EXECUTE privilege on the DBMS_DEFER package. This package is owned by SYS. Because deferred transactions are executed in the *privilege domain of the replication propagator,* EXECUTE privileges on the DBMS_DEFER package should not be widely granted.

Suggestion: To control access to these procedures, you should create a cover package in the replication propagator's schema, and grant EXECUTE on this cover package.

Specifying a Destination

In addition to building the calls that make up a deferred transaction, you must also specify the destination for this transaction. Transactions placed into the deferred transaction queue by the advanced replication facility are queued to all of the asynchronous locations (dblinks) for the replicated object, as listed in the DBA_REPPROP view. When you use the procedures in the DBMS_DEFER package to add a deferred transaction to the queue, you must specify a destination using one of the following methods. These methods are listed in order of precedence:

1. If you meet the following conditions, the DBA_REPPROP view is used to determine destinations for the deferred transaction:
   - You do not use the NODES parameter to specify a destination in the call to DBMS_DEFER.TRANSACTION.
   - You do not use the NODES parameter to specify a destination for any calls to DBMS_DEFER.CALL.
   - Every call corresponds to a procedure in a package generated for an object in DBA_REPOBJECT.

Note: This method cannot be combined with any of the following methods.

2. You specify one or more fully qualified database names as the NODES parameter to the DBMS_DEFER.CALL procedure. This value applies to the current deferred remote procedure call only.
3. You specify one or more fully qualified database names as the NODES parameter to the DBMS_DEFER.TRANSACTION procedure. This value applies to all deferred calls that make up the transaction.
4. If you do not use one of the previous mechanisms to specify a destination, Oracle uses the contents of the "DEFDEFAULTDEST View" to determine the destination for the calls.

Initiating a Deferred Transaction

Indicate the start of a new deferred transaction by calling the TRANSACTION procedure in the DBMS_DEFER package, as shown in the following example:

```
nodes dbms_defer.node_list_t;
node(1) := ’acct_hq.hq.com’;
node(2) := ’acct_ny.ny.com’;
DBMS_DEFER.TRANSACTION(nodes);
```

In this example, any calls that make up the deferred transaction for which you do not specify a destination when you call DBMS_DEFER.CALL, will be queued for the ACCT_HQ and ACCT_NY databases.

The call to TRANSACTION is optional. If you do not call TRANSACTION, Oracle considers your first call to DBMS_DEFER.CALL to be the start of a new transaction. Calling TRANSACTION is useful if you want to specify a list of nodes to which to forward the deferred calls, and the list is the same for all calls in the deferred transaction.

All deferred transactions are recorded in the DEFTRAN view. Each destination of the transaction is noted in the DEFTRANDEST view.

Additional Information: See the "TRANSACTION" procedure in the *Oracle8i Replication API Reference* book for details.

Deferring a Remote Procedure Call

To build a deferred call to a remote procedure, call the CALL procedure in the DBMS_DEFER package, as shown in the following example:

```
DBMS_DEFER.CALL(
     schema_name      =>   'accts_rec',
     package_name     =>   'hr',
     proc_name        =>   'hire_emp',
     arg_count        =>   3);
```

This example builds a deferred call to the HR.HIRE_EMP procedure in the ACCTS_REC schema. This HIRE_EMP procedure takes three arguments. No destination is specified for the deferred call, so the destination must have been specified using one of the other methods outlined on page on page 9-3.

All deferred remote procedure calls are recorded in the DEFCALL view. Each destination for the call is noted in the DEFCALLDEST view.

Additional Information: See the "CALL" procedure in the *Oracle8i Replication API Reference* book for details.

Queuing a Parameter Value for a Deferred Call

After deferring a call to a remote procedure, you must provide the data that is passed to this procedure (only IN parameters are supported). There must be one call for each of the arguments that is passed to the remote procedure, and these calls must be made in the order that the arguments must be passed. The type of the data determines which procedure in the DBMS_DEFER package you must call. For example, suppose you deferred a call to the HIRE_EMP procedure, and it took three arguments, as shown below:

```
HIRE_EMP(ename IN VARCHAR2, empno IN NUMBER, salary IN NUMBER)
```

After building the deferred call to HIRE_EMP, you could pass the necessary data to this procedure by making the following three calls:

```
DBMS_DEFER.VARCHAR2_ARG('scott');
DBMS_DEFER.NUMBER_ARG(12345);
DBMS_DEFER.NUMBER_ARG(30000);
```

Depending upon the type of the data that you need to pass to the procedure, you need to call one of the following procedures in the DBMS_DEFER package for each argument to the procedure:

```
DBMS_DEFER.NUMBER_ARG(arg IN NUMBER);
DBMS_DEFER.DATE_ARG(arg IN DATE);
DBMS_DEFER.VARCHAR2_ARG(arg IN VARCHAR2);
DBMS_DEFER.NVARCHAR2_ARG(arg IN NVARCHAR2);
DBMS_DEFER.CHAR_ARG(arg IN CHAR);
DBMS_DEFER.NCHAR_ARG(arg IN NCHAR);
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);
```

Note: The RAW_ARG, CHAR_ARG, NCHAR_ARG, VARCHAR2_ARG, and NVARCHAR2_ARG procedures can raise an ORA-23323 exception if the argument that you pass to the procedure is too long.

Adding a Destination to the DEFDEFAULTDEST View

If you use the DBMS_DEFER package to build a deferred transaction, and you do not supply a destination for a deferred transaction or the calls within that transaction, Oracle uses the DEFDEFAULTDEST view to determine the destination databases to which you want to defer a remote procedure call.

To add a destination database to this view call the ADD_DEFAULT_DEST procedure in the DBMS_DEFER_SYS package as shown in the following example:

```
DBMS_DEFER_SYS.ADD_DEFAULT_DEST( dblink => 'acct_ny.ny.com');
```

In this example, any *future* deferred transactions for which no destination has been specified will be queued for the ACCT_NY database.

Additional Information: See the "ADD_DEFAULT_DEST" procedure in the *Oracle8i Replication API Reference* book for details.

Removing a Destination from the DEFDEFAULTDEST View

To remove a destination database from the DEFDEFAULTDEST view, call the DELETE_DEFAULT_DEST procedure in the DBMS_DEFER_SYS package, as shown in the following example:

```
DBMS_DEFER_SYS.DELETE_DEFAULT_DEST( dblink => 'acct_ny.ny.com');
```

In this example, any future deferred transactions that you create will no longer be queued for the ACCT_NY database as the default.

To delete a transaction from the deferred transaction queue, you can use Replication Manager. For more information, see "Purging a Site's Deferred Transaction Queue".

Additional Information: See the "DELETE_DEFAULT_DEST" procedure in the *Oracle8i Replication API Reference* book for details.

Executing a Deferred Transaction

When you build a deferred transaction, the transaction is added to the deferred transaction queue at your local site. The remote procedures are not executed until this queue is pushed. You can either schedule this queue to be pushed at a periodic interval by creating a scheduled link or by calling DBMS_DEFER_SYS.SCHEDULE_PUSH, or you can force the queue to be pushed immediately with Replication Manager or by calling DBMS_DEFER_SYS.PUSH. These transactions are propagated in the same manner as your DML changes are propagated by the advanced replication facility.

LOB Storage

Oracle supports large internal objects (LOBs): binary LOBs (BLOBs); character LOBs (CLOBs); and national character LOBs (NCLOBs).

Note: For data manipulation language (DML) or for piecewise update, the larger the size of the LOB (update), the more the propagation time will increase.

Note: All sites must be Oracle8 or greater sites to support deferred RPCs of above named objects.

For security, note that a LOB parameter to a (deferred) RPC is visible in the transaction only while the RPC is being executed.

DEFLOB View of Storage for RPC

Oracle stores internal LOB parameters to deferred RPCs in a side table that is referenced only by way of a synonym. This gives the you flexibility for storage parameters and the containing schema. The following shows the default storage table for LOB parameters.

```
CREATE TABLE system.def$_lob(
 id RAW(16) CONSTRAINT def$_lob_primary PRIMARY KEY,
 deferred_tran_db VARCHAR2(128), -- origin db
 deferred_tran_id VARCHAR2(22), -- transaction id
 blob_col BLOB,
 clob_col CLOB
 nclob_col NCLOB)
/
-- make deletes fast
CREATE INDEX system.def$_lob_n1 ON system.def$_lob(
 deferred_tran_db,
 deferred_tran_id)
/
-- use a synonym in case underlying table is moved
CREATE SYNONYM sys.def$_lob FOR system.def$_lob
/
CREATE OR REPLACE VIEW DefLOB AS SELECT * FROM sys.def$_lob
/
CREATE PUBLIC SYNONYM DefLOB FOR DefLOB
/
```

A
New Features

This Appendix briefly describes the new replication features of the Oracle8*i* Server and provides pointers to other chapters in this document where you can get additional information. This chapter also retains Oracle8 Server new features to help those users migrating from Oracle7 to Oracle8*i*.

New features include:

Oracle8 Server

- Performance Enhancements
- Data Subsetting Based on Subqueries
- Enhanced, System-Based Security Model
- New Replication Manager Features

Oracle8*i* Server

- Performance Improvements
- Improved Mass Deployment Support
- Improved Security
- Replication Manager

Oracle8 — New Features

The Oracle8 features and enhancements described below comprise the overall effort to optimize multimaster replication performance and make certain types of snapshot subsetting fast refreshable. LOB support was added for Oracle8. The manageability and security was also enhanced for Oracle8. All are included in advanced replication.

Performance Enhancements

Oracle8 provides significant performance improvements based on the following new features.

Parallel Propagation of Deferred Transactions

Dramatically improves throughput performance by parallelizing the propagation of a replication transaction stream while maintaining consistency and transaction dependencies.

Additional Information: See "Deciding between Serial and Parallel Propagation" on page 2-14.

Internalized Replication Triggers

Internal triggers:

- Improve response time performance.
- Reduce processing overhead.
- Require less administration.

Additional Information: See "Oracle’s Multimaster Replication Architecture" on page 2-2.

Reduced Data Propagation

Oracle8 reduces the amount of replicated data propagated over the network. Propagation can be reduced to only the following: new values of columns updated; old values of columns needed for conflict detection and resolution; and primary key values. This feature is important for performance when replicating LOBs.

Additional Information: See "Minimizing Data Propagation for Update Conflict Resolution" on page 6-42.

Data Subsetting Based on Subqueries

Snapshots defined with certain types of subqueries can now be fast refreshed. This enables subsets of data to be easily defined and maintained. This feature is important for mass deployment applications, such as salesforce automation and branch automation.

Additional Information: See "Data Subsetting with Snapshots" on page 3-7.

Large Object Datatypes (LOBs) Support

Oracle8 supports the replication of the following types of large objects:

- Binary LOBs (BLOBs).
- Character LOBs (CLOBs).
- National language support character LOBs (NCLOBs).

Additional Information: See the following sections: "Datatype Considerations for Replicated Tables" on page 2-26 and "Datatype Considerations for Snapshots" on page 3-17.

Improved Management and Ease of Use

Oracle8 facilitates database management with the following features.

Fine grained Quiesce

Replication master groups can now be individually quiesced without impacting other replication groups. Master groups can continue to replication updates while other master groups are quiesced.

Additional Information: See "Suspending Replication Activity for a Master Group" on page 2-22.

Primary Key Snapshots

Primary key snapshots allow you to reorganize master tables while preserving fast refresh capability. Oracle8 adds primary key snapshots as the default, and continues to support ROWID snapshots.

Additional Information: See Appendix B, "Migration and Compatibility".

Snapshot Registration at Master Sites

Oracle8 automatically registers information about a snapshot s at master sites. This facilitates monitoring and distributed administration.

Additional Information: See "Registering a Snapshot at its Master Site" on page 7-36.

Reorganizing Tables With Capability of Fast Refresh

Oracle8 provides utilities to enable you to "reorg" master tables while preserving the consistency of master snapshot logs.

Additional Information: See "Reorganizing Master Tables that Have Snapshot Logs" on page 7-25.

Support for Offline Instantiation

Offline instantiation of schemas and database using Export/Import is now more automatic.

Additional Information: See "Snapshot Cloning and Offline Instantiation" on page 8-14.

Deferred Constraints for Updateable Snapshots

Updateable snapshots now support declarative referential and uniqueness constraints.

Additional Information: See "Replicating Object Definitions to Master Sites" on page 2-26.

Partitioned Tables and Indexes

Oracle8 supports the replication of partitioned tables and indexes. You can use this feature if you want the replicated table to have the same partitions as the table at the master definition site.

Enhanced, System-Based Security Model

Oracle8 system-based security model:

- Improves consistency: a single system-level model operates the same in both synchronous and asynchronous environments.
- Improves reliability: transactions are less likely to fail due to the lack of privileges at the receiving sites.

- Simplifies links: a single user can act as repadmin and repsys.
- Eliminates the need for a “user-level” model.
- One or more snapshot owners can perform snapshot refresh.

Additional Information: See "Preparing for Multimaster Replication" on page 2-7, "Prepare for Snapshots" on page 3-24, and Appendix B, "Migration and Compatibility".

New Replication Manager Features

Replication Manager now includes several wizards to help you configure your system quickly.

- Setup of multimaster configurations, including account, schema, and link creation.
- Setup of snapshot site configurations, including account, schema, and link creation.
- Setup of snapshot groups.
- Setup of snapshots.

Oracle Replication Manager includes support for most new features of Oracle8 Server replication functionality, including:

- Automatic recognition of each site's database version and dynamic configuration to manage both 8.x and 7.3 databases.
- Support for managing snapshot logs.
- Easy entry of date and interval expressions for jobs, refresh groups, and scheduled links.
- For Oracle8 databases, a flag to indicate that a table requires regeneration of replication support.
- Support for registering a site's replication administrator and propagator.
- A database objects folder to display database objects for master groups.
- Drag and drop functionality for master groups and snapshot refresh groups.
- Support for the validation of a master group at all master sites.

Oracle8*i* — New Features

The Oracle8*i* features and enhancements described below comprise the overall effort to optimize replication performance and make snapshot environment distribution and security more effective. All are included in advanced replication.

Performance Improvements

Significant performance gains are realized by the internalization of PL/SQL replication packages and by optimizations to snapshot refresh.

Internal Apply Packages

Continuing the trend started with Oracle8, more replication code has been moved into the database engine in Oracle8*i*. The PL/SQL generated packages used to apply replicated transactions at a remote site have been internalized. This allows replicated transactions to be more efficiently applied at remote sites and because packages are not generated, a site can be more quickly instantiated. Internal packages are also more secure because they are tamper proof.

Faster Snapshot Refresh

Snapshot refresh has been optimized to support large refresh groups. There is improved support for subquery snapshots, and for null refresh (no changes to the master tables since the last refresh). A single refresh group can now contain 400 snapshots, and the number of roundtrips required to refresh snapshots in a refresh group has been reduced.

Improved Mass Deployment Support

With Oracle8*i*, Oracle is shifting its advanced replication focus from back office types of applications requiring near real-time replication of data to the growing market of front office applications, in particular mass deployment.

Parameterized Snapshot Deployment Templates

These facilitate the mass deployment of information to support such applications as field service and sales force automation. These templates represent a grouping together of snapshots and other database objects to be instantiated at a node. They allow a DBA to centrally package a snapshot environment for easy, custom, and secure distribution to one or multiple sites. The goal is to create the environment once, then deploy the snapshot deployment template as often as necessary. Template parameters allow data subsetting at a remote site without redefining the template, and a template may be defined as public or private. Public templates may be instantiated at any site, whereas private ones can only be instantiated at pre-defined, authorized sites. An Oracle Replication Manager deployment wizard guides the DBA through the selection of schema objects to add to the template, the selection of parameters, and defining authorizations.

Column Level Snapshot Subsetting

Updateable snapshots can now be subsetted horizontally (selected rows) or vertically (selected columns). The previous release allows horizontal subsetting only. Vertical partitioning allows the deployment of the minimum amount of data needed by a remote site, thus reducing connection time. It also protects snapshot sites from changes to their associated masters. A column can be added to a master site without impacting the snapshot site, or a column can be deleted and not impact the snapshot site, if the snapshot site does not currently reference that column.

Offline Instantiation

Snapshot deployment templates can be instantiated online or offline. Online instantiation allows a snapshot site to instantiate the template while connected to the target master site. The advantage here is that the data will be current. However, this is at the cost of requiring a live connection, possibly of long duration and having the potential of generating extreme network traffic that could degrade other network services. Offline instantiation allows the DBA to package the deployment group templates and required data onto some type of storage media (tape, CD-ROM, etc.) for distribution to a snapshot site. Instead of connecting to the master site, instantiation can be done by pulling the template and data from the storage media. Users can fast refresh immediately after completing an offline instantiation, a full refresh is not required. Offline instantiation is an ideal solution for mass deployment situations where many disconnected laptops will be instantiating the target template.

Improved Security

While the scripts used to instantiate a snapshot site are generated at the master site and can control access to data, it is still necessary to connect to a receiver and proxy snapshot administrator to propagate replicated transactions and to refresh snap-shots. Oracle8*i* enhancements to the replication security model eliminate certain security deficiencies regarding the granting of privileges to untrusted sites.

- ❏ Object privileges, as required by a receiver of remote procedure calls (RPCs) at a master site, are now automatically managed. Only required object privileges are granted to untrusted sites.
- ❏ Proxy snapshot administrators now have a way of accessing objects in object groups without being granted excessive privileges.

Replication Manager

A number of improvements have been made to the Oracle Replication Manager, some have already been mentioned. A major and noticeable enhancement is that it has been rewritten to conform to the new Oracle Enterprise Manager (OEM) Java interface. Oracle Replication Manager can now be run from anywhere in the network, and it is not constrained to a Windows based machine. There is also a replication class of events in OEM which, for example, can be used to monitor errors or delinquent snapshot refreshes.

Improved Oracle Lite Integration

The integration between the Oracle8*i* Server / Oracle8*i* Server Enterprise Edition and Oracle8*i* Lite has been improved to provide better performance and increased functionality. All of the replication changes previously described are supported and Oracle Lite users with laptops at remote sites will especially benefit from reduced connection time.

B
Migration and Compatibility

This Appendix explains the steps that need to be taken to migrate a replication environment from Oracle7 to Oracle8*i*. Topics covered include:

Note: This appendix addresses the need to migrate from Oracle7 to Oracle8*i*. For more information about migrating from Oracle8 to Oracle8*i*, please refer to the *Oracle8i Migration* manual.

Migration Overview

In some cases you may find it easiest to migrate your environment, particularly the multimaster component of your environment, in one step. Typically, this will only be possible for small configurations. Instead, you may wish to migrate an existing Oracle7 replication environment to Oracle8*i* incrementally. Replication and administrative operations can be run successfully in a mixed Oracle7 and Oracle8*i* replication environment.

To successfully interoperate, however, you must observe the following restrictions:

- Oracle8*i* snapshot sites can only interact with Oracle7 Release 7.3.3 or greater master sites.
- Oracle8*i* master sites can only interact with Oracle7 Release 7.3.4 or greater snapshot sites and with Oracle7 Release 7.3.3 or greater master sites.

After migrating a master site to Oracle8*i*, perform a full refresh of all of associated snapshot sites.

Downgrading from Oracle8*i* to Oracle7 is not supported.

Certain Oracle8*i* replication features require that all sites be successfully migrated to at least Oracle8 before the features can be used. For example, before you can use primary key snapshots, both the snapshot site and its associated master site must be migrated to at least Oracle8. The Oracle8*i* simple snapshots with subqueries feature and the master table reorganization procedures require that you first upgrade from Rowid snapshots to primary key snapshots.

Migration using a full database export from Oracle7 and import to Oracle8*i* is also supported.

Migrating All Sites at Once

This section describes how to migrate your entire multimaster environment at once to Oracle8*i*. Note that any snapshot sites that you do not also migrate to Oracle8*i*, must be upgraded to Oracle7 Release 7.3.4 or greater.

Follow these steps to migrate all master sites and (optionally) snapshot sites at one time:

1. Quiesce the replication environment by executing DBMS_REPCAT. SUSPEND_MASTER_ACTIVITY at the master definition site for all master replication groups, and stopping all propagation and refreshing from snapshot sites to the master, for example, by temporarily suspending or “breaking” entries in the job queue that control automated propagation and refreshing at the snapshot sites.

You must also resolve and re-execute any errors in the local error queue until it is empty. For more information see the following sections in *Oracle 7 Server Distributed Systems*, *Volume II: Replicated Data:* Chapter 4, “Asynchronous Propagation of DML Changes”, and “Suspending Replication Activity”, as well as Chapter 7, “Resolving an Error Manually”.

2. Migrate all master sites using the Oracle7 to Oracle8*i* Migration Utility and by executing the appropriate migration scripts as documented in *Oracle8i Migration* (e.g. `catrep.sql` and *`rold_release.sql`*, where `old_release` is the previously installed release).

3. Using the Replication Manager setup wizard, create a primary master replication administrator account granting this user Oracle8*i* Replication Administrator, Propagator, and Receiver privileges on all master sites, and set up the appropriate links connecting all sites. See "Preparing for Multimaster Replication" on page 2-7.

4. Using Replication Manager or the replication management API, regenerate replication support for each replication base object. See "Generating Replication Support for Master Group Objects" on page 2-35 for more information. Among other activities, generating replication support will establish the registered propagator as the owner of generated objects

5. Using Replication Manager or the replication management API, resume replication activity by unquiescing the environment. See "Resuming Replication Activity for a Master Group" on page 2-23 for more information.

6. At a minimum, you must now upgrade all associated snapshot sites to Oracle7 Release 7.3.4. For instructions on migrating your snapshot sites to Oracle8*i*, see "Incremental Migration of Snapshot Sites" on page B-6.

7. All snapshots at all snapshot sites will need a full refresh after their master sites have been migrated to Oracle8*i*. Before the refresh, be certain that you have “unbroken” any jobs that you may have “broken” during migration of your snapshot sites by calling the DBMS_JOB.BROKEN procedure.

   If your snapshots have been defined with the refresh “FORCE” option, their next attempted refresh will full refresh automatically. Snapshots defined with the refresh “FAST” option will need to be manually refreshed using dbms_ refresh.refresh or other refresh procedures.

   If you are using procedural replication at snapshot sites, also regenerate snapshot support on all packages and package bodies used for procedural replication.

Note: If you are migrating all of the master's snapshot sites to Oracle8*i* when the master site is migrated to Oracle8*i*, in other words, you do not need to migrate the snapshot sites incrementally, you can alternatively drop the snapshot logs for the master and recreate them as primary key snapshot logs. The snapshots at each snapshot site should be altered to convert them to primary key snapshots. You can then do a full refresh for each primary key snapshot. See "Upgrading to Primary Key Snapshots" on page B-10 for additional details.

---

8. Drop any administrative accounts and links that you were using to maintain your Oracle7 multimaster replication environment that are not needed in your Oracle8*i* environment. Unnecessary privileges may also be revoked. Be careful not to drop accounts that are needed to maintain any Oracle7 snapshot sites.

Incremental Migration

It is possible to incrementally migrate your replication environment. However, you must carefully analyze the interdependencies between sites to ensure that they will continue to interoperate throughout your migration. Table B–1 describes the conditions that must be met to allow Oracle7 and Oracle8*i* replication sites to interoperate.

Table B–1 Interoperability in a Replication Environment

Environment	Action	Pre-Requisite
Multimaster	Migrate master site from Oracle7 to Oracle8*i*.	All other master sites must be Oracle7 Release 7.3.3 or greater.
Master with dependent snapshots	Migrate master site from Oracle7 to Oracle8*i*.	All dependent snapshot sites must be Oracle7 Release 7.3.4 or greater.
Master with dependent snapshots	Migrate snapshot site from Oracle7 to Oracle8*i*.	Associated master site must be Oracle7 Release 7.3.3 or greater.

To avoid interoperability problems within a replication environment, it is strongly recommended that if you must perform an incremental migration that you perform it in the following order:

1. Upgrade all of your master sites to Oracle7 Release 7.3.3 or greater and follow the steps in "Preparing Oracle7 Master Sites for Incremental Migration" on page B-5 to prepare your Oracle7 master sites for incremental migration.
2. Incrementally migrate all snapshot sites to Oracle8*i*.
3. Incrementally migrate all master sites to Oracle8*i*.

Preparing Oracle7 Master Sites for Incremental Migration

Before beginning incremental migration of Oracle7 master or snapshot sites, your Oracle7 Release 7.3.3 or greater master sites must be configured so that all replication administration and propagation is done within the security context of a single user at each site. Additionally, this primary master replication administrator must have the same username and password at all Oracle7 and Oracle8*i* sites. Your Oracle7 master sites may already be configured in this manner. If not, you must complete the following steps:

1. Choose a primary master replication administrator for your replication environment. You may select your current replication administrator or create a new user.
2. At each master site, grant the required privileges to the primary master replication administrator using both DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_REPGROUP and DBMS_REPCAT_AUTH.GRANT_SURROGATE_REPCAT.
3. If they do not already exist, you must create the following links for from each master site to all other master sites in the multimaster environment (for a total of 3N(N - 1) links):
   - A public database link, created as SYS, that includes a valid global database name, as well as a USING clause with a valid SQL*Net 2.3 TNS alias.
   - A private database link, created as SYS, that includes a valid global database name, as well as a CONNECT TO clause with the username and password of the primary master replication administrator.
   - A private database link, create as the primary replication administrator, that includes a valid global database name, as well as a CONNECT TO clause with the username and password of the primary master replication administrator.

Incremental Migration of Snapshot Sites

Before you can migrate a snapshot site to Oracle8*i*, its associated master site must have been upgraded to Oracle7 Release 7.3.3 or greater and the master site must have been fully prepared for incremental migration.

To incrementally migrate your Oracle7 snapshot sites to Oracle8*i*, complete the following steps:

1. Isolate the snapshot site from the replication environment by stopping all local updates to updateable snapshots at the snapshot site (in a separate session you may lock each snapshot's base table to prevent further transactions). Empty the local deferred transaction queue by pushing the queue to the snapshot's master. Stop all propagation from the snapshot site to its master, for example, by temporarily suspending or "breaking" entries in the job queue that control automated propagation and refreshing at the snapshot sites.
2. Run the Oracle7 to Oracle8*i* Migration Utility and execute the `catrep.sql` and `r`*`old_release`*`.sql` (where `old_release` is the previously installed database version) scripts as documented in *Oracle8i Migration*.
3. Use the Replication Manager setup wizard or execute the appropriate replication management API calls to configure the primary snapshot replication administrator as the replication administrator and propagator for the snapshot site, to configure a receiver account at the associated master, and to create the appropriate links to the master. For Oracle7 master sites your receiver at the master site must be the primary master replication administrator that you prepared in the previous section. If you are using the Replication Manager setup wizard select the customize option to specify this receiver.
4. Using Replication Manager or the appropriate replication management API calls, regenerate snapshot replication support. See "Regenerating Replication Support for an Updateable Snapshot" on page 5-22 for more information. Among other activities, generating replication support establishes the registered propagator as the owner of generated objects
5. Using Replication Manager or the appropriate replication management API calls, reschedule propagation and/or refresh intervals with the master and enable local updates where appropriate. If you used the DBMS_JOB.BROKEN procedure to help isolate your master site in Step 1, you need to "unbreak" your jobs to resume your replication activity from your snapshot sites.
6. Drop any administrative accounts and links that you were using to maintain your Oracle7 replication environment that are not needed in your Oracle8*i* environment. Unnecessary privileges may also be revoked.

Incremental Migration of Master Sites

Before upgrading a master site from Oracle7 to Oracle8*i*, you must meet the following conditions:

- All other master sites in a multimaster environment must be running Oracle7 Release 7.3.3 or greater.
- You must have completed the instructions in "Preparing Oracle7 Master Sites for Incremental Migration" on page B-5.
- Any dependent snapshot sites must be running Oracle7 Release 7.3.4 or greater.

To incrementally migrate your Oracle7 master sites to Oracle8*i*, complete the following steps:

1. Pick a master site to migrate. You should migrate your master definition site first.
2. If you are using procedural replication, record the configuration information and locations (schemas) of existing procedure wrappers. This information will be used later.
3. Isolate the master site from the replication environment. To do this you must:
   - Stop updates to the master site by either:

     calling DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY at the master definition site for all master replication groups,

     or by calling DBMS_DEFER_SYS.UNSCHEDULE_EXECUTION (for Oracle7 sites) or DBMS_DEFER_SYS.UNSCHEDULE_PUSH (for Oracle8*i* sites) at every remote master site and dependent snapshot site, and by preventing update activity at the master site being migrated. You should also refrain from executing any administrative operations at the master definition site that may affect the master site being migrated. Empty the local deferred transaction queue by manually pushing the queue to all sites.
   - Resolve and re-execute any errors in the local error queue until it is empty.
   - Stop any refreshes of the dependent snapshot sites from occurring by “breaking” entries in the job queue at each snapshot site that control automated propagation and refreshing at the snapshot sites.
   - For more information on completing the tasks in Step 1 refer to the following sections in *Oracle7 Server Distributed Systems, Volume II: Replicated Data:* Chapter 4, “Asynchronous Propagation of DML Changes”, “Suspending Replication Activity”, “Removing a Master Site from the Deferred Push List”, and “Forcing the Deferred Transaction Queue to Push List”. Also see Chapter 7, “Resolving an Error Manually”.

4. Migrate the master site using the Oracle7 to Oracle8*i* Migration Utility and execute the `catrep.sql` and `rold_release.sql` (where `old_release` is the previously installed database version) scripts as documented in *Oracle8i Migration*.

5. Using the Replication Manager setup wizard or the replication management API, grant your primary master replication administrator Oracle8*i* Primary Replication Administrator, Propagator, and Receiver privileges for the master site. See "Preparing for Multimaster Replication" on page 2-7 for more information. Database links from the primary replication administrator to the primary master replication administrator at all other Oracle7 and Oracle8*i* master sites should already exist if you prepared your Oracle 7 master site for compatibility with Oracle8*i* using the directions in "Preparing Oracle7 Master Sites for Incremental Migration" on page B-5.

6. If you are not already in a quiesced state, use Replication Manager **o**r the replication management API to suspend all replication activity for all master groups. See "Suspending Replication Activity for a Master Group" on page 2-22 for more information.

7. Using Replication Manager or the replication management API, regenerate replication support for each replicated object. See "Generating Replication Support for Master Group Objects" on page 2-35 for more information. If any sites in the replication environment are still running Oracle7, you must set the “min_communication” parameter to FALSE. The “min_communication” parameter should only be set to TRUE (the default) once all sites have been migrated to Oracle8*i*. For more information, see "Minimizing Data Propagation for Update Conflict Resolution" on page 6-42. Among other activities, generating replication support will establish the registered propagator as the owner of generated objects

8. If you are using procedural replication, check your remaining Oracle7 master sites to determine whether the wrappers have been moved (list created from Step 2). If they have been moved, create a synonym in their old location (in the schema of either the replication administrator or the table owner, depending on whether the site previously used the system-based or user-based model) pointing to the new location in the schema of the primary replication administrator. Confirm necessary object privileges have been granted to access the new owner and locations. If you are also using procedural replication at snapshot sites, regenerate snapshot support on all packages and package bodies used for procedural replication.

9. Using Replication Manager or the replication management API, resume replication activity and unquiesce the environment for each master group. See

"Resuming Replication Activity for a Master Group" on page 2-23 for more information. If you have isolated the master by unscheduling propagation to other masters and from other masters then reschedule propagation by executing DBMS_DEFER_SYS.SCHEDULE_EXECUTION (for Oracle7 sites) or following the instructions in"Editing a Scheduled Link" on page 2-15 (for Oracle8*i* sites) for all master sites.

10. All snapshots at both Oracle7, Oracle8, and greater snapshot sites will need a full refresh after their master site has been migrated to Oracle8*i*. Because of the Oracle8 and greater rowid format, the Oracle7 to Oracle8*i* migration utility truncates all master snapshot logs. If you used the DBMS_JOB.BROKEN procedure to help isolate you master site in Step 3, “unbreak” your jobs to resume your replication activity from your snapshot sites.

    If your snapshots have been defined with the refresh “FORCE” option, their next attempted refresh will full refresh automatically. Snapshots defined with the refresh “FAST” option will need to be manually refreshed using dbms_refresh.refresh or other refresh procedures.

    Note: If you are able to migrate all of the master’s snapshot sites to Oracle8*i* when the master site is migrated to Oracle8*i*, (that is, you do not need to migrate the snapshot sites incrementally) you can alternatively drop the snapshot logs for the master and recreate them as primary key snapshot logs. The snapshots at each snapshot site should be altered to convert them to primary key snapshots. You can then do a full refresh for each primary key snapshot. See "Upgrading to Primary Key Snapshots" on page B-10 for additional details.

11. Drop any administrative accounts and links that you were using to maintain your Oracle7 multimaster replication environment that are not needed in your Oracle8*i* environment. Unnecessary privileges may also be revoked. Be careful not to drop accounts that are needed to maintain any Oracle7 snapshot sites or master sites.

Migration Using Export/ Import

Full database export from Oracle7 Release 7.3.3 or greater and import to Oracle8*i* is supported for both masters and snapshots. You may use export/import as an alternative to the Oracle7 to Oracle8*i* Migration Utility and replication scripts in the procedures described above. Be sure that you follow all the steps, both before and after the actual migration from Oracle7 to Oracle8*i*, in the above procedures however.

To export a full database from Oracle7 Release 7.3.3 or greater and import to Oracle8*i*, follow these steps:

1. Export the Oracle7 Release 7.3.3 or greater database to a dump file using the Release 7.3 export utility under the SYSTEM schema with FULL=y.
2. Import the dump file to the Oracle8*i* database using the Oracle8*i* import utility under the SYSTEM schema with FULL=y.

You may also export data from individual Oracle7 tables, import the data to Oracle8*i* tables, and then configure those tables as masters in an Oracle8*i* replication environment using standard advanced replication procedures.

See the *Oracle8i Utilities* reference guide for more information.

Upgrading to Primary Key Snapshots

Once a snapshot site and its master have been migrated to Oracle8*i*, you can upgrade your rowid snapshots to Oracle8*i* primary key snapshots. To do this you must first alter the snapshot logs for each master table to log primary key information, as well as rowid information, when master rows are updated. Once this is completed at your master site(s), you can incrementally convert your Oracle8*i* snapshots sites by altering the snapshots to convert them to primary key snapshots. Oracle8*i* masters that have been altered to log primary key as well as rowid information can support Oracle7 rowid snapshots as well as Oracle8*i* rowid and primary key snapshots simultaneously to allow for incremental migration.

Note: A primary key snapshot cannot be converted or downgraded to a rowid snapshot.

Primary Key Snapshots Conversion at Master Site(s)

To support primary key snapshots, do the following at the Oracle8*i* master site:

1. Define and enable a primary key constraint on each master table that does not already have a primary key constraint enabled.
2. Alter the snapshot log for each master table supporting fast refresh to include primary key information using the ALTER SNAPSHOT LOG command. See

ALTER SNAPSHOT LOG in the *Oracle8i SQL Reference* manual for additional information.

Note: If the above conditions are not met an error will be raised when you execute the ALTER SNAPSHOT command at the snapshot sites to convert to primary key snapshots.

Primary Key Snapshot Conversion at Snapshot Site(s)

After the Oracle8*i* master site has been configured to support primary key snapshots, do the following at the Oracle8 and greater snapshot sites:

1. Isolate the snapshot site from the replication environment by stopping all local updates to updateable snapshots at the snapshot site.
2. If any read-only ROWID snapshots being converted to primary key snapshots do not include all the columns of the primary key, drop and recreate them with all the primary key columns. See Chapter 3, "Snapshot Concepts & Architecture" for more information.

Note: Constraints should not be defined on Rowid snapshots.

3. Perform a fast refresh of all snapshots to remove the need for any remaining rowid references in the master snapshot log.
4. Use the ALTER SNAPSHOT command to convert rowid snapshots to primary key snapshots. For complete syntax information, see the book *Oracle8i SQL Reference*.
5. Resume replication by rescheduling propagation and/or snapshot refresh with the master, enabling local updates where appropriate. If you used the DBMS_ JOB.BROKEN procedure to help isolate you master site in Step 1, you need to “unbreak” your jobs to resume your replication activity from your snapshot sites.

Features Requiring Migration to Oracle8/Oracle8*i*

The following features require that all the sites involved be successfully migrated to Oracle8:

1. Replication of LOB data types (Oracle8).
2. Reduced data propagation (Oracle8).
   - Use the min_communication parameter and,
   - the send_old_values and compare_old_values procedures.
3. Parallel propagation of deferred transactions (Oracle8).
4. Global authentication and privileged database links (Oracle8).
5. Validate procedure. (Oracle8)

Additional Information: See Appendix A, "New Features".

The following features require that all the sites involved must be successfully migrated to Oracle8 and primary key snapshots:

1. Simple snapshots with subqueries (Oracle8).
2. Master table reorganization procedures (Oracle8).
3. Vertically partitioned snapshots (Oracle8*i*).
4. Deployment Templates (Oracle8*i*).

The following features will automatically work in mixed Oracle7 and Oracle8*i* environments, but only affect Oracle8 sites:

1. Fine grained quiesce.
2. Snapshot registration.

Note: All master groups at Oracle7 sites will be quiesced if any master group at that site is quiesced.

Note: Oracle7 snapshots will not be automatically registered at Oracle8 sites but can be manually registered using the DBMS_ SNAPSHOT.REGISTER_ SNAPSHOT and DBMS_ SNAPSHOT.UNREGISTER_SNAPSHOT procedures at the master site(s). See "Registering a Snapshot at its Master Site" on page 7-36 for more information.

Obsolete procedures

Procedures that are obsoleted in Oracle8*i* include:

DBMS_REPCAT.GENERATE_REPLICATION_PACKAGE

DBMS_REPCAT.GENERATE_REPLICATION_TRIGGER

DBMS_REPCAT_ADMIN.GRANT_ADMIN_REPGROUP

DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_REPGROUP

DBMS_REPCAT_ADMIN.REVOKE_ADMIN_REPGROUP

DBMS_REPCAT_ADMIN.REVOKE_ADMIN_ANY_REPGROUP

DBMS_REPCAT_AUTH.GRANT_SURROGATE_REPCAT

DBMS_REPCAT_AUTH.REVOKE_SURROGATE_REPCAT

DBMS_DEFER_SYS.EXECUTE

Index