Oracle8*i*

Replication

Release 2 (8.1.6)

December 1999

A76959-01

ORACLE

Oracle8i Replication, Release 2 (8.1.6)

A76959-01

Primary Authors: William Creekbaum and Randy Urbano

Graphic Artist: Valarie Moore

Contributors: Sukanya Balaraman, Ruth Baylis, Alan Downing, Al Demers, Ira Greenberg, Maria Pratt, Gordon Smith, Jim Stamos, Curt Elsbernd, Pat McElroy, Wayne Smith, Neerja Shodhan, Mahesh Subramaniam, Lik Wong

Contents

6 Advanced Concepts & Architecture

7 Planning Your Replication Environment

8 Introduction to Replication Manager

A New Features

B Troubleshooting Replication Problems

C Configuring the Oracle8*i* Server for RepAPI

Index

Send Us Your Comments

Oracle8i Replication, Release 2 (8.1.6)

A76959-01

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

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

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

- E-mail - infodev@us.oracle.com
- FAX - 650.506.7228 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation Manager
  500 Oracle Parkway
  Redwood City, CA 94065
  U.S.A.

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

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

Preface

This Preface contains the following topics:

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

> **See Also:** *Getting to Know Oracle8i* for information about the differences between Oracle8*i* Standard Edition, Oracle8*i* Enterprise Edition, and Oracle8*i* Personal Edition.

Overview of the Oracle8*i* Replication Manual

This manual describes Oracle8*i* server replication capabilities. To use Oracle replication, 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:

- Replication
  - Read-only snapshots
  - Updateable snapshots
  - Singe master replication
  - Multimaster replication (requires Orcle8*i* Enterprise Edition)
  - Deployment templates
- Conflict resolution
- Administering a replicated environment
- Advanced techniques
- Troubleshooting
- Using job queues
- Deferred transactions

Audience

This manual is written for database administrators and application developers 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 This Book Is Organized

This manual contains the following chapters and appendices:.

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

Chapter 2, "Master Concepts & Architecture"
Describes the concepts and architecture of multimaster replication.

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

Chapter 4, "Deployment Templates Concepts & Architecture"
Describes the concepts and architecture of deployment templates. This chapter also discusses designing deployment templates.

Chapter 5, "Conflict Resolution Concepts & Architecture"
Describes the concepts and architecture of conflict resolution. This chapter describes conflict resolution methods.

Chapter 6, "Advanced Concepts & Architecture"
Describes advanced replication techniques, including how to do the following: use procedural replication, design for survivability, use dynamic ownership conflict avoidance, and modify tables without replicating the modifications.

Chapter 7, "Planning Your Replication Environment"
Describes planning your replication environment, including information about setting initialization parameters and preparing your environment for replication.

Chapter 8, "Introduction to Replication Manager"
Introduces you to the features of Oracle Replication Manager, a Java-based tool for creating, administering, and monitoring a replication environment.

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

Appendix B, "Troubleshooting Replication Problems"
Describes diagnosing and solving common replication problems.

Appendix C, "Configuring the Oracle8i Server for RepAPI"
Discusses configuring an Oracle8*i* server to be accessed by Java RepAPI clients. Java RepAPI is a runtime library that enables Oracle8*i* Lite clients to replicate data with Oracle servers.

Changes To This Book

The following major changes were made to this book:

- In past releases, this book contained detailed information about using Replication Manager. In the current release, detailed documentation about using Replication Manager is only in the Replication Manager online help. This book contains two new chapters: Chapter 7, "Planning Your Replication Environment" and Chapter 8, "Introduction to Replication Manager".
- Two appendices were added: Appendix B, "Troubleshooting Replication Problems" and Appendix C, "Configuring the Oracle8i Server for RepAPI".
- The appendix on migration and compatibility was moved to the *Oracle8i Migration* book.

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.

See Also: Indicates where to get more information.

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

Text of the Manual

The following sections describe conventions used this manual.

UPPERCASE Characters

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

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

Italicized Characters

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

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

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

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

Code Examples

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

```
INSERT INTO emp (empno, ename) VALUES (1000, 'SMITH');
ALTER TABLESPACE users ADD DATAFILE 'users2.ora' SIZE 50K;
```

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

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

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

Your Comments Are Welcome

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

Server Technologies Documentation Manager

Oracle Corporation

500 Oracle Parkway

Redwood City, CA 94065

U.S.A.

Fax - 650.506.7228 Attn: Server Technologies Documentation Manager

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

infodev@us.oracle.com

1

Replication Overview

This chapter explains the basic concepts and terminology behind Oracle replication. This chapter covers the following topics:

- Introduction to Replication
- Applications That Use Replication
- Replication Objects, Groups, and Sites
- Types of Replication Environments
- Administration Tools for a Replication Environment
- Replication Conflicts
- Other Options for Multimaster Replication

> **Note:** If you are using Trusted Oracle, see your documentation for Oracle security-related products for information about using replication in that environment.

Introduction to Replication

Replication is the process of copying and maintaining database objects, such as tables, 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. Oracle replication is a fully integrated feature of the Oracle server; it is not a separate server.

Replication uses distributed database technology to share data between multiple sites, but a replicated database and a distributed database are not the same. In a distributed database, data is available at many locations, but a particular table resides at only one location. For example, the EMP table might reside at only the DB1 database in a distributed database system that also includes the DB2 and DB3 databases. Replication means that the same data is available at multiple locations. For example, the EMP table might be available at DB1, DB2, and DB3.

Some of the common reasons for using replication are:

Availability Replication improves the availability of applications because it provides them with alternative data access options. If one site becomes unavailable, users can continue to query or even update the remaining locations. In other words, replication provides excellent failover protection.

Performance Replication provides fast, local access to shared data because it balances activity over multiple sites. Some users can access one server while other users access other servers, thereby reducing the load at all servers. Also, users can access data from the replication site that has the lowest access cost, which is typically the site that is geographically closest to them.

Disconnected Computing A *snapshot* is a complete or partial copy (replica) of a target master table from a single point in time. Snapshots enable users to work on a subset of a database while disconnected from the central database server. Later, when a connection is established, users can synchronize (refresh) snapshots on demand. When users refresh snapshots, they update the central database with all of their changes, and they receive any changes that may have happened while they were disconnected.

Network Load Reduction Replication can be used to distribute data over multiple regional locations. Then, applications can access various regional servers instead of accessing one central server. This configuration can reduce network load dramatically.

Mass Deployment Increasingly, organizations need to deploy many applications that require the ability to use and manipulate data. With Oracle replication, deployment templates enable you to create multiple snapshot environments quickly. You can use variables to customize each snapshot environment for its individual needs. For example, you can use deployment templates for sales force automation; in this case, the template could contain variables for various sales regions and salespersons.

You will find more detailed descriptions of the uses of replication in later chapters.

Applications That Use Replication

Replication supports a variety of applications that often have different requirements. Some applications allow for relatively autonomous individual snapshot sites. For example, sales force automation, field service, retail, and other mass deployment applications typically require data to be periodically synchronized between central database systems and a large number of small, remote sites, which are often disconnected from the central database. Members of a sales force must be able to complete transactions, regardless of whether they are connected to the central database. In this case, remote sites must be autonomous.

On the other hand, applications such as call centers and Internet systems require data on multiple servers to be synchronized in a continuous, nearly instantaneous manner to ensure that the service provided is available and equivalent at all times. For example, a retail web site on the Internet must ensure that customers see the same information in the online catalog at each site. Here, data consistency is more important than site autonomy.

Oracle replication can be used for both types of applications, and for applications that combine aspects of both. In fact, one Oracle replication environment can support both mass deployment and server-to-server replication, which enables integration into one coherent environment. In such an environment, for example, sales force automation and customer service call centers can share data.

Replication Objects, Groups, and Sites

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

Replication Objects

A *replication object* is a database object existing on multiple servers in a distributed database system. In a replication environment, any updates made to a replication object at one site are applied to the copies at all other sites. Oracle replication enables you to replicate the following types of objects:

- Tables
- Indexes
- Views
- Packages and Package Bodies
- Procedures and Functions
- Triggers
- Sequences
- Synonyms

Replication Groups

In a replication environment, Oracle manages replication objects using *replication groups*. A replication group is a collection of replication objects that are logically related. The objects in a replication group are administered together.

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. However, replication groups and schemas do not need to correspond with one another. A replication group can contain objects from multiple schemas, and a single schema can have objects in multiple replication groups. However, a replication object can be a member of only one replication group.

Replication Sites

A replication group can exist at multiple *replication sites*. Replication environments support two basic types of sites: *master sites* and *snapshot sites*. One site can be both a master site and a snapshot site at the same time.

The differences between master sites and snapshot sites are:

- A replication group at a master site is more specifically referred to as a *master group*. A replication group at a snapshot site is more specifically referred to as a *snapshot group*. Additionally, every master group has exactly one *master definition site*. A replication group's master definition site is a master site serving as the control center for managing the replication group and the objects in the group.
- A master site maintains a complete copy of all objects in a replication group, while snapshots at a snapshot site can contain all or a subset of the table data within a master group. For example, if the SCOTT_MG master group contains the tables EMP and DEPT, all of the master sites must maintain a complete copy of EMP and DEPT. However, one snapshot site might contain only a snapshot of the EMP table, while another snapshot site might contain snapshots of both the EMP and DEPT tables.
- All master sites in a multimaster replication environment communicate directly with one another to continually propagate data and schema changes in the replication group. Snapshot sites contain an image, or snapshot, of the table data from a certain point in time. Typically, a snapshot is refreshed periodically to synchronize it with its master site. You can organize snapshots into *refresh groups*. Snapshots in a refresh group can belong to one or more snapshot groups, and they are refreshed at the same time to ensure that the data of all snapshots in the refresh group correspond to the same transactionally consistent point in time.

Types of Replication Environments

Oracle replication supports the following types of replication environments:

- Multimaster Replication
- Snapshot Replication
- Multimaster and Snapshot Hybrid Configurations

Multimaster Replication

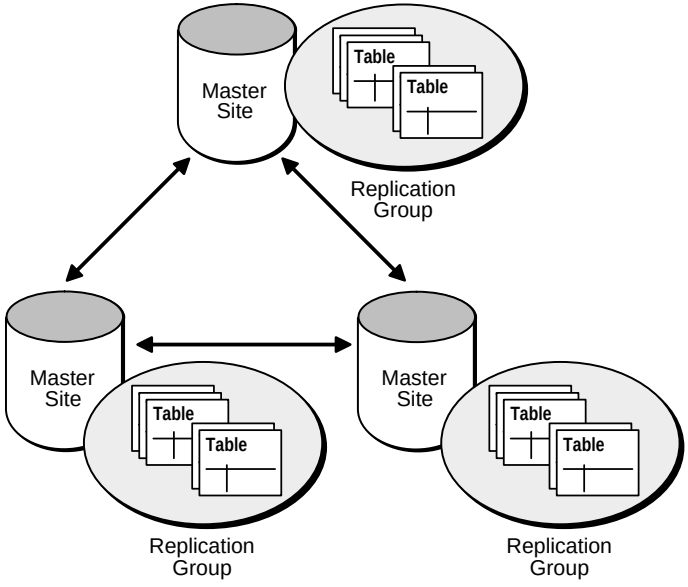
Multimaster replication (also called peer-to-peer or *n*-way replication) allows multiple sites, acting as equal peers, to manage groups of replicated database objects. Each site in a multimaster replication environment is a master site.

Applications can update any replicated table at any site in a multimaster configuration. Oracle database servers operating as master sites in a multimaster environment automatically work to converge the data of all table replicas and to ensure global transaction consistency and data integrity.

Asynchronous replication is the most common way to implement multimaster replication. Other ways include synchronous replication and procedural replication, which are discussed later in this chapter. When you use asynchronous replication, an update of a table is stored in the deferred transaction queue at the master site where the change occurred. These changes are called *deferred transactions*. The deferred transactions are pushed (or propagated) to the other participating master sites at regular intervals. You can control the amount of time in an interval.

Using asynchronous replication means that conflicts are possible because the same row value might be updated at two different master sites at nearly the same time. However, you can use techniques to avoid conflicts and, if conflicts occur, Oracle provides built-in mechanisms to resolve them.

Figure 1–1 Multimaster Replication



Quiescing Master Groups

At times, you must stop all replication activity for a master group so that you can perform certain administrative tasks on the master group. For example, you must stop all replication activity for a master group to issue data definition language (DDL) statements on any table in the group. Stopping all replication activity for a master group is called *quiescing* the group. When a master group is quiesced, users cannot perform data manipulation language (DML) statements on any of the objects in the master group.

Snapshot Replication

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

All snapshots provide the following benefits:

- Enable local access, which improves response times and availability.
- Offload queries from the master site, because users can query the local snapshot instead.
- Increase data security by allowing you to replicate only a selected subset of the target master table's data set.

Read-Only Snapshots

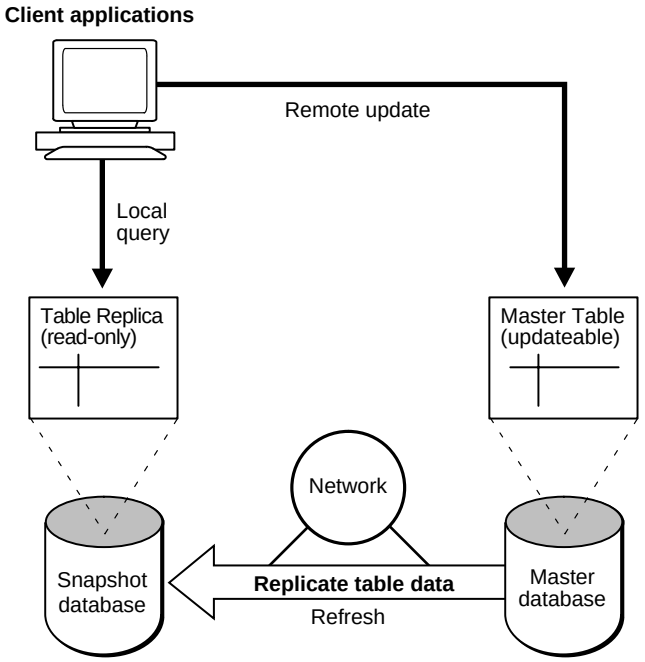
In a basic configuration, snapshots can provide read-only access to the table data that originates from a master site. Applications can query data from read-only snapshots to avoid network access regardless of network availability. However, applications throughout the system must access data at the master site to perform an update. Figure 1–2 illustrates basic, read-only replication. The master tables of read-only snapshots do not need to belong to a master group.

Read-only snapshots provide the following benefits:

- Eliminate the possibility of conflicts because they cannot be updated.
- Support complex snapshots. Examples of complex snapshots are snapshots that contain set operations or a CONNECT BY clause.

See Also: "Available Snapshots" on page 3-4 for more information about complex snapshots.

Figure 1–2 Read-Only Snapshot Replication

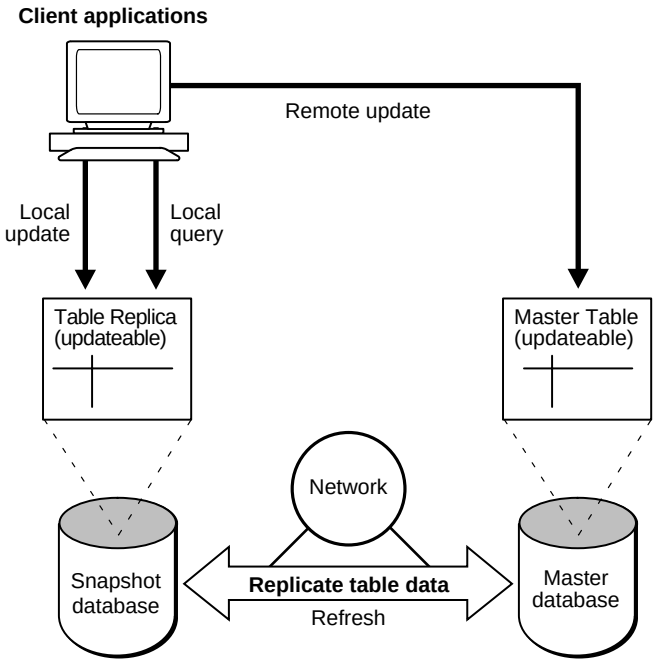


Updateable Snapshots

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 by performing these operations on the snapshot. An updateable snapshot may also contain only a subset of the data in the target master table. Figure 1–3 illustrates a replication environment using updateable snapshots.

Updateable snapshots are based on tables at a master site that have been set up to support replication. In fact, updateable snapshots must be part of a snapshot group that is based on a master group at a master site.

Figure 1–3 Updateable Snapshot Replication



Updateable snapshots have the following properties.

- Updateable snapshots are always based on a single table.
- Updateable snapshots can be incrementally (or "fast") refreshed.
- Oracle propagates the changes made to an updateable snapshot to the snapshot's remote master table. If necessary, the updates to the master table then cascade to all other master sites.
- Oracle can refresh an updateable snapshot as part of a refresh group in the same way it refreshes read-only snapshots.

Updateable snapshots provide the following benefits:

- Allow users to query and update a local replicated data set even when disconnected from the master site.
- Require fewer resources than multimaster replication, while still supporting data updates. For example, because snapshots can reside on an Oracle8*i* Lite database, the disk space and memory requirements for snapshot clients are much less than the requirements for an Oracle8*i* server.

Snapshot Refresh

To ensure that a snapshot is consistent with its master table, you need to *refresh* the snapshot periodically. Oracle provides the following three methods to refresh snapshots:

- *Fast refresh* uses snapshot logs to update only the rows that have changed since the last refresh.
- *Complete refresh* updates the entire snapshot.
- *Force refresh* performs a fast refresh when possible. When a fast refresh is not possible, force refresh performs a complete refresh.

When it is important for snapshots to be transactionally consistent with each other, you can organize them into *refresh groups*. By refreshing the refresh group, you can ensure that the data in all of the snapshots in the refresh group correspond to the same transactionally consistent point in time. A snapshot in a refresh group still can be refreshed individually, but doing so nullifies the benefits of the refresh group because refreshing the snapshot individually does not refresh the other snapshots in the refresh group.

Snapshot Log

A *snapshot log* is a table that records all of the DML changes to a master table. A snapshot log is associated with a single master table, and each master table has only one snapshot log, regardless of how many snapshots refresh from the master. A fast refresh of a snapshot is possible only if the snapshot's master table has a snapshot log. When a snapshot is fast refreshed, entries in the snapshot's associated snapshot log that have appeared since the snapshot was last refreshed are applied to the snapshot.

Deployment Templates

Deployment templates simplify the task of deploying and maintaining many remote snapshot sites. Using deployment templates, you can define a collection of snapshot definitions at a master site, and you can use parameters in the definitions so that the snapshots can be customized for individual users or types of users.

For example, you might create one template for the sales force and another template for field service representatives. In this case, a parameter value might be the sales territory or the customer support level. When a remote user connects to a master site, the user can query a list of available templates. When the user *instantiates* a template, the appropriate snapshots are created and populated at the remote site. The appropriate parameter values can either be supplied by the remote user or taken from a table maintained at the master site.

Online and Offline Instantiation When a user instantiates a template at a snapshot site, the object DDL (for example, CREATE SNAPSHOT... or CREATE TABLE...) is executed to create the appropriate schema objects at the snapshot site, and the objects are populated with the appropriate data.

Users can instantiate templates while connected to the master site over a network (online instantiation), or while disconnected from the master site (offline instantiation).

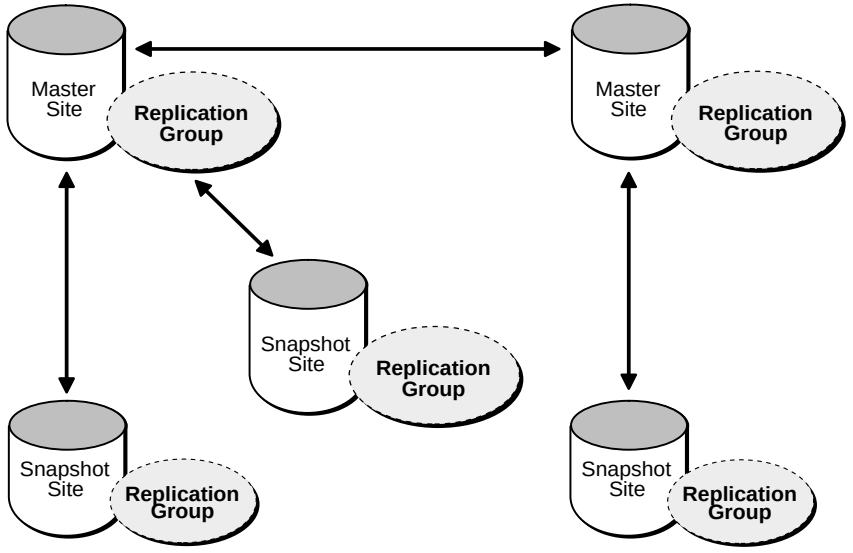
Offline instantiation is often used to decrease server loads during peak usage periods and to reduce remote connection times. To instantiate a template offline, you package the template and required data on some type of storage media, such as tape, CD-ROM, and so on. Then, instead of pulling the data from the master site, users pull the data from the storage media containing the template and data.

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 1–4, multimaster (or *n*-way) 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 1–4 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.
- Master sites detect and resolve the conflicts that occur from changes made to multiple copies of the same data.

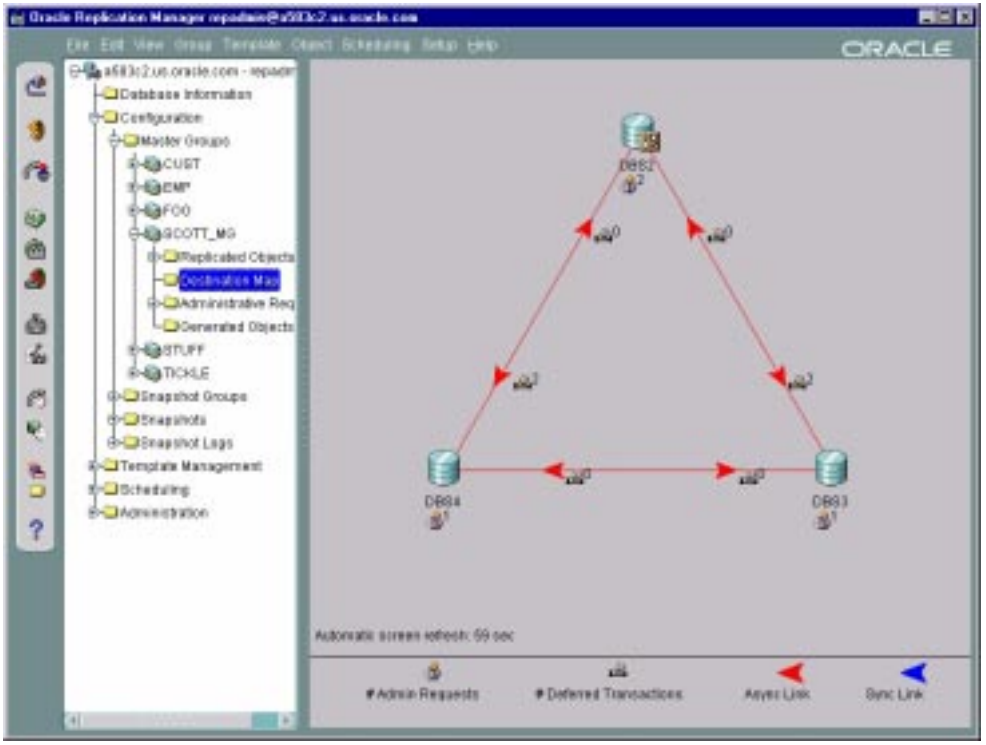
Administration Tools for a Replication Environment

Several tools are available for administering and monitoring 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.

Oracle Replication Manager

Replication environments supporting both multimaster and snapshot replication can be challenging to configure and manage. To help administer these replication environments, Oracle provides a sophisticated management tool called Oracle Replication Manager. Other sections in this book include information and examples for using Replication Manager. However, the Replication Manager online help is the primary documentation source for Replication Manager.

Figure 1–5 Replication Manager



See Also: Chapter 8, "Introduction to Replication Manager" for an introduction to Replication Manager, and see the Replication Manager online help for complete instructions on using Replication Manager.

Replication Management API

The replication management application programming interface (API) is a set of PL/SQL packages that encapsulate procedures and functions that you can use to configure an Oracle replication environment. The replication management API is a command-line alternative to Replication Manager. In fact, Replication Manager uses the procedures and functions of the replication management API to perform its work. 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. The replication management API makes it easy for you to create custom scripts to manage your replication environment.

> **See Also:** *Oracle8i Replication Management API Reference* for more information about using the replication management API.

Replication Catalog

Every master and snapshot site in a replication environment has a *replication catalog*. A replication catalog for a site 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.

> **See Also:** *Oracle8i Replication Management API Reference* for more information about the replication catalog.

Distributed Schema Management

In a replication environment, all DDL statements must be issued using either Replication Manager or the DBMS_REPCAT package. When you use either of these interfaces, all DDL statements are replicated to all of the sites participating in the replication environment.

Note: Any DDL statements issued directly using a tool such as SQL*Plus are not replicated to other sites.

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 occur, you need a mechanism to ensure that the conflict is resolved in accordance with your business rules and to ensure 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 built-in conflict resolution methods that enable you to define a conflict resolution system for your database that resolves conflicts in accordance with your business rules. If you have a unique situation that Oracle's built-in conflict resolution methods cannot resolve, you have the option of building and using your own conflict routines.

See Also:

- Chapter 5, "Conflict Resolution Concepts & Architecture" for information about how to design your database to avoid data conflicts and how to build conflict resolution routines that resolve such conflicts when they occur.
- The online help for Replication Manager for instructions on using Replication Manager to configure conflict resolution methods.
- Chapter 6, "Conflict Resolution" of the *Oracle8i Replication Management API Reference* for a description of how to build conflict resolution routines using the replication management API.

Other Options for Multimaster Replication

Asynchronous replication is the most common way to implement multimaster replication. However, you have two other options: synchronous replication and procedural replication.

Synchronous Replication

A multimaster replication environment can use either asynchronous or synchronous replication to copy data. With asynchronous replication, changes made at one master site occur at a later time at all other participating master sites. With synchronous replication, changes made at one master site occur immediately at all other participating master sites.

When you use synchronous replication, an update of a table results in the immediate replication of the update at all participating master sites. In fact, each transaction includes all master sites. Therefore, if one master site cannot process a transaction for any reason, the transaction is rolled back at all master sites.

Although you avoid the possibility of conflicts when you use synchronous replication, it requires a very stable environment to operate smoothly. If communication to one master site is not possible because of a network problem, for example, no transactions can be completed until communication is re-established.

Procedural Replication

Batch processing applications can change large amounts of data within a single transaction. In such cases, typical row-level replication might load a network with many 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. It does not replicate the data modifications themselves.

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 the 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 to either avoid or detect replication conflicts and to resolve them appropriately. Consequently, procedural replication is most typically used when databases are modified only with large batch operations. In such situations, replication conflicts are unlikely because numerous transactions are not contending for the same data.

> **See Also:** "Using Procedural Replication" on page 6-2 for more information.

2

Master Concepts & Architecture

This chapter explains the concepts and architecture of Oracle's master replication sites in both a single master and multimaster replication environments. This chapter covers the following topics:

- Master Replication Concepts
- Master Replication Architecture

Master Replication Concepts

To understand the architectural details of master replication, you need to understand concepts of master replication. Knowing how and why replication is used provides you with a greater understanding of how the individual architectural elements work together to create a multimaster replication environment.

What is Master Replication?

Oracle has two forms of master replication: single master replication and multimaster replication.

Single master replication involves a master site supporting one or more snapshot sites. All of the connected snapshots update their master site; all data conflict detection and resolution is performed at the master site.

> **See Also:** Chapter 3, "Snapshot Concepts & Architecture" for more information about snapshots.

Oracle's *multimaster replication*, also known as peer-to-peer or n-way 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 2–1 illustrates a multimaster replication system.

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

Master Sites

A master site can be both a node in a multimaster replication environment and/or the master for one or more snapshot sites in a single master replication environment. The replicated objects are stored at the master site and are available for user access.

Master Definition Site In a multimaster replication environment, a single master site operates as the master definition site for a master group. This particular site performs many of the administrative and maintenance tasks for the multimaster replication environment.

There can be only one master definition site for a master group, though the master definition site can be any of the master sites in the multimaster environment. Additionally, the master definition site can be changed to a different master site if necessary.

A single master site supporting snapshot replication is by default the master definition site.

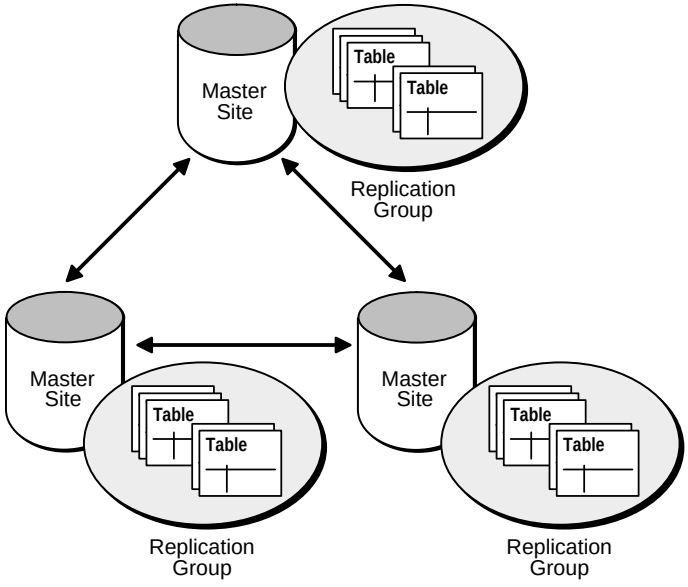
Why Use Multimaster Replication?

From a very basic point of view, replication is used to make sure that information (data) is available when and where you need it. The following sections describe several different environments that have different information delivery requirements. Perhaps your replication environment also has one or more of the following requirements.

Failover

Multimaster replication can be used 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 also can serve as a fully functional database to support application access when the primary site is concurrently operational, whereas a standby database can only become fully functional if the primary site is unavailable.

Figure 2–1 Multimaster Replication



You can use Net8 to configure automatic connect-time failover, which enables Net8 to fail over to a different master site if the first master site fails. You configure automatic connect-time failover in your `tnsnames.ora` file by setting the FAILOVER option to ON and specifying multiple connect descriptors.

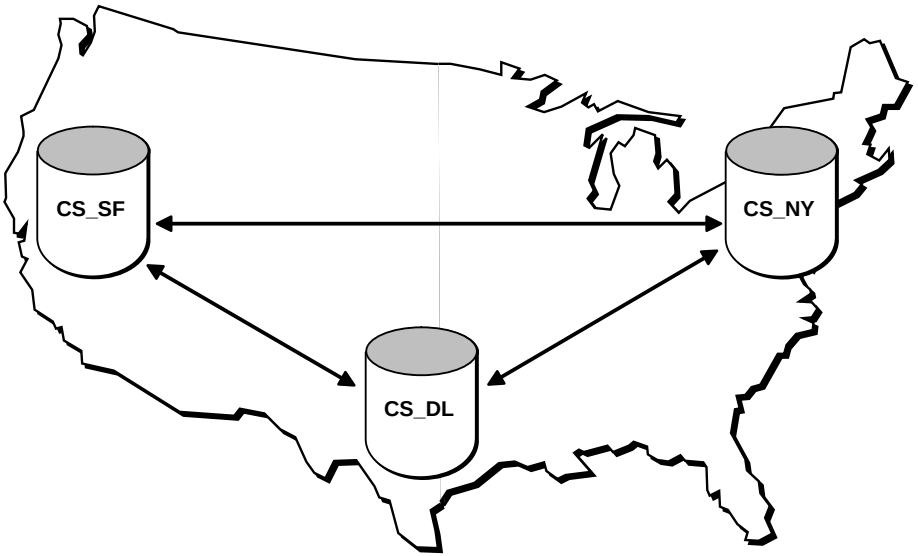
> **See Also:** *Net8 Administrator's Guide* for more information about configuring connect-time failover.

Load Balancing

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.

Figure 2–2 Multimaster Replication Supporting Multiple Points of Update Access



Parallel Server vs. Replication

There are two major areas where you may need to consider whether Oracle replication or Oracle Parallel Server better serves your needs; these areas are load balancing and survivability.

Load Balancing: Oracle Parallel Server provides read and write load balancing over multiple instances. Oracle replication provides read load balancing over multiple databases, though the net effect of the write load balancing is that each write is performed at each master site.

Survivability: Replication provides greater survivability protection with regards to natural disasters, power outages, and / or sabotage because the remaining replication sites may be positioned in a geographically different region. Oracle Parallel Server operates on a cluster or other massively parallel system and is located in the same physical environment, and thus cannot protect against the physical problems that replication can protect against.

Support for Disconnected Snapshot Environments

Snapshot replication allows users to remotely store all or a subset of replicated data from a master site in a disconnected environment. This scenario is typical of sales force automation systems where an individual's laptop (a disconnected device) stores a subset of data related to the individual salesperson.

Master sites operate as the target of the snapshot environment. Master site support can be:

- A single master site supporting all of the snapshots, which eliminates the potential of divergent data because all conflict resolution is performed at the single master site.
- A combination of multimaster and snapshot replication where groups of snapshots are targeted to different masters of the multimaster configuration. This configuration distributes the network load across multiple master nodes, providing improved scalability and availability should one of the master nodes become unavailable.

Types of Master Replication

There are two types of master replication: multimaster and single master. Multimaster replication includes multiple master sites, where each master site operates as an equal peer. In single master replication, a single master site supporting snapshot replication provides the mechanisms to support potentially hundreds or thousands of snapshot sites. A single master site that supports one or more snapshot sites can also participate in a multiple master site environment, creating a hybrid replication environment (combination of multimaster and snapshot replication).

Multimaster Replication

Multimaster replication is comprised of multiple master sites equally participating in an update-anywhere model. Updates made to an individual master site are propagated to all other participating master sites.

Conflict resolution is independently handled at each of the master sites. Multimaster replication provides complete replicas of each replicated table at each of the master sites.

Single Master Replication

A single master site can also function as the target master site for one or more snapshot sites. Unlike multimaster replication, where updates to a single site are propagated to all other master sites, snapshots update only their target master site. If the replication environment is a hybrid environment (it has multiple master sites supporting one or more snapshot sites), the target master site propagates any of the snapshot updates to all other master sites in the multiple site replication environment.

Conflict resolution is handled only at master sites. Snapshot replication can contain complete or partial replicas of the replicated table.

> **See Also:** Chapter 3, "Snapshot Concepts & Architecture" for more information about snapshot replication with a master site.

Multimaster Replication Process

There are two types of multimaster replication: asynchronous and synchronous.

Asynchronous replication, often referred to as *store-and-forward* replication, captures any local changes, stores them in a queue, and, at regular intervals, propagates and applies these changes at remote sites. With this form of replication, there is a period of time before all sites achieve data convergence.

Synchronous replication, also known as *real-time replication*, applies any changes or executes any replicated procedures at all sites participating in the replication environment as part of a single transaction. If the DML or procedure fails at any site, the entire transaction rolls back. Synchronous replication ensures data consistency at all sites in real-time.

You can change the propagation mode from asynchronous to synchronous or vice versa for a master site. If you change the propagation mode for a master site in a master group, you must regenerate replication support for all master group objects. Also, a multimaster replication environment may contain a mixture of both synchronous and asynchronous replication.

> **See Also:** "Understanding Mixed-Mode Multimaster Systems" on page 2-33 for more information.

Asynchronous Replication

Asynchronous replication independently propagates (sends) any DML or replicated procedure execution to all of the other master sites participating in the multimaster replication environment. Propagation occurs in a separate transaction after the DML or replication procedure has been executed locally.

Asynchronous replication is the default mode of replication. Asynchronous replication requires less networking and hardware resources than does synchronous replication, resulting in better availability and performance.

Asynchronous replication, however, means that the data sets at the different master sites in the replication environment may be different for a period of time before the changes have been propagated. Also, data conflicts may occur in an asynchronous replication environment.

The following describes the process of asynchronous replication:

- **User issues DML or executes a wrapper for a replicated procedure.**

  Once a table has been set up for replication, any DML that a user issues on the table is captured for replication to all other master sites.

  For each row that is inserted, updated, or deleted, an internal trigger creates a deferred *remote procedure call (RPC)*. The deferred transaction queue contains all deferred RPCs.

  If a procedure has been replicated and its wrapper is executed at a master site, the procedure call is also placed in the deferred transaction queue.

- **Deferred Transaction Queue stores deferred RPCs.**

  Each transaction in the deferred transaction queue has a list of destinations that define where the deferred transaction should be propagated; this list contains all master sites except for the originating site.

- **Propagation sends deferred transaction queue entry to destination.**

  At scheduled intervals or on-demand, the deferred transactions in the deferred transaction queue are propagated to the target destinations.

- **Deferred transaction queue entry applied at remote destination.**

  As a deferred transaction is being propagated to a target destination, each deferred RPC is applied at the destination site. If the deferred transaction cannot be successfully applied at the destination site, it is resent and placed into the error queue at the destination site, where the DBA can fix the error condition and re-apply the deferred transaction.

  When a deferred transaction queue entry is applied at the remote destination, Oracle checks for data conflicts. If a conflict is detected, it is logged and optionally a conflict resolution method is invoked.

  > **See Also:** Chapter 5, "Conflict Resolution Concepts & Architecture" for more information.

Synchronous Replication

Synchronous replication propagates any changes made at a local site to other synchronously linked masters in a replication environment during the same transaction as the initial change. If the propagation fails at any of the master sites, the entire transaction, including the initial change at the local master site, rolls back. This strict enforcement ensures data consistency across the replication environment.

Unlike asynchronous replication, there is never a period of time when the data at any of the master sites does not match.

> **See Also:** "Understanding Mixed-Mode Multimaster Systems" on page 2-33 for a discussion on using both synchronous and asynchronous replication in a single environment.

Synchronous replication also ensures that no data conflicts are introduced into the replication environment. These benefits have the cost of requiring many hardware and networking resources with no flexibility for downtime. Consider, if a single master site of a six node multimaster environment is unavailable, a transaction is not completed. Whereas in asynchronous replication, the deferred transaction is held until the downed site becomes available.

Additionally, while query performance remains high because they are performed locally, updates are slower because of the 2-phase commit protocol that ensures that any updates are successfully propagated and applied to the remote destination sites.

The following describes the process of synchronous replication:

- **User issues DML or executes a wrapper for a replicated procedure.**

  Once a table has been setup for replication, any DML that a user issues on the target table is captured for replication to all other master replication sites.

  If a procedure has been replicated and its wrapper is executed at a master site, the procedure call is also captured for replication.

- **DML or wrapper execution is immediately propagated to destination sites.**

  The internal trigger captures any DML and immediately propagates these actions to all other master sites in the replication environment. The internal trigger applies these actions in the security context of the propagator's database link and uses an internal RPC to apply these actions at the destination site.

  Like an internal trigger, a wrapper for a replicated procedure immediately propagates the procedure call to all other master sites in the replication environment.

  If the transaction fails at any one of the master replication sites, then the transaction is rolled back at all master sites. This methodology ensures data consistency across all master replication sites. Because of the need to rollback a transaction if any site fails, synchronous replication is extremely dependent on highly-available networks, databases, and the associated hardware.

Conflict Resolution Concepts

When Oracle replicates a table, any DML applied to the replicated table at any replication site (either master or snapshot site) that causes a data conflict is automatically detected by the Oracle server at a master site. Snapshot sites cannot detect nor resolve data conflicts. Any data conflicts introduced by a snapshot site are detected and resolved at the target master site of the snapshot.

For example, if the following master group is scheduled to propagate changes once an hour, consider what happens when:

Time	Master Site A	Master Site B	Status
8:00 AM	Propagate Changes to Master Site B	Propagate Changes to Master Site A	Data converges.
8:15 AM	Updates Row 1		
8:30 AM		Updates Row 1	
9:00 AM	Propagate Changes to Master Site B	Propagate Changes to Master Site A	Conflict Detected on Row 1

If the time between propagations is considered an interval and two or more sites update the same row during a single interval, a conflict occurs.

In addition to the update conflict described above, there are insert and delete conflicts. Consider the following summaries of each type of conflict. Each conflict occurs when the conflicting actions occur within the same interval:

- Update conflict: two or more updates are applied to the same row before the preceding update can be propagated to the other sites.
- Uniqueness conflict: an insert is performed at two or more sites and the primary key (or other set of unique columns) for each insert contains the same value, or an update at one site modifies the primary key (or other set of unique columns), which contains the same value as an insert at another site.
- Delete conflict: a row is deleted at one site and an update occurs at another site, which may result in an attempt to update a row that does not exist, or the same row is deleted in the same interval at more than one site.

> **See Also:** Chapter 5, "Conflict Resolution Concepts & Architecture" for more information about the different types of data conflicts.

Once a data conflict is detected, the following actions occur:

- The conflict resolution methods try to resolve the data conflict.
- If the conflict is not resolved, the data conflict is logged in the error queue at the destination site.

If the data conflict is logged in the error queue, the database administrator is responsible for resolving the data conflict manually.

If you choose to use Oracle or user-defined conflict resolution methods, the Oracle server automatically tries resolve the data conflict. The conflict resolution methods that are implemented should conform to the business rules defined for your replicated environment and should work to guarantee data convergence.

Master Replication Architecture

Though you can build a replicated environment by following the procedures and examples described in this book, in the Replication Manager online help, and in the *Oracle8i Replication Management API Reference*, understanding the architecture of replication gives you valuable information for setting up your database environment to support replication, tuning your replication environment, and troubleshooting your replication environment when necessary. This section describes the architecture of replication in terms of mechanisms and processes.

Master Site Mechanisms

To support a replicated environment, Oracle uses the following mechanisms at each master site that is participating in either a multimaster replication or single master replication environment. Some of the following master site mechanisms are required only in special circumstances.

Master Site Roles/Users

Depending on your security requirements, the following three roles may be consolidated into a single replication administrator. In fact, most multimaster replication environments use a single user to perform the replication administration, propagation, and receiving roles. If you have more stringent security requirement, you may assign the following roles to different users.

Note: The term "roles" in this context is not related to the SQL term "roles." The referenced replication roles are granted using stored PL/SQL procedures and/or individual privileges.

Replication Administrator The replication administrator performs all of the administrative functions relating to a master site in a replication environment. In general, it is preferable to have a single replication administrator for a replication environment. In addition to preparing a database to support replication, the replication administrator is responsible for building and maintaining the individual master replication groups, adding and removing participating master sites, managing the queues, and controlling the state of the replication environment (normal and quiesced). The default username for this administrator is REPADMIN, but you can use any username you wish.

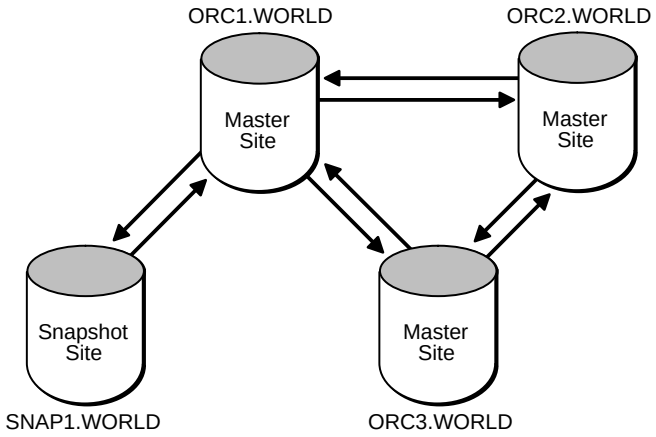
Propagator The propagator performs the task of propagating each transaction contained in the deferred transaction queue to the transaction's destinations. There is a single propagator for the database. In other words, it is possible for you to have multiple replication administrators to manage different schemas, but there can only be a single propagator per database.

Receiver The receiver is responsible for receiving and applying the deferred transactions from the propagator. If the receiver does not have the appropriate privileges to apply a call in the deferred transaction, the entire deferred transaction is placed in the error queue at the destination.

Database Links

Database links provide the conduit to replicate data between master and snapshot sites. In a multimaster environment, there is a database link from each individual master site to all other master sites. Another way to look at the configuration of database links is that there are N - 1 database links for each master site, where N is the total number of master sites.

Figure 2–3 Each Arrow Represents a Database Link



In Figure 2–3, each master site has two database links to the other master sites (N-1 or in this case 3-1=2). This configuration ensures the bi-directional communication channels between master sites needed for multimaster replication.

The most basic setup would find a database link from the replication administrator at the individual master site to the replication administrators at the other participating master replication sites.

A common approach, however, adds an additional set of database links to your replication environment. Before creating any replication administrator database links, you would create public database links between all of the participating master sites, without specifying a CONNECT TO clause. The public database link would specify the target of each database link with the USING clause.

Once the public database links have been created, you can create the private replication administrator database links. Though you still must specify the CONNECT TO clause, setting up public database links eliminates the need to specify a USING clause.

The approach of using both public and private database links reduces the amount of administration needed to manage database links. Consider the following advantages:

- If the target database of a database link changes, you only need to alter the USING clause of the public database link. All private database links for the same target point to the destination defined in the USING clause in the public database link.
- Multiple sets of private database links can share the same public link, further simplifying the administration of database links.

As previously described, the replication administrator usually performs the tasks of administration and propagation in a multimaster environment. Because these tasks are performed by only a single user, only a set of private database links must be created for the replication administrator.

However, in multimaster replication environments where propagation is performed by users other than the replication administrator, the appropriate set of private database links must be created for these alternate users.

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

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

For the purposes of replication, connection qualifiers can also enable you to more closely control the propagation characteristics for multiple master groups. Consider, if each master site contains three separate master groups and you are not using connection qualifiers, the scheduling characteristics for the propagation of the deferred transaction queue is the same for all master groups. This may be costly if one master group propagates deferred transactions once an hour while the other two master groups propagate deferred transactions once a day.

Associating a connection qualifier with a master group gives you the ability to define different scheduling characteristics for the propagation of the deferred transaction queue on a master group level versus on a database level as previously described.

> **See Also:** 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. After a master group is created, you cannot remove, add, or change the connection qualifier for the group.

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 groups with different connection qualifiers.

Caution: If you plan to use connection qualifiers, you probably need to increase the value of the OPEN_LINKS initialization parameter at all master sites. The default is four open links per process. Estimate the required value based on your usage. See "Initialization Parameters" on page 7-3, and see the *Oracle8i Reference*, for more information about OPEN_LINKS.

Replication Objects

The most visible part of your replicated environment is the replicated objects themselves. Of these replicated objects, replicated tables are the foundation of your replicated environment. The following sections discuss replicating the related database objects. These discussions highlight the benefits and potential limitations of replicating particular types of database objects.

Tables In most cases, replicated tables are the foundation of your replication environment. Once a table is selected for replication and has had replication support generated, it is monitored by internal triggers to detect any DML applied to it (see "Internal Triggers" on page 2-19).

When you replicate a table, you have the option of replicating the table structure and table data to the remote data sites or just the table structure. Additionally, if a table of the same name and structure already exists at the target replication site, you have the option of using the existing object in your replication environment.

Note: On tables with self-referential integrity constraints, Oracle replication cannot guarantee that the deletes will be performed in the correct order. To perform deletes on tables with self-referential integrity constraints, use procedural replication. See "Using Procedural Replication" on page 6-2 for information.

Alternative Uses for Table Replication

Though replicating a table is intended for replicating any table data changes to all sites participating in the replication environment, there are other uses for replicating a table.

Object and Data Transport: After an object has been replicated to a target destination site, replication support is not automatically generated. You can use this approach as an easy way to distribute objects and data to remote destinations. If you do not drop the replication objects and do not generate replication support, the table (or other objects) and the data remains at the remote destination site, and any changes at the remote destination site are not replicated. This approach enables you to distribute a standard database environment and data set to a new database environment.

Object Transport: Similarly, you can replicate a table to a target destination site without copying the data. This approach creates the object at the destination site, but does not populate it with data. Therefore, you can quickly distribute an empty database environment.

Indexes Any index that is used to enforce a constraint in a table is automatically created at the remote destination sites when a table is selected for replication and created at the remote site. Any index that is used for performance reasons, however, must be explicitly selected for replication to be created at the other master sites participating in the replication environment. When an index is replicated to other sites, it operates as if the index was created locally. There are no replication mechanisms that monitor the trigger to capture any changes.

Packages/Package Bodies Selecting packages and package bodies for replication and generating the needed replication support gives you the ability to do procedural replication. Procedural replication can offer performance advantages for large, batch-oriented operations on large numbers of rows that can be run serially within a replicated environment.

All parameters for a procedure with replication support 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. A replicated procedure must be declared in a package. Stand-alone procedures cannot have replication support.

See Also: "Using Procedural Replication" section on page 6-2 for detailed information about using procedural replication.

Note: Similar to the concepts presented in the "Alternative Uses for Table Replication" sidebar on page 2-16, you can select a package and package body for replication but not generate replication support to use replication as an easy way to distribute the object to a remote site, though any calls made to the package are not replicated.

Procedures/Functions Procedures and functions not declared as part of a package cannot have replication support. Though you cannot create a procedural replication environment with stand-alone procedures and functions, you can still use replication to distribute these stand-alone procedures and functions to the sites in your replication environment. When the stand-alone procedure or function is created at the remote site using replication, the created object does not have replication support and operates as though the object was created locally.

Triggers To make sure that any application or database logic is present at each master site, you can select triggers for replication. An important example of replicating a trigger is replicating a trigger that automatically inserts a timestamp into a table when any DML is applied to the table.

To avoid refiring of the trigger, it is important to insert an API call into the trigger to detect if the trigger is being fired through a local or remote call. This is to avoid the situation where the trigger updates a row that causes the trigger to fire again. Notice line 5 in the following code example:

```
1) CREATE TRIGGER scott.insert_time
2)    BEFORE
3)       INSERT OR UPDATE ON scott.emp FOR EACH ROW
4)    BEGIN
5)       IF DBMS_REPUTIL.FROM_REMOTE = FALSE THEN
6)          :NEW.TIMESTAMP := SYSDATE;
7)       END IF;
8)    END;
```

If the DBMS_REPUTIL.FROM_REMOTE function determines that the insert or update was locally initiated, then the defined action (that is, assign timestamp) occurs. If this function determines that the insert or update is from a remote site, then the timestamp value is not assigned.

Sequences Because two sequences at different databases can generate the same value, replicating sequences is not supported.

There are three alternatives to replicating sequences that guarantee to generate unique values and avoid any uniqueness data conflicts. Beginning with Oracle8*i* release 8.1.5, you can retrieve a unique identifier by executing the following select statement:

```
SELECT sys_guid() FROM dual;
```

This SQL operator returns a 16-byte globally unique identifier. This value is based on an algorithm that uses time and datestamp and machine identifier to generate a globally unique identifier. The globally unique identifier appears in a format similar to the following:

```
4595EF13AB785E73E03400400B40F58B
```

An alternate solution to using the sys_guid() function is to create a sequence at each of the master sites and concatenate the site name (or other globally unique value) with the local sequence. This approach helps you to avoid any potential duplicate sequence values and helps in preventing insert conflicts as described in the "Conflict Resolution Concepts" section on page 2-10.

Additionally, you can create a sequence at each of the master sites so that each generates a unique value in your replication environment. You can accomplish this by using a combination of starting, incrementing, and maximum values. For example, you might configure the following:

Parameter	Master Site A	Master Site B	Master Site C
start_with	1	3	5
increment_by	10	10	10
Range Example	1, 11, 21, 31, 41,...	3, 13, 23, 33, 43,...	5, 15, 25, 35, 45,...

Using a similar approach, you can define different ranges for each master site by specifying a START_WITH and MAXVALUE that would produce a unique range for each site.

Views/Synonyms When you replicate a view or synonym, you are simply using replication to distribute these objects to the other master sites that are involved in the replication environment. Once the object is replicated to the other sites, it operates as if the view or synonym was created locally. There is no internal trigger or package that monitors the object to capture any changes. Because it is a replicated object, though, you can still drop or modify it using either Replication Manager or the replication management API.

Internal Triggers

Oracle uses internal triggers to capture and store information about updates to replicated data. Internal triggers build RPCs to reproduce data changes made to the local site at remote replication sites. These deferred RPCs are stored in the deferred transaction queue and are propagated to the other master sites participating in the replication environment. 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.

Deferred Transactions

Oracle forwards data replication information by propagating (that is, sending and executing) the RPCs that are generated by the internal triggers described above. These RPCs are stored in the deferred transaction queue. In addition to containing the execution command for the internal procedure at the destination site, each RPC also contains the data to be replicated to the target site. Oracle uses distributed transaction protocols to protect global database integrity automatically and ensure data survivability.

Internal Procedure

When a deferred RPC created by an internal trigger is propagated to the other master sites participating in a replication environment, an internal procedure at the destination site is used to apply the deferred RPC at the remote site. These internal procedures are activated automatically when you generate replication support for a table. These internal procedures are executed based on the RPCs that are received from the deferred transaction queue of the originating site.

Queues

The following queues manage the transactions that are generated by Oracle replication:

Deferred Transaction Queue This queue stores the transactions (for example, DML) that are bound for another destination in the master group. Oracle stores RPCs produced by the internal triggers in the deferred transaction queue of a site for later propagation. Oracle also records information about initiating transactions so that all RPCs from a transaction can be propagated and applied remotely as a transaction. Oracle's replication facility implements the deferred transaction queue using Oracle's advanced queueing mechanism.

Error Queue The error queue stores information about deferred transactions that could not be applied successfully at the local site. The error queue does not display information about errors at other master sites in the replication environment. When the error condition has been resolved, the DBA can either re-execute the transaction or delete the transaction from the error queue.

Job Queue Oracle manages the propagation process using Oracle's *job queue mechanism* and *deferred transactions*. Each server 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 a 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.

Administrative Mechanisms

Several mechanisms are required to handle the administrative tasks that are often performed to support a replicated environment. These mechanisms allow you to turn on and off a replication environment, as well as monitor the administrative tasks that are generated when you build or modify a replicated environment.

Replication Modes of Operation

There are three modes of operation for a replication environment.

Normal A replicated environment in the normal mode allows replication to occur. Any transaction against a replicated object is allowed and is appropriately propagated.

Quiescing Quiescing is the mode that transfers a replicated environment from the normal mode to the quiesced mode. While a replication environment is quiescing, the user is no longer able to execute a transaction against a replicated object, but any existing deferred transactions are propagated. Queries against a quiescing table are allowed. When all deferred transactions have been successfully propagated to their respective destinations, the replication environment proceeds to the quiesced mode.

Quiesced A quiesced replication environment can be considered disabled for normal replication use and is used primarily for administrative purposes (that is, adding, removing, or altering replicated objects). A quiesced state prevents users from executing any transactions against a replicated object in the quiesced master group unless they turn off replication, which can result in divergent data once replication is resumed. Transactions include DML against a replicated table or the execution of a wrapper for a replicated procedure. If master tables are quiesced, snapshots based on those master tables cannot propagate their changes to the target master tables, but local changes to the snapshot can continue.

A replication environment is quiesced on a master group level. However all master sites participating in the replicated master group are affected. When a master group reaches a quiesced state, you can be certain that any transactions in the deferred transaction queue have been successfully propagated to the other master sites or put into the error queue. Quiescing one master group does not affect other master groups. A master group in normal mode can continue to process updates while other master groups are quiesced.

Replication Mode Control

Though there are three modes of replication operation, there are only two mechanisms to control these modes (recall that the quiescing mode is a transition from a normal to quiesced mode).

Suspend Executing the suspend mechanism begins the quiescing mode that transfers the mode of replication operation for a master group from normal to quiesced. When the deferred transaction queue has no unpropagated deferred transactions for the master group, the replication environment proceeds to the quiesced mode.

The suspend mechanism can only be executed when the replication environment is in normal mode. Execute suspend when you need to modify the replication environment.

Resume The resume mechanism transfers a master group from the quiesced replication mode to the normal mode. If you have been performing administrative work on your replication environment (for example, adding replicated objects), you should verify that the administrative queue (DBA_REPCATLOG) is empty before executing the resume mechanism.

Administration Requests

To configure and manage a replication environment, each participating server uses Oracle's replication management 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.

The Administration Request Mechanisms When you use Replication Manager or make a call to a procedure in the DBMS_REPCAT package to administer a replication system, Oracle uses its internal mechanisms to broadcast the request using synchronously. 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 DBA_REPCATLOG view and the corresponding DDL statement in a child table of the DBA_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 DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN procedure. 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 is 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 may experience a delay if you do not have enough background processes available to execute the outstanding jobs.

Note: When JOB_QUEUE_PROCESSES is set to zero at a site, you must apply administration requests manually for all groups at the site.

See Also: "Initialization Parameters" on page 7-3, and the *Oracle8i Reference* for more information about JOB_QUEUE_INTERVAL and JOB_QUEUE_PROCESSES.

For each call of DO_DEFERRED_REPCAT_ADMIN at a master site, the site checks the DBA_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 DBA_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 DBA_REPCATLOG view at the master definition site. If a request completes successfully at all sites, all entries in the DBA_REPCATLOG view at all sites, including the master definition site, are removed. If a request at a non master definition site fails, Oracle removes the

request at the master site and updates the corresponding AWAIT_CALLBACK request at the master definition site with the reason for the failure.

By synchronously broadcasting the change, Oracle ensures that all sites are aware of the change, and thus are capable of remaining synchronized. 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 DBA_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 recreate the snapshot.

Administrative Queue

Often referred to as the administrative queue, the DBA_REPCATLOG view stores administration requests that manage and modify your replication environment. Some DBMS_REPCAT procedures that are executed are listed in the administrative queue; for example, if you wanted to add an additional replicated table to an existing master group, you would see a request naming the DBMS_REPCAT.CREATE_MASTER_REPOBJECT procedure.

You can view the administrative queue by querying the DBA_REPCATLOG view with a tool such as SQL*Plus or view the Deferred Transactions by Destination node in Replication Manager.

Each request has a status that displays the state of the request. There are four states:

- **Ready**: The ready state declares that the request is ready to be executed. If you monitor the administrative queue and a request remains in the ready state for a long time, there may be a request in front of the ready request that is awaiting callback.
- **Awaiting Callback**: The awaiting callback state declares that the request is waiting for a request to be executed at another site and is awaiting confirmation of the request execution. Once the request receives the callback, the request is either deleted or has its status changed.
- **Error**: If a request cannot be successfully executed, it is placed in the error state. The error number appears in the Message column of the administrative queue (it is in the Error column when using Replication Manager).

  Note: If a request is in the error state, resolve the error condition as described by the error number and re-submit the request.

- **Do Callback**: If a request at a master site is in do callback state, it means that the master site must inform the master definition site about the success or failure of the request. If a request at the master definition site has a do callback status, it means that the master definition site must notify a master site to proceed to the next phase of the request when all other master sites have completed the current phase.

The administrative queue of each master site lists only the administration requests to be performed at that master site. The master definition site for a master group, however, lists administration requests to be performed at each of the master sites. The administrative queue at the master definition site lets the DBA monitor administration requests of all the master sites in the replication environment.

Organizational Mechanisms

Oracle uses several organizational mechanisms to organize the previously described master site and administrative mechanisms to create discrete replication groups. Most notable of these organizational mechanisms is the master group. An additional organization mechanism helps to group columns that are used to resolve conflicts for a replicated table.

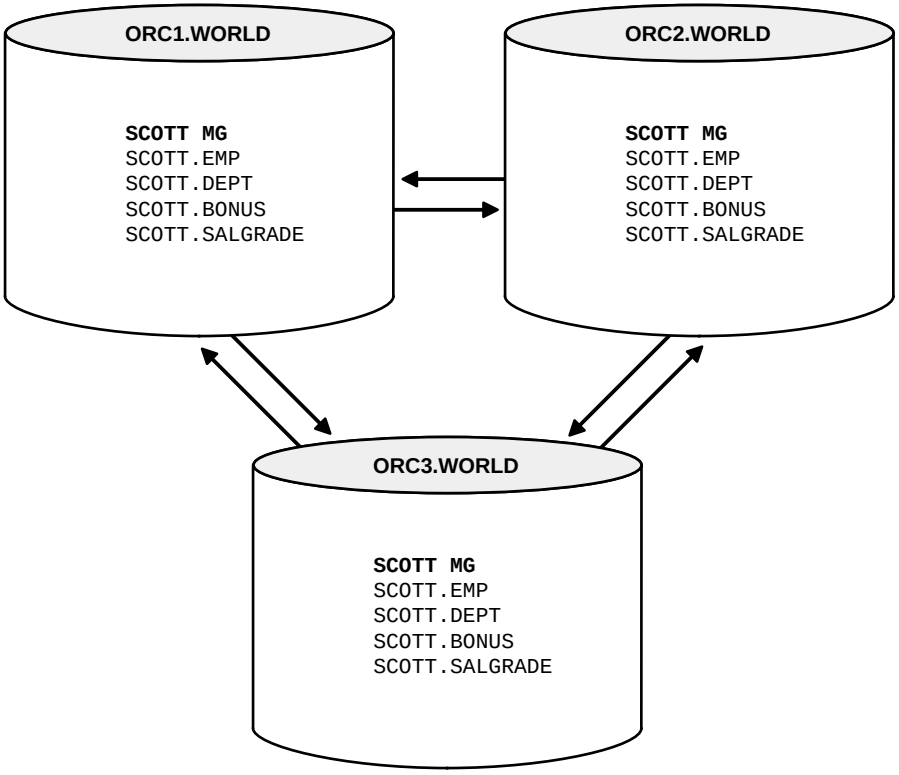
Master Group

In a replication environment, Oracle manages replication objects using *replication groups*. A replication group is a collection of replication objects that are always updated in a transactional consistent manner.

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.

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 2-15). Figure 2–4 illustrates that master group "SCOTT_MG" contains an exact replica of the replicated objects at each master site.

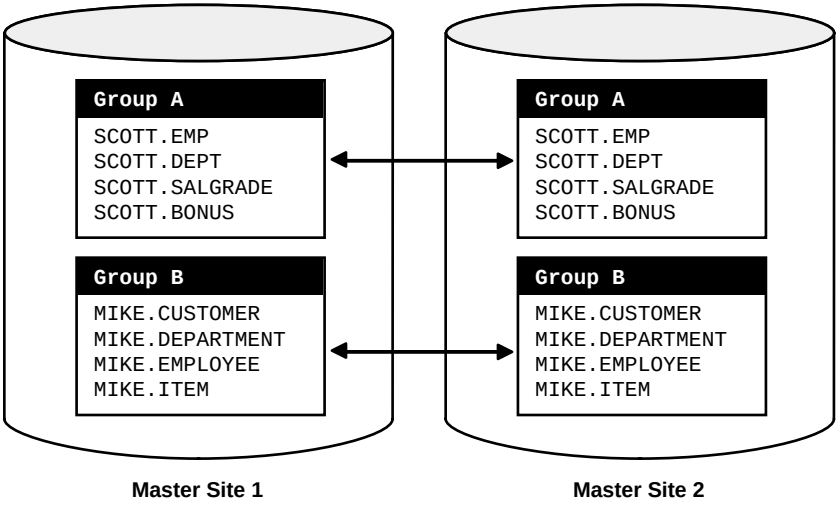
Figure 2–4 Master Group SCOTT_MG contains same replication objects at all sites.



The master group organization at the master site plays an integral role in the organization of replication objects at a snapshot site. See the "Organizational Mechanisms" section on page 3-23 for more information on the organizational mechanisms at a snapshot site.

Additionally, Figure 2–5 illustrates that each site may contain multiple replication groups, though each group must contain exactly the same set of objects at each master site.

Figure 2–5 Master Groups are Identical at Each Master Site



Column Groups

Column groups provide the organization mechanism to group all columns that are involved in a conflict resolution routine. If a conflict occurs in one of the columns of the group, the remainder of the group's columns may be used to resolve the conflict. For example, if a column group for a table contains a MIN_PRICE, LIST_PRICE, COST_PRICE, and TIMESTAMP field and a conflict arises for the list_price field, the timestamp field may be used to resolve the conflict, assuming that a timestamp conflict resolution routine has been used.

Initially, you might think that you should put all columns in the table into a single column group. Though this makes setup and administration easier, it may decrease the performance of your replicated table and may increase the potential for data conflicts. As you will learn in the "Performance Mechanisms" section, the Min Communications feature does not send extraneous data from other column groups when a conflict occurs. Placing all columns into a single column group may negate the advantages of the Min Communications feature, unless you use the DBMS_REPCAT.SEND_OLD_VALUES and DBMS_REPCAT.COMPARE_OLD_VALUES procedures.

> **See Also:** Chapter 5, "Conflict Resolution Concepts & Architecture" for more information about column groups.

Propagation Process

The propagation process is the essence of replication because it is the mechanism that sends or distributes any actions to all other master sites in the replication environment.

Propagation Types

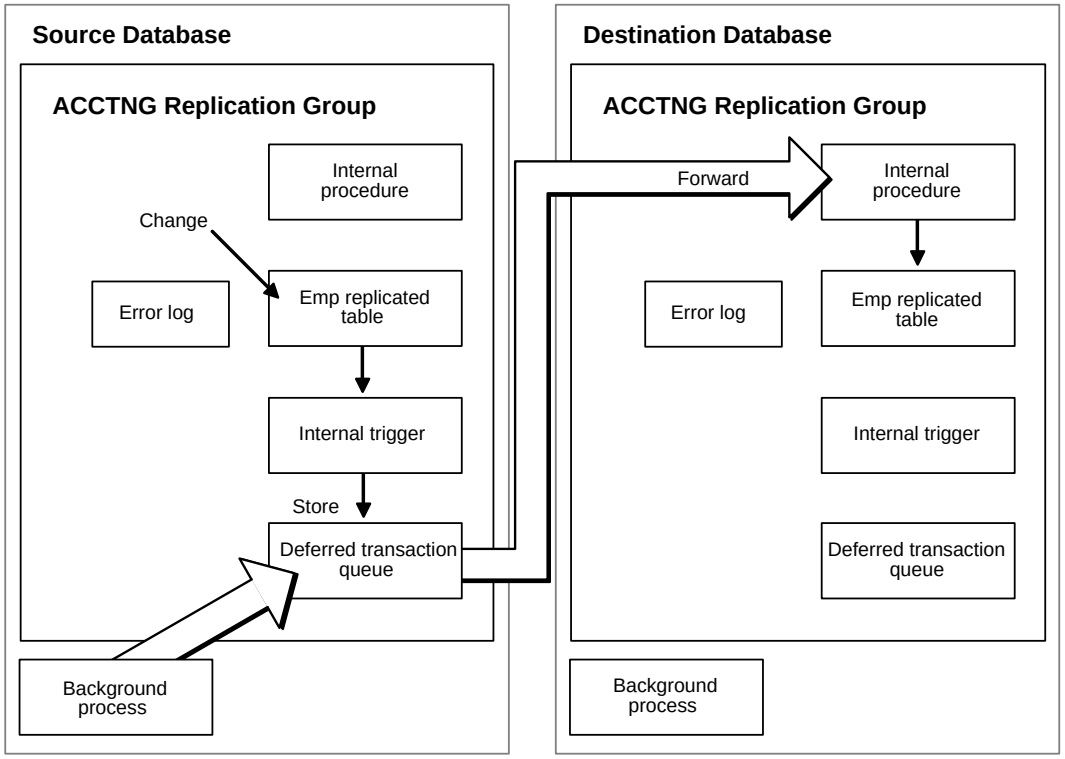
As the internal trigger captures any DML applied to a replicated table, the DML must be *propagated* or sent to the other master sites in the replication environment. Internal triggers are described in the section "Internal Triggers" on page 2-19.

Oracle replication supports both asynchronous and synchronous replication.

Asynchronous Typical replication configurations use *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*.

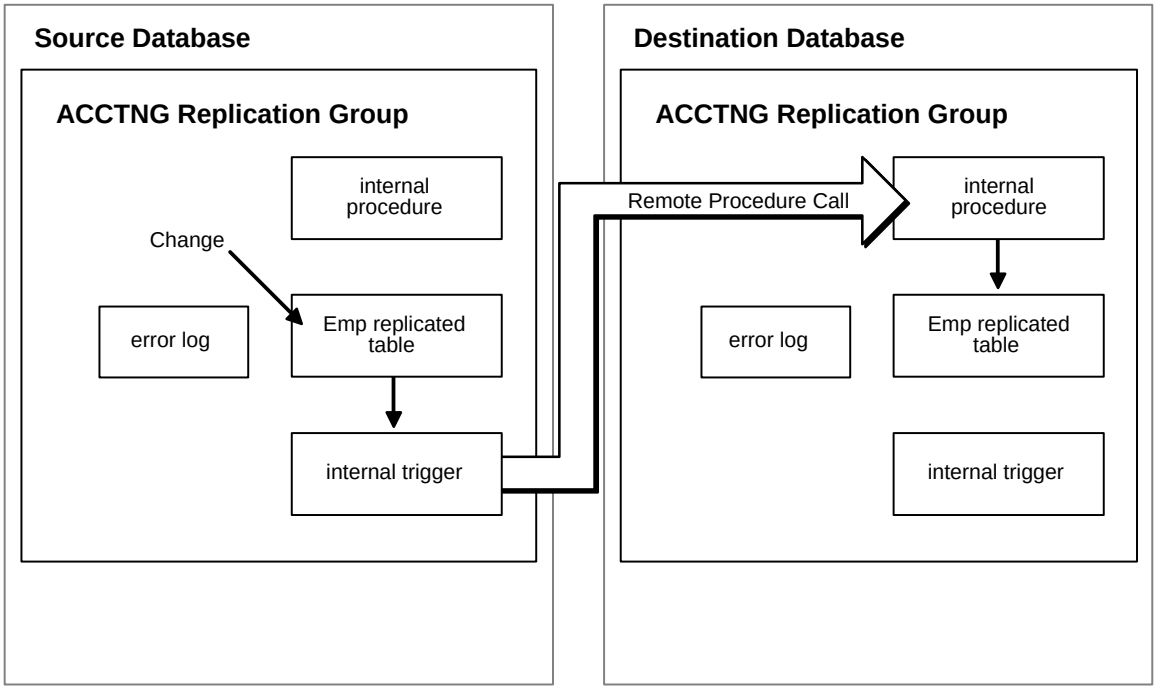
As Figure 2–6 shows, Oracle uses its internal triggers, deferred transactions, deferred transaction queues, and job queues to propagate data-level changes asynchronously among master sites in a replication system, as well as from an updateable snapshot to its master table.

Figure 2–6 Asynchronous Data Replication Mechanisms



Synchronous 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 at least one other replica 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 2–7 Synchronous Data Replication Mechanisms

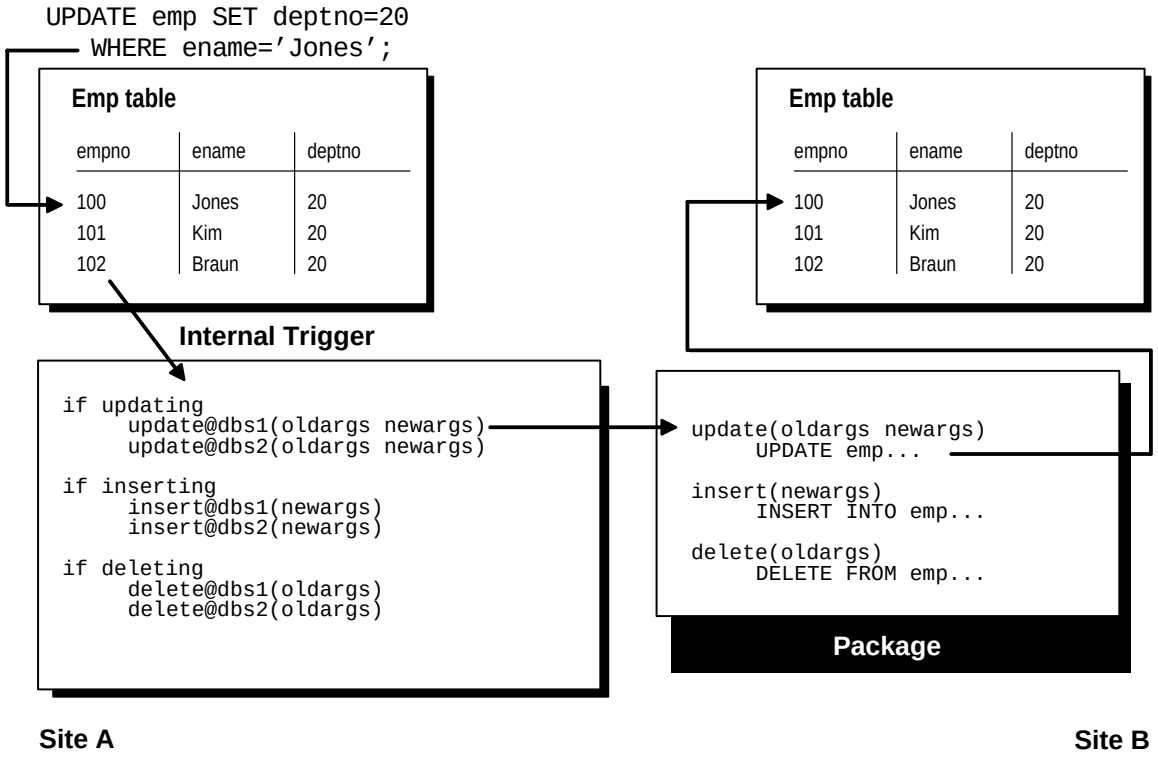


As Figure 2–7 shows, Oracle uses the same internal triggers to generate RPCs that asynchronously replicate data-level changes to other replication sites to support synchronous, row-level data replication. However, Oracle does not defer the execution of such RPCs. Instead, data replication RPCs execute within the boundary of the same transaction that modifies the local replica. Consequently, a data-level change must be possible at all synchronously linked sites that manage a replicated table; otherwise, a transaction rollback occurs.

Understanding Synchronous Data Propagation

As shown in Figure 2–8, 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 2–8 Propagating Changes using Synchronous Row-Level Replication



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 is 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.

> **See Also:** Chapter 5, "Conflict Resolution Concepts & Architecture".

Understanding Mixed-Mode Multimaster Systems

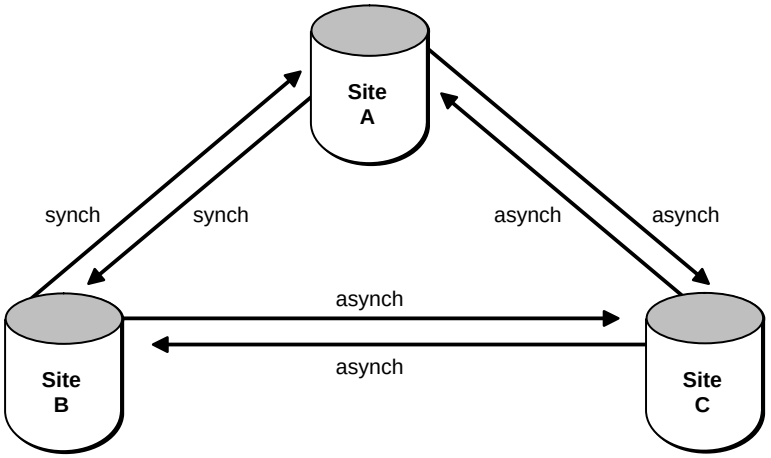
In some situations, you might decide to have a mixed-mode environment in which some master sites propagate a master group's changes asynchronously and others propagate changes synchronously. The order in which you add new master sites to a group with different data propagation modes can be important.

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 sends changes to site B synchronously and site B sends 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 2–9.

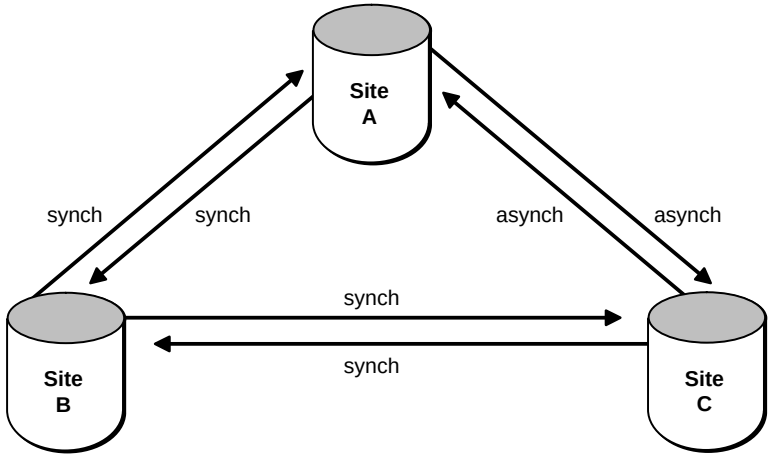
Figure 2–9 Selecting a Propagation Mode



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 2–10.

Figure 2–10 Ordering Considerations



Each time that you add a new master site to a mixed-mode multimaster system, consider how the addition affects the data propagation modes to and from existing sites.

Initiating Propagation

When synchronous propagation is used, the propagation of the DML is handled immediately and is automatically initiated. If asynchronous propagation is used, there are two methods to propagate the deferred transactions. In most cases, use a job to automatically propagate the deferred transactions according to a scheduled interval of your choosing. You can also manually propagate the changes by executing a stored procedure or using Replication Manager. You may occasionally need to manually propagate your deferred transactions if you do not want to wait for the job queue to automatically propagate the deferred transactions.

Performance Mechanisms

As with any enterprise database solution, performance is always an important issue for the database administrator. Oracle replication provides several mechanisms to help increase the performance of your replicated environment.

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 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 transaction dependencies, allocates work to the server processes, and tracks their progress.

Parallel processes remain associated with a parallel operation on the server 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 push until it is complete.

To configure a pool of parallel server processes for a server properly, you must consider several issues related to the configuration of a 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 need 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 uses to push changes to its destination. Furthermore, you should also consider how long each push holds 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

See Also:

- "Initialization Parameters" on page 7-3
- *Oracle8i Reference*
- *Oracle8i Concepts*

Implementing Parallel Propagation For most users, setting the parallel propagation parameter to a value of 1 satisfies their replication performance needs. However, some users may want to further tune the parallel propagation value.

The following procedure is the recommended method that should be used to further tune the parallel propagation value:

1. Set the parallel propagation value to 1.
2. Test your database environment and carefully measure the propagation throughput.

   If you have achieved your performance goals with a parallel propagation value of 1, skip to Step 5.

   Note: As you increase the value of the parallel propagation parameter, be aware of the trade-offs between increased parallel propagation and the resources required to support the extra parallel slaves.

   ---

3. If you want to try and achieve greater propagation throughput than with a value of 1, set your parallel propagation value to 2.
4. Test your database environment and carefully measure the propagation throughput.

   In many cases, you will experience propagation throughput degradation with a value of 2. This reduction is due to round-trip delays associated with the coordinator assigning dependent transactions to available slaves and waiting for the necessary commit acknowledgments before assigning additional transactions.

Repeat Steps 3 and 4 with the parallel propagation value set to 4 and again with 8. If throughput still does not improve, it suggests that the transactions in your environment are highly dependent on each other. Reduce the parallel propagation value to 1 and proceed to Step 5.

> **See Also:** "Tuning Parallel Propagation" on page 2-38 to learn about techniques to reduce transaction dependencies.

If your performance did improve with a value of 2, 4, or 8, it suggests that your transactions have a low degree of interdependence. You may even set your parallel propagation parameter to any value greater than 8. Just be sure to thoroughly test your environment and remain aware of the trade-offs between increased parallelism and the necessary resources to support those extra parallel slaves.

5. You have completed adjusting the parallel propagation value to best meet the performance needs of your database environment.

Tuning Parallel Propagation To gain the greatest amount of performance benefits from parallel propagation, you should try to reduce the amount of dependent transactions that are created. Remember that a dependent transaction cannot start until all of its dependent transactions have been committed.

Keep the following tips in mind when trying to reduce the number of dependent transactions:

- Use smaller transactions if possible (that is, commit more often, without destroying autonomy).
- Increase number of freelists for each table that receives inserts.
- Try to avoid hotspots (a row that is frequently modified - if the same row is touched, then those transactions are serialized). For example, use an Oracle sequence instead of using a counter in a row and incrementing it "manually."

Min Communication

To increase the replication performance for tables, be sure to enable the Min Communication setting when generating replication support for a replicated table.

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 vary.

For example, when you create a replicated table and all participating sites are Oracle release 8.0 or greater databases, you can choose to minimize the amount of data that must be communicated to detect conflicts for each changed row in the table. In this case, Oracle propagates:

- The old value of the primary key and the old value of each column in each modified column group (the value before the modification).
- The new value of each updated column.

> **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 Oracle release 8.0 or greater replication environments to reduce the amount of data that Oracle transmits across the network. As a result, you can help to improve overall system performance.

Alternatively, when a replicated environment uses both Oracle7 and release 8.0 or greater sites, you cannot minimize the communication of row data for update conflict detection. In this case, Oracle must propagate the entire old and new versions of each changed row to perform conflict detection.

> **See Also:** "Performance Mechanisms" on page 5-26 for more information about conflict resolution, min communication, and additional conflict resolution performance techniques.

Delay Seconds

Though not directly a performance mechanism, properly configuring the DELAY_SECONDS parameter can give you greater control over the timing of your propagation of deferred transactions.

DELAY_SECONDS controls how long a job remains aware of the deferred transaction queue. The effects of the DELAY_SECONDS parameter can best be illustrated with the following two examples:

DELAY_SECONDS = 0 (default)
If a scheduled job with a 60 minute interval wakes up at 2:30 pm and checks the deferred transaction queue, any existing deferred transactions are propagated. The propagation takes 2 minutes and therefore the job is complete at 2:32 pm.

If a deferred transaction enters the queue at 2:34 pm, the deferred transaction is not propagated because the job is complete. In this scenario, the deferred transaction will be propagated at 3:30 pm.

DELAY_SECONDS = 300
If a scheduled job with a 60 minute interval wakes up at 2:30 pm and checks the deferred transaction queue, any existing deferred transactions are propagated. The propagation takes 2 minutes and therefore the job is complete at 2:32 pm.

If a deferred transaction enters the queue at 2:34 pm, the deferred transaction is propagated because the job remains aware of the deferred transaction queue for 300 seconds (5 minutes) after the job has completed propagating whatever was in the queue. In this scenario, the deferred transaction is propagated at 2:34 pm.

One question that you might ask is "why don't I just set the job to execute more often?" Starting and stopping the job has a greater amount of overhead than starting the job and keeping it aware for a set period of time. In addition to decreasing the overhead associated with starting and stopping these jobs, using the DELAY_SECONDS parameter can reduce the amount of redo logging required to support scheduled jobs.

As with most performance features, there is a point of diminishing returns. There are several reasons why you want to keep the length of the DELAY_SECONDS parameter in check:

- **Parallel Propagation**: Each parallel process that is used when pushing the deferred transaction queue is not available for other parallel activities until the propagation job is complete; a long DELAY_SECONDS value may keep the parallel process unavailable for other operations.
- **Serial Propagation**: If you are only using serial propagation (not parallel propagation), the DELAY_SECONDS value causes the open session to "sleep" for the entire length of the delay, providing none of the benefits earlier described.
- **Precise Purge**: If you specify the precise_purge method when using the DBMS_DEFER_SYS.PURGE procedure and you have defined a large DELAY_SECONDS value, you may experience performance degradation when performing the specified purge.

As a general rule of thumb, there are few viewable benefits of setting the DELAY_SECONDS parameter to a value greater than 20 minutes (DELAY_SECONDS=1200).

Additionally, if you are only using serial propagation (parallel propagation=0), you need to consider the benefits of large DELAY_SECONDS values. Unlike parallel propagation, serial propagation only checks the queue after the duration of the delay seconds value has elapsed. If you set a value of 20 minutes for parallel propagation, the parallel push checks once a minute. If you can sacrifice the lock on the resources for 20 minutes (the SNP process is not available for other jobs), then you can set the delay seconds value to 1200.

If you cannot afford this resource lock, set the DELAY_SECONDS value to 10 or 20 seconds; though you will need to execute the jobs more often than if the value was set to 1200 seconds, you still gain many of the benefits of the delay seconds feature (versus a value of 0 seconds).

Conflict Resolution Mechanisms

The receiving master site in a replication environment 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, which are the values before the modification, and the current values of the same row at the receiving site in either the primary key columns or the columns in an updated column group.
- 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 "Minimum Communication" on page 5-26.

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 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.

Caution: 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.

Resolve Data Conflicts

Oracle provides a mechanism that allows you to define a conflict resolution method that resolves a data conflict when detected. Oracle provides several built in conflict resolution methods:

- Latest and Earliest Timestamp
- Overwrite and Discard
- Maximum and Minimum
- Additive and Average
- Timestamp
- Priority Group
- Site Priority

If the built-in Oracle conflict resolution methods do not meet the needs of your replicated environment, you have the option of writing your own conflict resolution method using PL/SQL and implementing it as a user-defined conflict resolution method. See Chapter 5, "Conflict Resolution Concepts & Architecture" to learn how conflict resolution works.

See Also: The online help for Replication Manager to learn how to implement conflict resolution with Replication Manager, and see Chapter 6, "Conflict Resolution" in the *Oracle8i Replication Management API Reference* to learn how to implement conflict resolution using the replication management API

3

Snapshot Concepts & Architecture

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

Snapshot Concepts

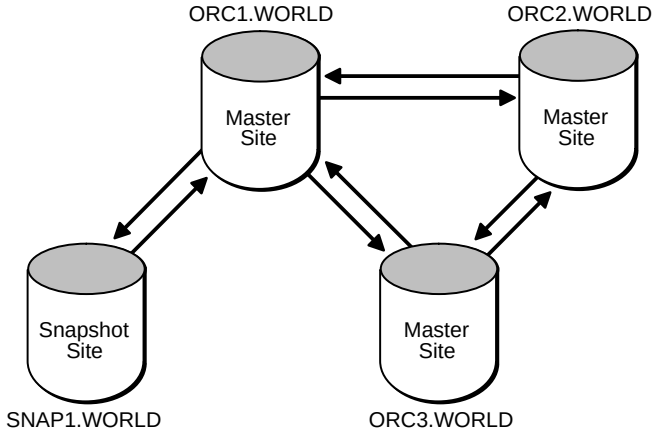
Oracle uses *snapshots*, also referred to as *materialized views*, to replicate data to non-master sites in a replicated environment and to cache "expensive" queries in a data warehouse environment. This chapter, and this *Oracle8i Replication* manual in general, discusses snapshots for use in a replicated environment.

> **See Also:** *Oracle8i Data Warehousing Guide* to learn more about materialized views for data warehousing.

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 from one or more master tables through individual batch updates, known as a *refreshes*, from a single master site, as illustrated in Figure 3–1.

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



Snapshots can also contain a WHERE clause so that snapshot sites can contain customized data sets. Such snapshots can be helpful for regional offices or sales forces that do not require the complete corporate data set.

When a snapshot is refreshed, Oracle must examine all of the changes to the master table to see if any apply to the snapshot. Therefore, if any changes where made to the master table since the last refresh, a snapshot refresh will take some time, even if

the refresh does not apply any changes to the snapshot. If, however, no changes at all were made to the master table since the last refresh of a snapshot, the snapshot refresh should be very quick.

Why Use Snapshots?

You might use a snapshot to achieve one or more of the following goals:

- 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. Also, a snapshot can be a subset of a master table, which decreases the amount of data that is replicated.

While multimaster replication also distributes a corporate database to multiple sites, the networking requirements are greater than those for replicating with snapshots because of the transaction by transaction nature of multimaster replication. Also, the ability of multimaster replication to provide real-time or near real-time results causes greater network traffic, and might require a dedicated network link.

Snapshots are updated through an efficient batch process from a single master site. They have lower network requirements and dependencies than multimaster replication because of the point in time nature of snapshot replication. Whereas multimaster replication requires constant communication over the network, snapshot replication requires only periodic refreshes. 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), greatly enhance the performance and reliability of your replicated database.

Mass Deployment

Deployment templates allow you to precreate a snapshot environment locally. You can then use deployment templates to quickly and easily deploy snapshot environments to support sales force automation and other mass deployment 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, while multimaster replication requires replication of the entire table. Data subsetting allows you to replicate information that pertains only 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 replication management API for refresh on-demand into the sales application. When the salesperson has completed the day's order, the salesperson simply dials-up the network and uses the integrated mechanism to refresh the database, thus transferring the orders to the main office.

Available Snapshots

Oracle offers several types of snapshots to meet the needs of many different replication (and non-replication) situations. The following sections describe each type of snapshot and also describe some environments for which they are best suited.

The following sections contain examples of creating different types of snapshots. Whenever you create a snapshot, regardless of its type, always specify the schema name of the table owner in the query for the snapshot. For example, consider the following CREATE SNAPSHOT statement:

```
CREATE SNAPSHOT emp_snap AS SELECT * FROM scott.emp@db1.world;
```

Here, the schema SCOTT is specified in the query.

Primary Key

Primary key snapshots are the default type of snapshot. They are updateable if the snapshot was created as part of a snapshot group and “FOR UPDATE” was specified when defining the snapshot. Changes are propagated according to the row-level changes that have occurred, as identified by the primary key value of the row (not the ROWID). The SQL statement for creating an updateable, primary key snapshot might look like:

```
CREATE SNAPSHOT sales.customer FOR UPDATE AS
 SELECT * FROM sales.customer@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. 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.customer@dbs1.acme.com c
    WHERE o.c_id = c.c_id AND zip = 19555);
```

> **See Also:** "Snapshot Groups" on page 3-23 for more information about snapshot groups, and see "Snapshots with Subqueries" on page 3-9 for more information about snapshots with subqueries.

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 Oracle release 8.0 or greater databases.

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

> **See Also:** "Snapshot Log" on page 3-20 for more information on the differences between a ROWID and Primary Key snapshot.

Complex

To be fast refreshed, the defining query for a snapshot must observe certain restrictions. If you require a snapshot whose defining query is more general and cannot observe the restrictions, then the snapshot is *complex* and cannot be fast refreshed.

Specifically, a snapshot is considered complex when the defining query of the snapshot contains:

- A CONNECT BY clause
- Clauses that do not comply with the requirements detailed in Table 3–1, "Restrictions for Snapshots with Subqueries"
- A set operation, such as UNION, INTERSECT, or MINUS
- In most cases, a distinct or aggregate function, although it is possible to have a distinct or aggregate function in the defining query and still have a simple snapshot

> **See Also:** *Oracle8i Data Warehousing Guide* for more information about complex materialized views. "Snapshot" is synonymous with "materialized view" in Oracle documentation, and "materialized view" is used in the *Oracle8i Data Warehousing Guide*.

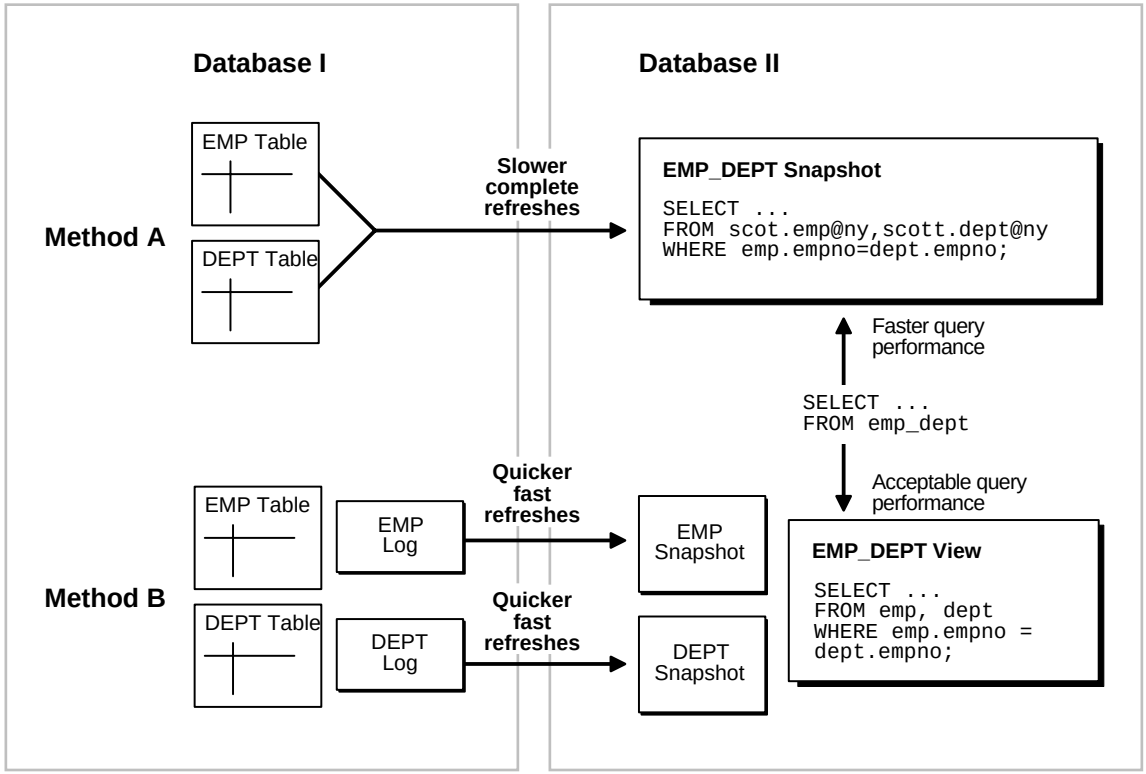
Note: If possible, you should avoid using complex snapshots because they cannot be fast refreshed, which may degrade network performance (see "Refresh Process" on page 3-28 for information).

The following statement is an example of a complex snapshot CREATE statement:

```
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 using 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 was completed during the snapshot's refresh. However, complete refreshes must be performed because the snapshot is complex, and these refreshes will probably be slower than fast refreshes.
- **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 refreshed more efficiently using fast refresh and 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 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 Also:** "Snapshot Groups" on page 3-23 for more information.

In addition, using read-only snapshots eliminates the possibility of a snapshot introducing data conflicts at the master site, although this convenience means that updates cannot be made at the remote snapshot site. You might define a read-only snapshot as:

```
CREATE SNAPSHOT sales.customer AS
  SELECT * FROM sales.customer@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 may want your snapshot to reflect a horizontal or vertical subset of the data in the master table. 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 4-22. Some reasons to consider partitioning data are:

- **Reduce Network Traffic**: Only changes that satisfy the snapshot's WHERE clause of the defining query are propagated to the snapshot site, thereby reducing the amount of data transferred and reducing network traffic.
- **Keep Sensitive Data Secure**: Users can only view data that satisfies the defining query for the snapshot.

- **Reduce Resource Requirements**: If the snapshot is located on a laptop, hard disks are generally significantly smaller than the hard disks on a corporate server. Partitioned snapshots may require significantly less storage space.
- **Improve Refresh Times**: Because less data is propagated to the snapshot site, the refresh process is 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 statement creates a snapshot that contains information about customers who are in the 19555 zip code:

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

Snapshots with Subqueries

The above example works well for individual snapshots that do not 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 called CUSTOMER, ORDERS, 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.

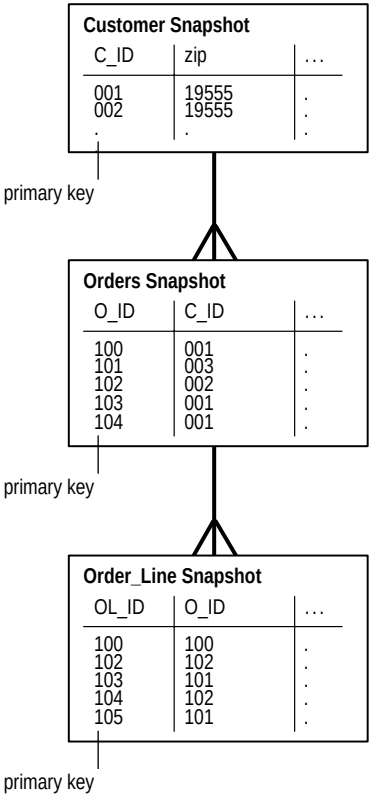
The CUSTOMER snapshot has a very simple defining query because the CUSTOMER master table is at the top of the hierarchy:

```
CREATE SNAPSHOT sales.customer AS
  SELECT * FROM sales.customer@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. Look at the relationships in Figure 3–3 below, and note that the CUSTOMER and ORDERS table are related through the C_ID column. The following statement creates 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.customer@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 ORDERS tables are related through the O_ID row. The following statement creates the ORDER_LINE snapshot with the appropriate data set:

```
CREATE SNAPSHOT sales.order_line 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.customer@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 also will 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 are populated with data that satisfies the defining query for each of these snapshots, and are refreshed only with data that satisfies these defining queries.

Using Assignment Tables

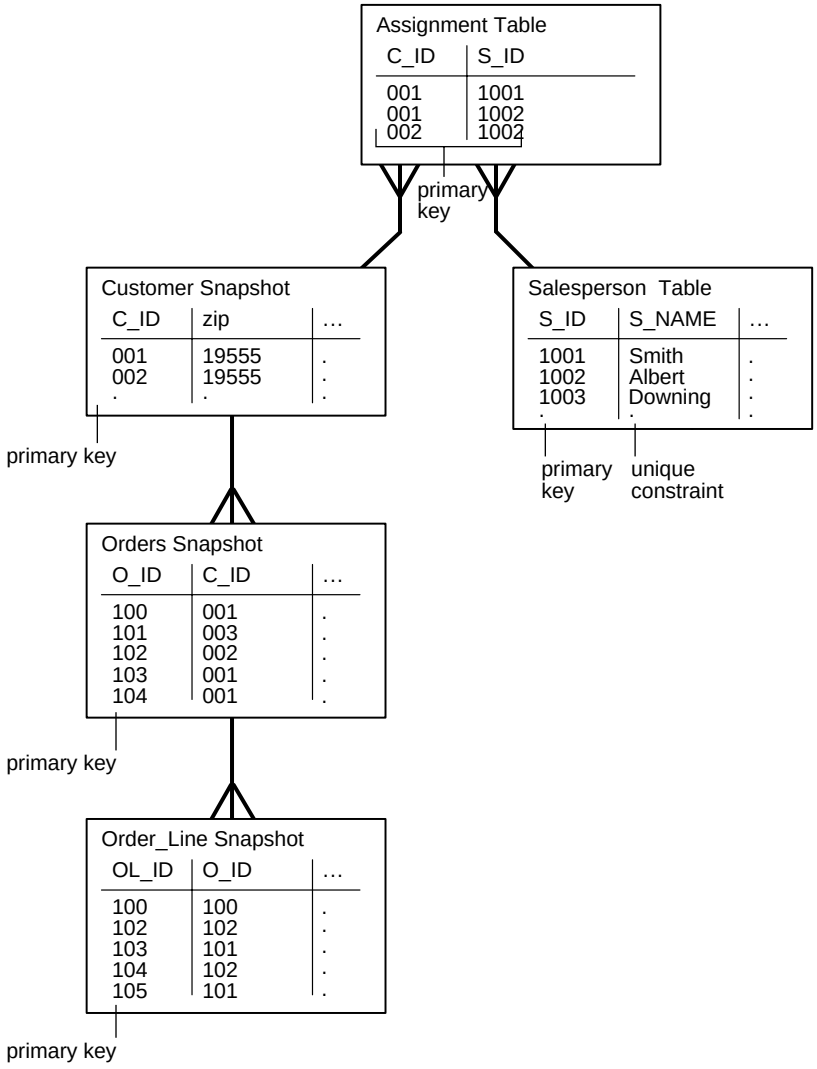
While the previous subquery examples demonstrate greater flexibility for snapshots, there are still certain limitations in the above example. For example, if the salesperson changed territories or the existing territory was assigned an additional zip code, the above snapshot definitions would need to be altered or recreated because 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 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, salesperson #1001 cannot view the

customer information of other salespersons, which is 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 the following:

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

Then, the ORDERS snapshot is populated with order data for the customers that are assigned to salesperson 'gsmith'. Notice the 'gsmith' value in the last line of the CREATE SNAPSHOT statement.

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. 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.

See Also: "Refresh Types" on page 3-28 for more information.

Restrictions for Snapshots with Subqueries

Snapshots with a subquery must be of the primary key type. Additionally, the defining query of a snapshot with a subquery is subject to several other restrictions to preserve the snapshot's fast refresh capability.

> **See Also:** "Primary Key" on page 3-5 for more information about primary key snapshots.

Note: To determine whether a snapshot's subquery satisfies the many restrictions detailed in Table 3–1, create the snapshot with fast refresh. Oracle returns errors if the snapshot violates any restrictions for simple subquery snapshots. If you specify force refresh, you may not receive any errors because, when a force refresh is requested, Oracle automatically performs a complete refresh if it cannot perform a fast 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 columns 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 exists on the join columns and a separate UNIQUE constraint exists on the filter columns of the table to which the intersection table joins. If there is only an intersection table, then there must be a PRIMARY KEY or UNIQUE constraint that includes both the join columns and filter columns 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" on page 3-11. 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 3-37.
- 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 Registration at a Master Site

At the master site, an Oracle database automatically registers information about a snapshots based on its master table(s). The following sections explain more about Oracle's snapshot registration mechanism.

Viewing Information about Registered Snapshots

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 query the DBA_SNAPSHOT_REFRESH_TIMES 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 snapshot at its master database when you create the snapshot, and unregisters the snapshot when you drop it.

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. Complex snapshots might not be registered.

Note: Oracle7 master sites cannot register snapshots.

Manual Snapshot Registration

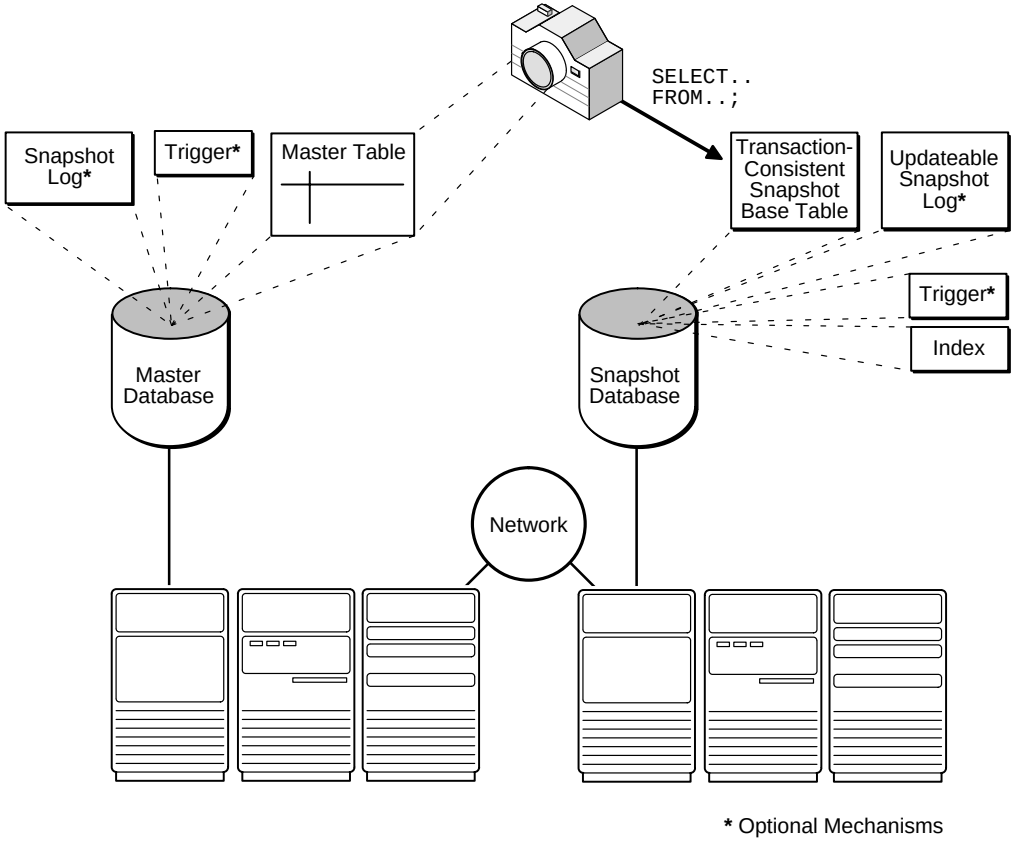
If necessary, you can maintain registration manually. Use the REGISTER_SNAPSHOT and UNREGISTER_SNAPSHOT procedures of the DBMS_SNAPSHOT package at the master site to add, modify, or remove snapshot registration information.

> **See Also:** The REGISTER_SNAPSHOT and UNREGISTER_SNAPSHOT procedures are described in the *Oracle8i Replication Management API Reference*.

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 do not have an updateable snapshot log or an 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.

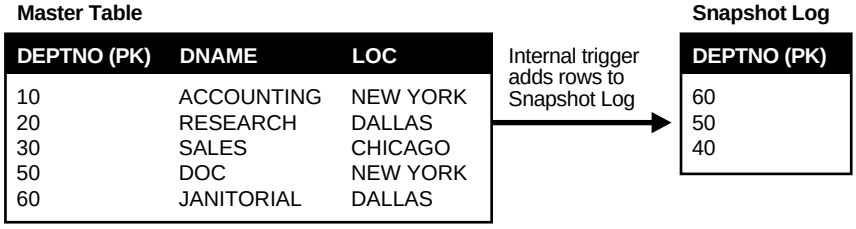
Figure 3–5 Snapshot Replication Mechanisms



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.

Note: Snapshots can be created based on tables only; creating tables based on a synonym or a view is not supported.

Internal Trigger

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

Note: Filter columns are required when the snapshot contains a subquery. See "Data Subsetting with Snapshots" on page 3-8 for information on subquery snapshots and "Using Filter Columns" on page 3-37 for more information.

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 / or 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. 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. One snapshot log can support multiple snapshots on its master table.

As described in the previous section, the internal trigger adds change information to the snapshot log whenever a DML transaction has taken place on 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 table based on both 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 that 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 small (one records affected rows using the primary key, while the other records affected rows using the physical ROWID), the practical impact is large. Using ROWID snapshots and snapshot logs makes reorganizing and truncating your master tables difficult because it prevents your ROWID snapshots from being fast refreshed. If you reorganize or truncate your master table, your ROWID snapshot must be COMPLETE refreshed because the ROWIDs of the master table have changed.

Importing Snapshots and Snapshot Logs Into a Different Schema Snapshots and snapshot logs are exported with the schema name explicitly given in the DDL statements. Therefore, snapshots and snapshot logs cannot be imported into a schema that is different than the schema from which they were exported. If you attempt to use the FROMUSER / TOUSER import options to import an export dump file that contains snapshots or snapshot logs, an error will be written to the Import log file and the items will not be imported.

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 possibly a view are created. If you create an updateable snapshot, an internal trigger and a local log (the 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 has the name that you specified during snapshot creation.

When the snapshot site compatibility setting is less than 8.1.0, the base table is the underlying support mechanism for a view. The compatibility setting is defined by the COMPATIBLE initialization parameter in the initialization parameter file. 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.

Note: When SNAP$_*Snapshot_Name* exceeds the 32 character limit, the table name is truncated and a sequence number is appended.

See Also: "Initialization Parameters" on page 7-3 and the *Oracle8i Migration* for more information about COMPATIBLE, and see "View" on page 3-22 for more information about the view that is created in support of snapshots with a compatibility level prior to 8.1.0.

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.

Note: An updateable snapshot based on a master table that has defined column default values does not automatically use the master table's default values.

Note: Updateable snapshots do not support the DELETE CASCADE constraint.

View

A view is created only to support snapshot replication with Oracle release 8.0 and earlier, or if a release 8.1 snapshot site's compatibility setting is less than 8.1.0. If a view is created, the view has the same name specified in the CREATE SNAPSHOT statement. For example, a CREATE SNAPSHOT SALES.SNAP_CUSTOMER AS .... statement creates a view named SNAP_CUSTOMER.

Index

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

Note: When I_SNAP$_*Snapshot_Name* exceeds the 32 character limit, the table name is truncated and a sequence number is appended.

Updateable Snapshot Log

An updateable snapshot log (USLOG$_*Snapshot_Name*) is used to determine data that must be pulled from the target master table. A read-only snapshot does not require this 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.

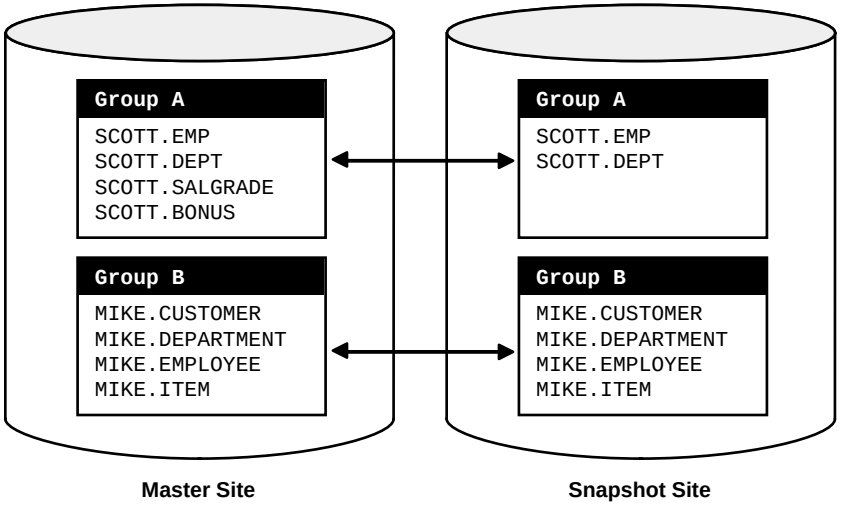
Organizational Mechanisms

In addition to the snapshot mechanisms described in the previous section, there are several other 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 a replication system maintains a partial or complete copy of the objects at the target master group. Snapshot groups cannot span 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 (see 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, a snapshot group has the same name as the master group on which the snapshot group is based. For example, a snapshot group based on a "PERSONNEL" master group is also 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 must 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. Doing so enables you to define different subqueries for your snapshot definitions in each snapshot group, and allows each user to access only his or her subset of the data.

Defining multiple data sets with different snapshot groups is more secure than defining different WHERE clauses for multiple views supporting different users. When you define multiple data sets with different snapshot groups, you can grant users access to individual snapshot objects, and 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 what a user the deletes or inserts.

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, instead of as individual objects. For example, you can refresh the snapshots as a departmental group, and you can drop the objects as a 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 has 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.

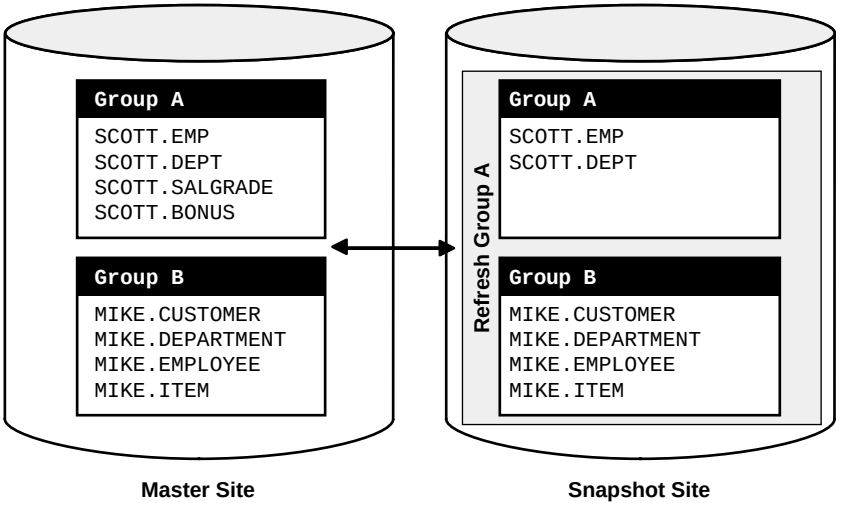
> **See Also:** The "Using a Group Owner" section in Chapter 7 of the *Oracle8i Replication Management API Reference* manual for more information on defining a group owner using the replication management API.

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 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, which is an increase 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 be used only when you have in-depth knowledge of the database environment and can prevent any referential integrity problems.

Refresh Group Size

There are a few trade-offs to consider when you are deciding on the size of your refresh groups. Oracle is optimized for large refresh groups. So, large refresh groups refresh faster than an equal number of snapshots in small refresh groups, assuming that the snapshots in the groups are similar. For example, refreshing a refresh group with 100 snapshots is faster than refreshing five refresh groups with 20 snapshots each. Also, large refresh groups enable you to refresh a greater number of snapshots with only one call to the replication management API.

During the refresh of a refresh group, each snapshot in the group is locked at the snapshot site for the amount of time required to refresh all of the snapshots in the refresh group. This locking is required to prevent users from updating the snapshots during the refresh operation, as updates might make the data inconsistent. Therefore, having smaller refresh groups means that the snapshots are locked for less time when you perform a refresh.

Network connectivity must be maintained while performing a refresh. If the connectivity is lost or interrupted during the refresh, all changes are rolled back so that the database remains consistent. Therefore, in cases where the network connectivity is difficult to maintain, consider using smaller refresh groups.

Release 8.1 supports null refresh optimization. That is, if there were no changes to the master tables since the last refresh for a particular snapshot, almost no extra time is required for the snapshot during snapshot group refresh. However, for snapshots in a database prior to release 8.1, consider separating snapshots of master tables that are not updated often into a separate refresh group of their own. Doing so shortens the refresh time required for other snapshot groups that contain snapshots of master tables that are updated frequently.

Table 3–2 summarizes the advantages of large and small refresh groups.

Table 3–2 Large and Small Refresh Groups

Advantages of Large Refresh Groups	Advantages of Small Refresh Groups
■ Refreshes faster than an equal number of snapshots in multiple refresh groups	■ Snapshots locked for shorter periods of time
■ Refreshes with single replication management API call	■ Rollback of refresh changes due to loss of connectivity is less likely

Refresh Process

A snapshot's data does not necessarily match the current data of its master table at all times. A snapshot is a transactionally (read) consistent reflection of its master table as the data existed at a specific point in time (that is, at creation or when a refresh occurs). To keep a snapshot's data relatively current with the data of its master table, the snapshot must be periodically refreshed. A *snapshot refresh* is an efficient batch operation that makes a 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 may 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.

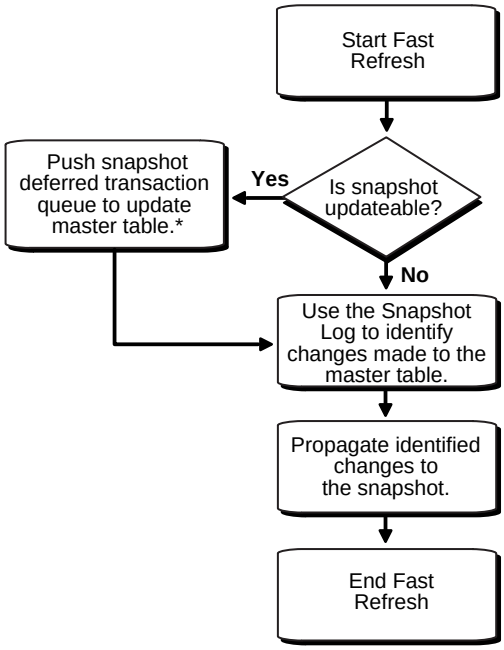
Note: If a snapshot is completely refreshed, set its PCTFREE to 0 and PCTUSED to 100 for maximum efficiency.

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 a smaller amount of data. You can perform fast refreshes of snapshots only when the master table has a snapshot log.

After a direct path load on a master table using SQL*Loader, a fast refresh does not apply the changes that occurred during the direct path load. Also, fast refresh does

not apply changes that result from other types of bulk load operations on master tables. Examples of these operations include some INSERT statements with an APPEND hint and some INSERT ... SELECT * FROM statements.

Figure 3–9 Fast Refresh of a Snapshot



* 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 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 a snapshot to refresh if a fast refresh is not possible.

Initiating a Refresh

When creating a refresh group, administrators may configure the group so that Oracle can automatically refresh the group's snapshots at scheduled intervals. Conversely, administrators may omit scheduling information so that the refresh group needs to be refreshed manually or "on-demand". Manual refresh is an ideal solution when the refresh 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 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, such as a button, that a salesperson 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.

Preparing 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.
- Allocate sufficient job processes.

The Replication Manager Setup Wizard automatically performs the tasks that are described below. The following discussion is provided to help you understand the replication environment and to help users who use the replication management API. After running Setup Wizard, create the necessary snapshot logs. See the Replication Manager online help for instructions on using Replication Manager to set up your snapshot site. You are encouraged to use Replication Manager whenever possible.

See Also: "Creating a Snapshot Log" on page 3-37.

If you are not able to use Replication Manager, review the "Set Up Snapshot Sites" section in Chapter 2 of the *Oracle8i Replication Management 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 Management API Reference* does to set up 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 must create and grant the necessary privileges to the snapshot administrator and to the refresher.

Create Master Site Users

You 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 are created.

Schema

A schema containing a snapshot in a remote database must 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.

> **See Also:** If you are reviewing the steps in *Oracle8i Replication Management API Reference*, the necessary schemas are created as part of the script described in Chapter 5, "Create Snapshot Group".

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.

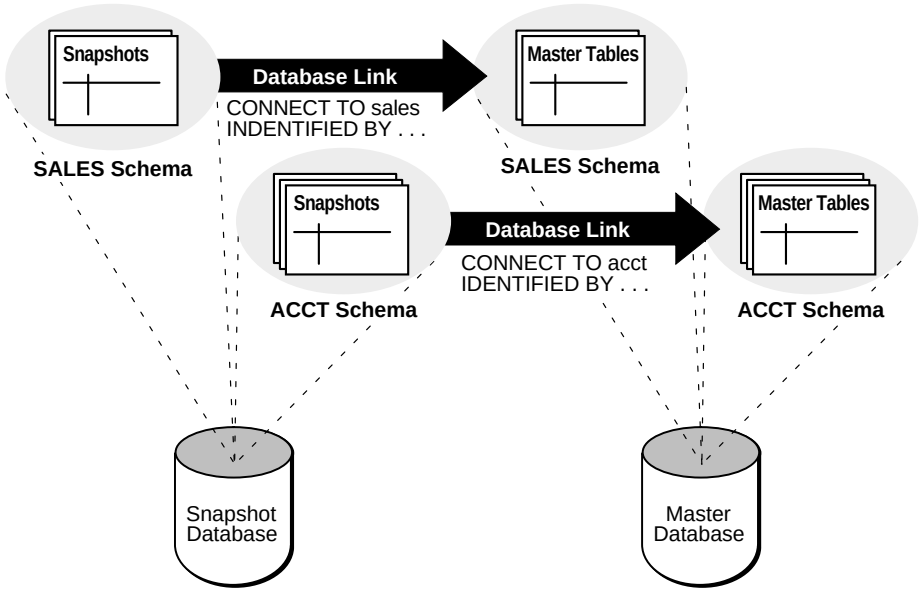
Before creating your snapshots, you need to create several administrative database links. Specifically, you should create a PUBLIC database link from the snapshot site to the master site; doing so makes defining your private database links easier because you do not need to include the USING clause in each link. You also need private database links from the snapshot administrator to the proxy administrator and from the propagator to the receiver, but, if you use the Replication Manager Setup Wizard, these database links are created for you automatically.

> **See Also:** Appendix A, "Security Options" in *Oracle8i Replication Management API Reference* for more information.

After the administrative database links have been created, a private database link must 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 to database 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 the snapshot must be able to issue the defining SELECT statement of the snapshot. The owner is the schema that contains the snapshot. If a user other than the replication or snapshot administrator creates the snapshot, then that user must have the CREATE SNAPSHOT privilege and the appropriate SELECT privileges to execute the defining SELECT statement.

> **See Also:** If you are reviewing the steps in *Oracle8i Replication Management API Reference*, the necessary privileges are granted as part of the script described in Chapter 5, "Create Snapshot Group".

Schedule Purge at Master Site

To keep the size of the deferred transaction queues 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; scheduling the purge operation again does 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 automatically propagates the deferred transactions at the snapshot site to the associated target master site. Typically, there is only a single scheduled link per snapshot group at a snapshot site, because a snapshot group only has a single target master site.

SNP Background Processes and Interval

It is important that you have allocated sufficient SNP (or job queue) background processes to handle the automatic refreshing of your snapshots. Because your snapshot site typically has only a single scheduled link to the target master site, the snapshot site only requires a single SNP process, but to handle additional activity, such as scheduled jobs, you may want to allocate at least two SNP processes at the snapshot site. Also, you need at least one SNP process for each degree of parallelism.

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

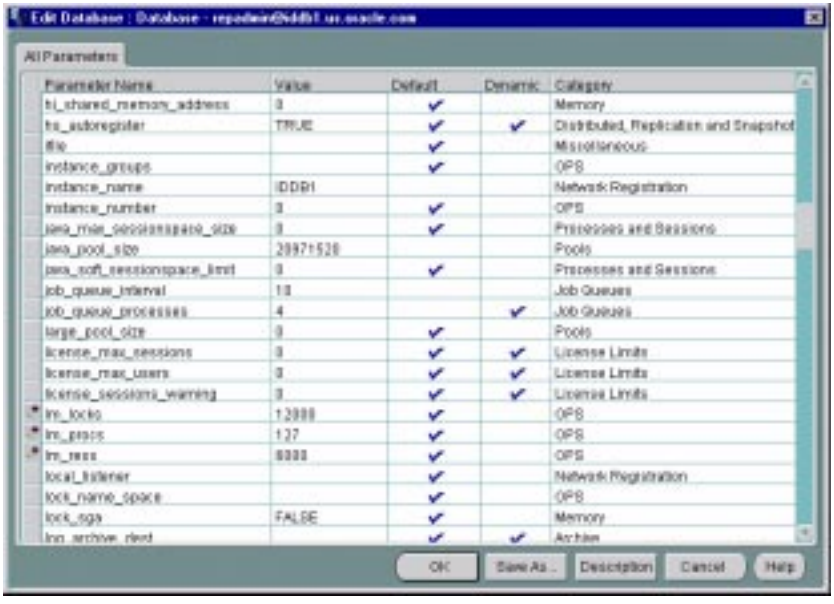
The SNP job interval determines how often your SNP processes “wake up” to execute any pending operations, such as pushing a queue. While the default value of 60 seconds is adequate for most replicated environments, you may need to adjust this value to maximize performance for your individual requirements. 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 want your SNP process to check for a pending operation only once an hour.

> **See Also:** "Initialization Parameters" on page 7-3 and the *Oracle8i Reference* for more information about JOB_QUEUE_PROCESSES.

Instance Manager

You will often use the Edit Database dialog box of Instance Manager to configure the SNP processes and the SNP job interval at the snapshot site if you have a dedicated network link to the snapshot site or if you are able to schedule the network link. This is required because Instance Manager is not installed at the snapshot site in most cases and thus the configuration must 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 Number of Job Processes



Complete the following to set your job processes using Instance Manager:

1. Start Instance Manager and connect to the target database.
2. Select the database in the Navigator pane.
3. On the General tab in the right pane, click the All Initialization Parameters button. The Edit Database dialog box.
4. Enter 2 (or a higher value) in the Value field next to the JOB_QUEUE_ PROCESS.

5. If necessary, modify your JOB_QUEUE_INTERVAL (entered values should be in seconds).
6. Click the OK button.

   You can save this configuration, which is helpful if you use Instance Manager to manage your database.

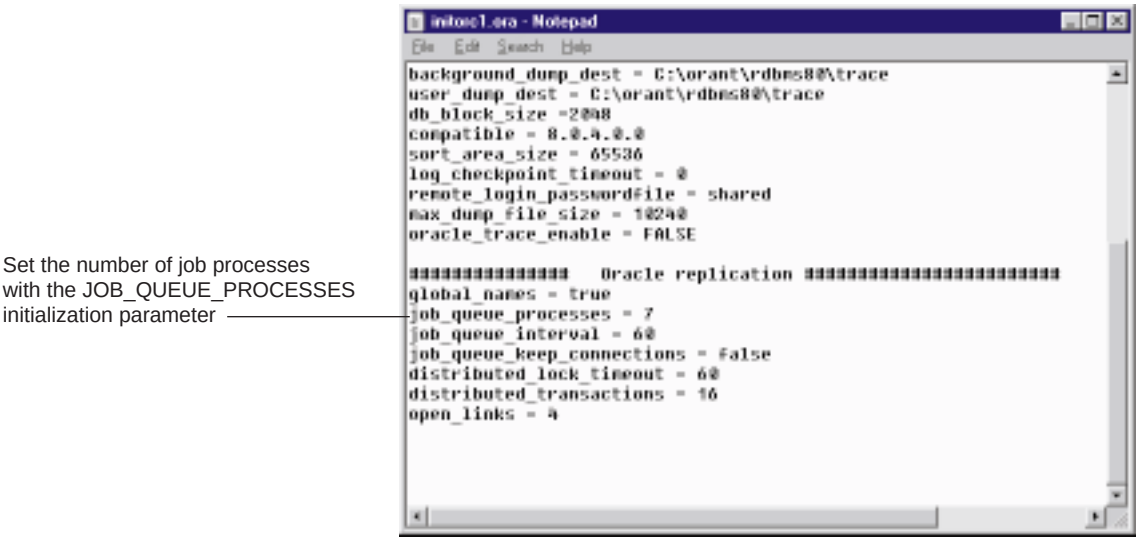
   See Also: *Oracle Enterprise Manager Administrator's Guide* and the Instance Manager online help for more information on using Instance Manager.

Manually Edit the Initialization Parameter File

If you do not have access to Instance Manager, you can manually edit the initialization parameter file. Use a text editor to modify the contents of your initialization parameter file.

In most cases, all of the initialization parameters used in replication are grouped together under an "Oracle replication" heading in your initialization parameter file.

Figure 3–12 Use a Text Editor to Edit Your Initialization Parameter File



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

> **See Also:** "Initialization Parameters" on page 7-3 the *Oracle8i Reference* for more information about initialization parameters important for Oracle replication, and see *Oracle8i Administrator's Guide* for information on restarting your database.

Creating 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, you need the following privileges:

- CREATE ANY TABLE
- CREATE ANY TRIGGER
- SELECT (on the snapshot log's master table)
- COMMENT ANY TABLE

> **See Also:** The "Creating Snapshot Logs" topic in the Replication Manager online help for detailed information about creating snapshot logs at the master site with Replication Manager. To access this topic in the online help, open Snapshot Replication in the Help Contents.

Using Filter Columns

When you create a snapshot log, you can specify whether you are using filter columns. Filter columns are an essential component when using subquery snapshots. A filter column must be defined in a snapshot log that is supporting a snapshot that references a column that is in a WHERE clause and is not part of the equijoin columns.

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.customer@dbs1.acme.com c
5)       WHERE o.c_id = c.c_id AND zip = 19555);
```

Notice in line 5 of the above DDL 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. Therefore, create a filter column in the snapshot log for the ZIP column of the SALES.CUSTOMER table.

You are encouraged to analyze the defining queries of your planned snapshots and identify which filter columns must be created in your snapshot logs. 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.

See Also:

- "Data Subsetting with Snapshots" on page 3-8 for information about snapshots with subqueries.
- "Creating a Snapshot Log" on page 3-37 for information about creating a snapshot log.
- "Restrictions for Snapshots with Subqueries" on page 3-14 for additional information about snapshots with subqueries.

Creating a Snapshot Environment

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

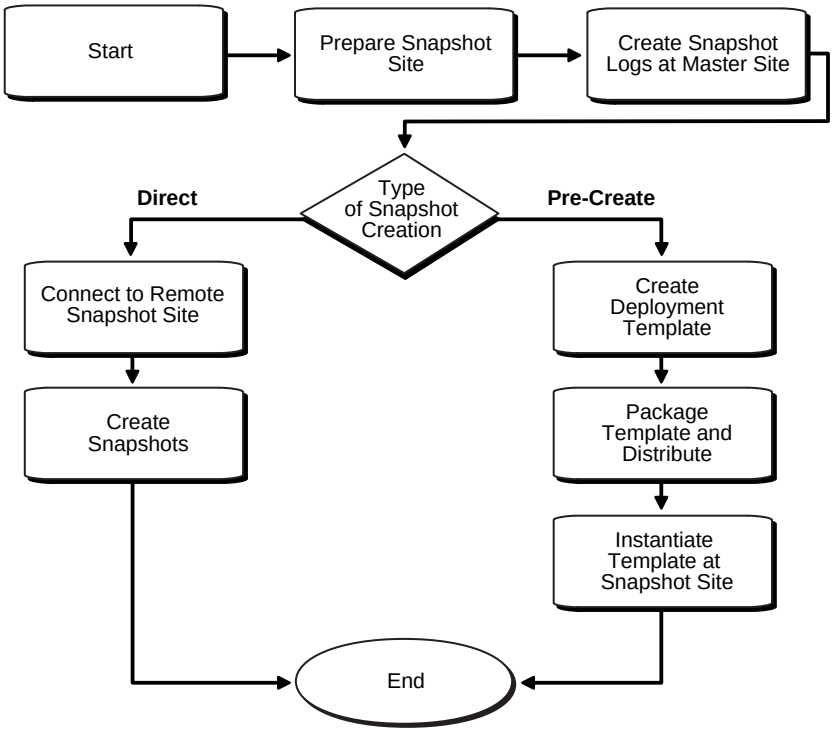
You can also individually create the snapshot environment by establishing a connection to the snapshot site and building the snapshot environment directly.

Replication Manager

See the Replication Manager online help for information on using deployment templates to centrally create a snapshot environment using Replication Manager.

See the Replication Manager online help for information on individually creating the snapshot environment with a direct connection to the remote snapshot site using Replication Manager.

Figure 3–13 Flowchart for Creating Snapshots



Replication Management API

See Chapter 4, "Create Deployment Template" of the *Oracle8i Replication Management API Reference* manual for information on using deployment templates to centrally pre-create a snapshot environment using the replication management API.

See Chapter 5, "Create Snapshot Group" of the *Oracle8i Replication Management 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

Deployment Templates Concepts & Architecture

This chapter introduces deployment templates and describes how to use them to easily and efficiently distribute snapshot environments. The chapter covers the following topics:

- Mass Deployment Challenge
- Oracle Deployment Templates Concepts
- Deployment Template Architecture
- Deployment Template Design
- Local Control of Snapshot Creation

> **Note:** Read Chapter 3, "Snapshot Concepts & Architecture" before you create a deployment template. Understanding snapshots better prepares you to build deployment templates.

Mass Deployment Challenge

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

The need to have accurate information at any time and at any place continues to grow rapidly. At the same time, information is becoming decentralized and users are often disconnected from the network, 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 roll out the data and the database infrastructure (tables, indexes, constraints, triggers, and so on) to all sites in an efficient and timely manner.

Traditionally, the DBAs have 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 customize 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 is limited to the delivery of data and data infrastructure.

Deployment Templates and the Mass Deployment Goal

Mass deployment tools and technologies should aid the database administrator in delivering the data and database infrastructure. The goal is to define the environment once and create as many instances of the deployment template as necessary, while still maintaining the ability to customize individual sites.

To support this goal, Oracle's deployment templates enable you to:

Define Snapshot Environment Once	You define the structure of a snapshot environment once using a deployment template so that each user (site) receives the database infrastructure to support the front-end application.
Customize Snapshot Sites Individually	You use deployment template parameters to customize each snapshot environment so that each user receives the particular data subset needed.

Mass deployment has many applications, such as distributing information to mobile sales forces, field technicians, retail stores, remote inventory collection sites, and so on. Such environments use deployment templates to build the database infrastructure at the remote site, largely because deployment templates support data subsetting, disconnected replication, and lower resource requirements, making them 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 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.

Deployment template features include the following:

- Centralized control
- Ability 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, you create a deployment template at the master site. This template stores all of the information needed to deploy a snapshot environment, including the data definition language (DDL) to create the objects at the remote site and the target refresh group. This template also maintains links to user security information and template parameters for custom snapshot creation.

Deployment Template Elements

Each deployment template contains the "blueprint" for creating the necessary snapshots and related objects at a snapshot site. Specifically, you create the deployment template at the master site, adding the necessary snapshots, triggers, views, and so on to the template as needed to create the snapshot environment. You 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 the following 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 data dictionary views. You add 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 can copy a deployment template to another site using the same deployment template name.

Figure 4–1 Refresh Group Template View Relationships

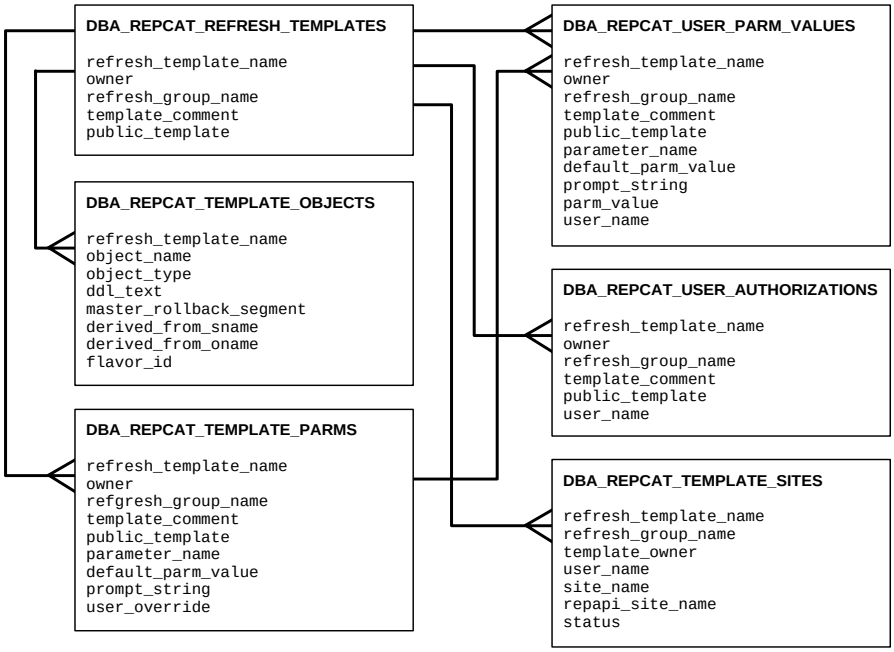
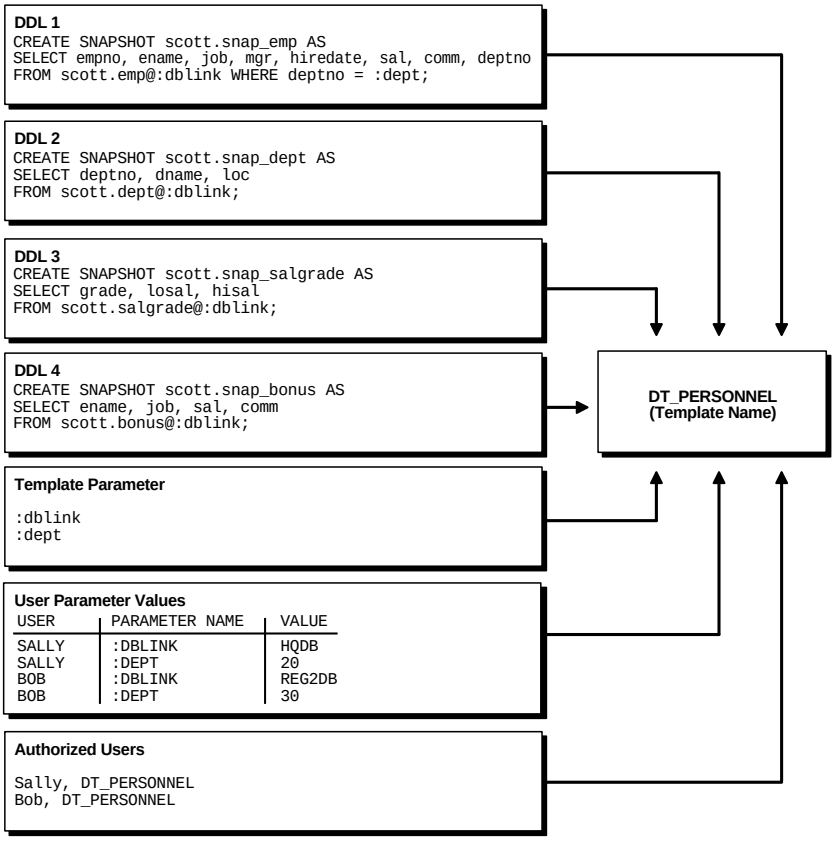


Figure 4–2 Deployment Template Elements Added to Template



Object Definitions

After the template has been defined, you add objects to the template. When the template is instantiated at the snapshot site, the object DDL (that is, CREATE SNAPSHOT, CREATE TABLE, and so on) is 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, but if necessary, you can create a new template object by defining DDL to create the object. Oracle checks any new object DDL to make sure that it is lexically correct, which prevents the execution of faulty DDL. Updateable snapshots added to a deployment template must be based on a table in a master group, but other

objects, such as read-only snapshots, can be based on objects that are not in master groups.

In most cases, you add snapshots to the template, but if necessary, you 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, you may even want to include all objects for an application in a deployment template. Snapshots created using a deployment template are automatically added to the refresh group defined for the template.

Deployment templates that are instantiated at Oracle8*i* Lite snapshot sites only build snapshots. If snapshots and other types of objects are included in a deployment template instantiated at an Oracle8*i* Lite site, the snapshots are built when the template is instantiated, but the other objects are not built.

> **See Also:** "General Template Information" on page 4-4 for more information about the refresh group.

Template Parameters

If each target snapshot site requires a data set unique to its site, you can define variables in the object DDL. These variables create a parameterized template that allows for custom data sets when the template is instantiated, allowing different snapshot sites to have different data sets. These parameters are embedded in the object DDL. During template instantiation, the individual user values for these parameters are substituted.

Oracle allows you to specify default values and user-specific parameter values for a template. You can enter the parameter values during the creation of the deployment template or after the template is created, but you must enter the parameter values before the template is instantiated. Users cannot enter values for parameters during instantiation.

If user-specific parameter values exist, then these values are automatically used when the specified user instantiates the template. For example, consider the variable REGION. Suppose you establish the following user-specific parameter values for template SALES_TEMP:

USER	REGION
SCOTT	EAST
LARRY	WEST

The defining SELECT statement for the snapshot is the following:

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

When users SCOTT and LARRY instantiate template SALES_TEMP, their resulting snapshot data sets are the following:

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

Template Parameters in the WHERE Clause and Security

In addition to creating customized data subsets, you can use template parameters in the WHERE clause of a CREATE SNAPSHOT statement to securely limit the snapshot site to viewing and changing only the data that satisfies the WHERE clause. For example, suppose you have specified the following for the REGION parameter in the user specific parameters list:

USER	REGION
SCOTT	EAST
LARRY	WEST

Users accessing the snapshot instantiated by user SCOTT only see data for region EAST and can only view, update, or delete data that complies with this WHERE clause. In other words, a user of this snapshot cannot view, update, or delete data for region WEST, because the snapshot only contains data for region EAST.

User Authorization

Deployment templates can be either public or private. You set this when you create the template. 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, then only authorized users can instantiate the target template. To enforce private use, you create a list of authorized users at the master site. If an unauthorized user attempts to instantiate the target template, the instantiation process fails.

Deployment Sites

Maintaining the emphasis on centralized control, you can monitor and manage certain characteristics of the instantiated environment at the remote snapshot site. Specifically, you have 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.

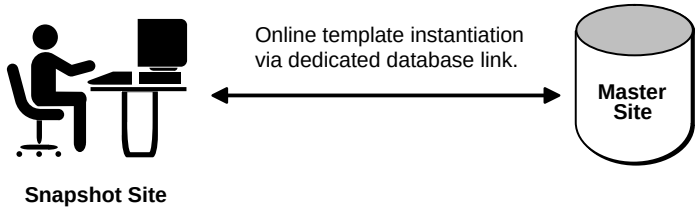
Packaging and Instantiating Deployment Templates

When you have completed defining your deployment template, the template needs to be *packaged* to prepare it for *instantiation* at the remote snapshot site. When the packaged deployment template is instantiated at a snapshot site, the snapshot site objects are created and the snapshots are populated with data. Remote snapshot sites can be created either through *online* or *offline* instantiation.

Online Instantiation

Online instantiation allows a snapshot site to instantiate a deployment template while connected to the target master site. During the online instantiation process, the structure of the snapshot site is created, and the specified data subset is pulled from the master site and stored in the appropriate snapshots.

Figure 4–3 Online Instantiation



For Oracle8*i* Enterprise Edition, Oracle8*i* Standard Edition, or Oracle8*i* Personal Edition snapshot clients, packaging a deployment template for online instantiation means generating a script file that, when run at the snapshot site, creates the snapshot objects and connects to the master site to populate the snapshots with data. SQL statements such as CREATE SNAPSHOT ... AS SELECT are used to populate the snapshots with data over a network from the master site.

When you package a template for online instantiation of an Oracle8*i* Lite snapshot client, no script or binary file is required for instantiation. A user connects to the master site with a client application, such as the Oracle Client Replication Tool, to both package and instantiate a deployment template in one operation.

One of the benefits of online instantiation is that the data subset is 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 increase network traffic.

Furthermore, laptop users connected by a modem may need to stay connected for a long time. The duration of the connection depends on the number of objects created, the complexity of the snapshot subqueries, and the amount of data transmitted, especially over low bandwidth modem lines.

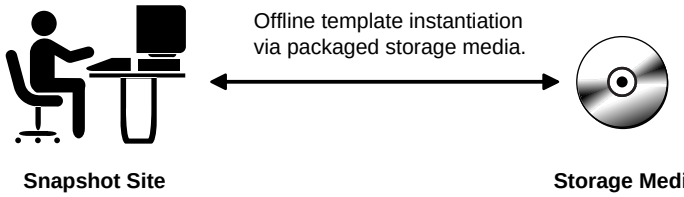
Offline Instantiation

To decrease server loads during peak usage periods and reduce remote connection times, you may choose offline instantiation of the template for your environment. Packaging a template for offline instantiation means generating a script or a binary file that contains the DDL and data manipulation language (DML) to build the snapshot environment defined in the deployment template and populate the environment with data. You package the script or binary file and save the file to some type of storage media (such as tape, CD-ROM, and so on), and then provide a means of transferring the script or binary file to the snapshot site. Each snapshot site requires a separate offline instantiation script.

When you package a template for instantiation, the snapshot logs for each master table on which a snapshot is based in the template begin to log changes. The snapshot log for a particular master table does not clear these changes until every snapshot based on the master table refreshes after instantiation. Therefore, to prevent the snapshot log from growing large, the template should be instantiated, and the snapshots should be refreshed as soon as possible after packaging.

During instantiation, the template and data are pulled from the storage media, instead of being pulled from the master site. This has the benefit of reducing network traffic and eliminating the need for a constant network connection. However, after instantiation, the data in the snapshot site reflects the master site data at packaging time and must be made current by a refresh.

Figure 4–4 Offline Instantiation



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

Scenarios for Instantiating a Deployment Template

The target instantiation site must run one of the following Oracle database clients:

- Oracle8*i* Enterprise Edition
- Oracle8*i* Standard Edition
- Oracle8*i* Personal Edition
- Oracle8*i* Lite

The most typical mass deployment scenario has Oracle8*i* Enterprise Edition at the master site and Oracle8*i* Lite at the remote site, which is often on a laptop.

Table 4–2 summarizes the scenarios for instantiating of a deployment template.

Table 4–1 Scenarios for Instantiating of a Deployment Template

Type of Instantiation	Type of Snapshot Client	Description
Offline	Oracle8*i* Enterprise Edition Oracle8*i* Standard Edition Oracle8*i* Personal Edition	The user runs the offline instantiation script with SQL*Plus. The offline instantiation script contains both CREATE statements to create snapshot site objects and INSERT statements to populate the snapshots with data.
Offline	Oracle8*i* Lite (Java RepAPI)	A user opens the binary file for offline instantiation with a client application tool, such as the Oracle Client Replication Tool. This tool then uses the binary file to create the snapshot site objects and populate the snapshots with data.
Online	Oracle8*i* Enterprise Edition Oracle8*i* Standard Edition Oracle8*i* Personal Edition	The user runs the online instantiation script with SQL*Plus. The online instantiation script contains CREATE statements to create snapshot site objects. When snapshot objects are created, the online instantiation script connects to the master site and uses CREATE SNAPSHOT ... AS SELECT statements to create the snapshots and populate them with data.
Online	Oracle8*i* Lite (Java RepAPI)	A user connects to the master site with a client application tool, such as the Oracle Client Replication Tool. This tool packages and instantiates the template in one operation. No file is necessary at the client.

Either you (the DBA) or the target user can package the deployment template. Either use Replication Manager's Offline Instantiation Wizard to package a template for offline instantiation, or the replication management API to package a template for offline or online instantiation. End-users use the public API to package a deployment template, while DBAs generally use the private API for packaging. Typically, when a deployment template will be instantiated offline, the DBA performs the packaging, but when the deployment template will be instantiated online, the user may perform the packaging. However, there are no restrictions on users or DBAs performing either online or offline packaging, other than the use of different API calls.

The following replication management API functions can be used to package a deployment template.

Private functions (DBA use only):

- DBMS_REPCAT_RGT.INSTANTIATE_OFFLINE
- DBMS_REPCAT_RGT.INSTANTIATE_OFFLINE_REPAPI
- DBMS_REPCAT_RGT.INSTANTIATE_ONLINE

Public functions:

- DBMS_REPCAT_INSTANTIATE.INSTANTIATE_OFFLINE
- DBMS_REPCAT_INSTANTIATE.INSTANTIATE_OFFLINE_REPAPI
- DBMS_REPCAT_INSTANTIATE.INSTANTIATE_ONLINE

See Also: "Preparing Snapshot Sites for Instantiation of Deployment Templates" on page 7-13, and see *Oracle8i Replication Management API Reference* for information about the functions.

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 should 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, because logging begins when the template is packaged.

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 to execute all of the stored DDL at the remote snapshot site, which creates 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 packaged, the stored object DDL is pulled from these system tables to create the instantiation script of binary file. The only exception to this is that online instantiation of an Oracle8*i* Lite snapshot site does not require a local file because a client application tool, such as the Oracle Client Replication Tool, performs the packaging automatically when it connects to the master site.

Use of 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 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 use the Replication Manager's Deployment Template Wizard, or you execute the CREATE_TEMPLATE_OBJECT function, as shown in the following example:

```
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;
/
```

Note: Do not place a terminating semi-colon in the DDL statement inside the single quotation marks for the DDL_TEXT parameter.

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 is created. In addition to creating snapshots, you can add table, trigger, procedure, index, and other object definitions to the deployment template.

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

See Also: DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT in the *Oracle8i Replication Management API Reference* for information about using this function.

Packaging and Instantiation Process

When a deployment template is packaged in preparation for remote snapshot site instantiation, the template is being prepared for online or offline instantiation. The instantiation procedure creates the remote snapshot environment and populates the environment with data.

Packaging a Deployment Template for Online Instantiation

When a deployment template is packaged for online instantiation, the resulting DDL that is required to create the remote snapshot environment is generated and all template parameter substitutions are performed. Where this generated DDL is stored depends on the type of snapshot client.

- If the snapshot client is Oracle8*i* Enterprise Edition, Oracle8*i* Standard Edition, Oracle8*i* Personal Edition, the online instantiation script is stored locally on the hard drive of the computer from which replication management API is executed to package the template. If this computer is not the snapshot site computer, the online instantiation file must be transferred to the snapshot site for online instantiation.
- If the snapshot client is Oracle8*i* Lite, no file is required. During an online instantiation, a user uses a client application tool, such as the Oracle Client Replication Tool, at the remote snapshot site to pull the snapshot site structure and data directly from the master site. Therefore, the client application tool performs the template packaging automatically.

Packaging a Deployment Template for Offline Instantiation

When a deployment template is packaged for offline instantiation, the DDL that is required to create the remote snapshot environment and the DML that is required to populate the environment with the data are both stored in a generated file. Also, during packaging, all template parameter substitutions are performed.

When a template is packaged, a script or binary file is created for offline instantiation and is saved to a storage device, such as hard disk, CD-ROM, tape, and so on. Either Replication Manager's Offline Instantiation Wizard or the replication management API can be used to package a deployment template for offline instantiation.

- If the snapshot client is Oracle8*i* Enterprise Edition, Oracle8*i* Standard Edition, Oracle8*i* Personal Edition, the offline instantiation script is stored locally on the hard drive of the computer from which the request is made to package the template. If this computer is not the snapshot site computer, the offline instantiation file must be transferred to the snapshot site for offline instantiation.
- If the snapshot client is Oracle8*i* Lite, the binary file for offline instantiation is stored at the master site and must be transferred to the snapshot site for offline instantiation.

When the remote snapshot site instantiates the template, the script or binary file is executed from the storage media or from the local hard drive. This execution creates the snapshot environment and populates the environment according to the data set defined during the packaging process. Recall that any template parameters that define the data set for individual sites are defined during the packaging process.

Online Instantiation

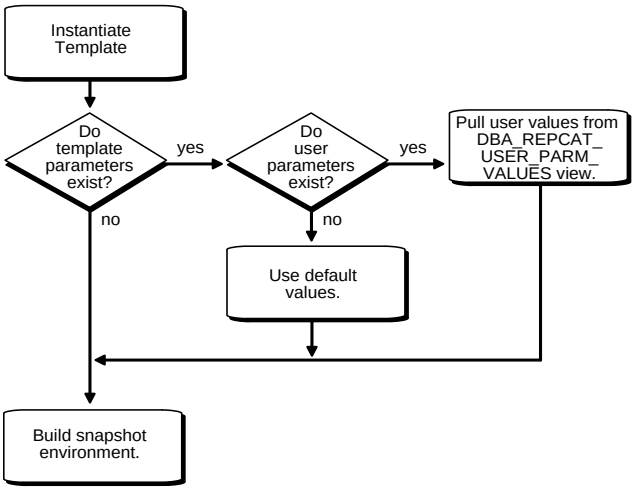
During the online instantiation process, the structure of the snapshot site is created, and the specified data subset is pulled from the master site and stored in the appropriate snapshots. Also, once the remote snapshot site begins the online instantiation process, Oracle evaluates the parameters that 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 can be installed at the remote snapshot site. At the same time, the snapshots are registered at the master site, and the snapshot logs begin logging the changes to the master tables.

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

1. User Parameter Values
2. Default Parameter Values

If user parameter values have been defined and a listed user is instantiating the template, the user parameter values are used when instantiating the template. If no user parameter values have been defined, Oracle uses the default parameter values. Figure 4–5 shows the parameter checking process.

Figure 4–5 Checking for Parameters During Online Instantiation



After the parameters are checked, the objects created by the template are added to the refresh group specified when the template was created.

Offline Instantiation

In a mass deployment environment, most snapshot environments use the offline instantiation method to create the necessary snapshot environment. When you package the deployment template, a script or binary file is created to store the DDL needed to create the snapshot environment, the parameter values used during the instantiation process, and the DML necessary to populate the snapshot environment with data.

The script or binary file 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.

Packaging and Instantiation Options

There are a number of possibilities for deployment template packaging and instantiation. Table 4–2 illustrates the possibilities, identifies the mechanism for packaging and instantiation, and lists the documentation to use when you perform an operation.

Table 4–2 Packaging and Instantiation Options **(Page 1 of 2)**

Type of Instantiation	Type of Client for Instantiation	Package Template Using	Packaging Documentation	Instantiate Template Using	Instantiating Documentation
Offline	Oracle8*i* Enterprise Edition Oracle8*i* Standard Edition Oracle8*i* Personal Edition	Replication Manager Offline Instantiation Wizard	See Replication Manager online help topic "Package for Offline Instantiation: Overview" under "Deployment Templates" > "Packaging and Instantiation" in the the Help Contents.	Offline Instantiation Script and SQL*Plus	See Replication Manager online help topic "Instantiate at Remote Snapshot Site" under "Deployment Templates" > "Packaging and Instantiation" in the the Help Contents.
Offline	Oracle8*i* Enterprise Edition Oracle8*i* Standard Edition Oracle8*i* Personal Edition	The Replication Management API (PL/SQL Packages and SQL*Plus)	See the "Package for Instantiation" section of Chapter 4 in *Oracle8i Replication Management API Reference*.	Offline Instantiation Script and SQL*Plus	See the "Instantiate Deployment Template" section of Chapter 4 in *Oracle8i Replication Management API Reference*.
Online	Oracle8*i* Enterprise Edition Oracle8*i* Standard Edition Oracle8*i* Personal Edition	The Replication Management API (PL/SQL Packages and SQL*Plus)	See the "Package for Instantiation" section of Chapter 4 in *Oracle8i Replication Management API Reference*.	Online Instantiation Script and SQL*Plus	See the "Instantiate Deployment Template" section of Chapter 4 in *Oracle8i Replication Management API Reference*.

Table 4–2 Packaging and Instantiation Options **(Page 2 of 2)**

Type of Instantiation	Type of Client for Instantiation	Package Template Using	Packaging Documentation	Instantiate Template Using	Instantiating Documentation
Offline	Oracle8*i* Lite (Java RepAPI)	Replication Manager Offline Instantiation Wizard	See Replication Manager online help topic "Package for Offline Instantiation: Overview" under "Deployment Templates" > "Packaging and Instantiation" in the the Help Contents.	Binary File for Offline Instantiation and a Client Application Tool, such as the Oracle Client Replication Tool	See the Oracle8*i* Lite documentation for information about instantiating the template.
Offline	Oracle8*i* Lite (Java RepAPI)	The Replication Management API (PL/SQL Packages and SQL*Plus)	See the "Package for Instantiation" section of Chapter 4 in *Oracle8i Replication Management API Reference*.	Binary File for Offline Instantiation and a Client Application Tool, such as the Oracle Client Replication Tool	See the Oracle8*i* Lite documentation for information about instantiating the template.
Online	Oracle8*i* Lite (Java RepAPI)	A Client Application Tool, such as the Oracle Client Replication Tool	See the Oracle8*i* Lite documentation for information about packaging the template.	A Client Application Tool, such as the Oracle Client Replication Tool	See the Oracle8*i* Lite documentation for information about instantiating the 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.

With respect to offline instantiations, the longer the duration between the packaging at the server and the instantiation at the remote site, the longer it takes for the first refresh after instantiation at the remote snapshot site. The snapshot site uses 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 are logged to the snapshot log as soon as you package the deployment template.

> **See Also:** "Snapshot Architecture" on page 3-17 for more information.

Snapshot Groups

Objects created by an instantiated deployment template are added automatically 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 are 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 Also: "Snapshot Groups" on page 3-23 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. After the instantiation process is finished, you can specify that the snapshots in the refresh group be refreshed automatically at set intervals, assuming a constant network connection to the master site.

For Oracle8*i* Enterprise Edition, Oracle8*i* Standard Edition, or Oracle8*i* Personal Edition, Replication Manager or DBMS_REFRESH.CHANGE procedure can be used to change the refresh interval and next refresh data of a refresh group. To change these settings in Replication Manager, select the refresh group and edit the Next Date and Interval fields. To change these settings with the DBMS_ REFRESH.CHANGE procedure, set the interval and next_date parameters appropriately. If snapshot sites do not have a constant network connection to the master site, they can refresh their refresh groups on-demand.

Note: Refresh groups at Oracle8*i* Lite sites can only be refreshed on-demand, not at set intervals. Also, 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 Oracle8*i* Lite site.

Deployment Template Design

Before you begin assembling your deployment template, you should consider how you want to build your templates. The combination of deployment template parameters and subquery subsetting gives the database administrator a powerful tool to administer a widely distributed database environment using *assignment tables* and horizontally partitioned data. Additional design consideration must be given to vertical partitioning requirements and data sets needed for a replicated environment.

> **See Also:** "Data Subsetting with Snapshots" on page 3-8 for more information on data subsetting, and see "Vertical Partitioning" on page 4-22 for more information on vertical partitioning.

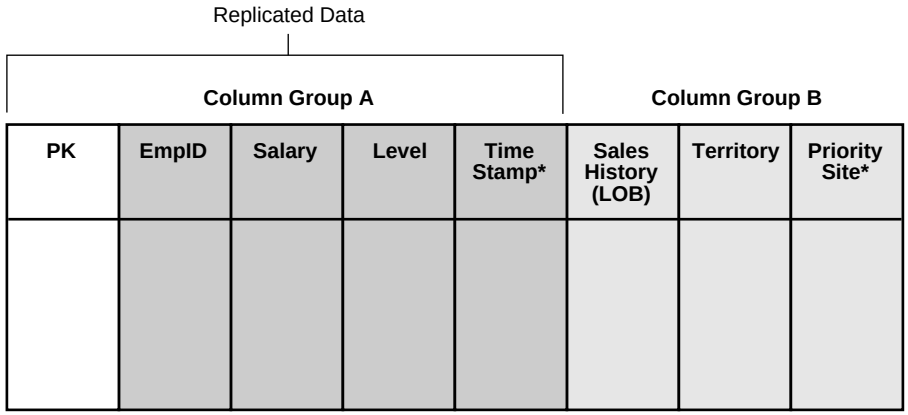
Vertical Partitioning

Deployment templates offer the additional advantage of allowing you to build updateable snapshots that are vertically partitioned. Vertical partitioning is also referred to as column subsetting. 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 4–6)

Figure 4–6 Replicate Vertically Partitioned Data



If you are adding a snapshot object that replicates columns PK, EmpID, Salary, and Level (illustrated in Figure 4–6), you also need to include the Time Stamp column because it is used for conflict resolution for columns contained in Column Group A.

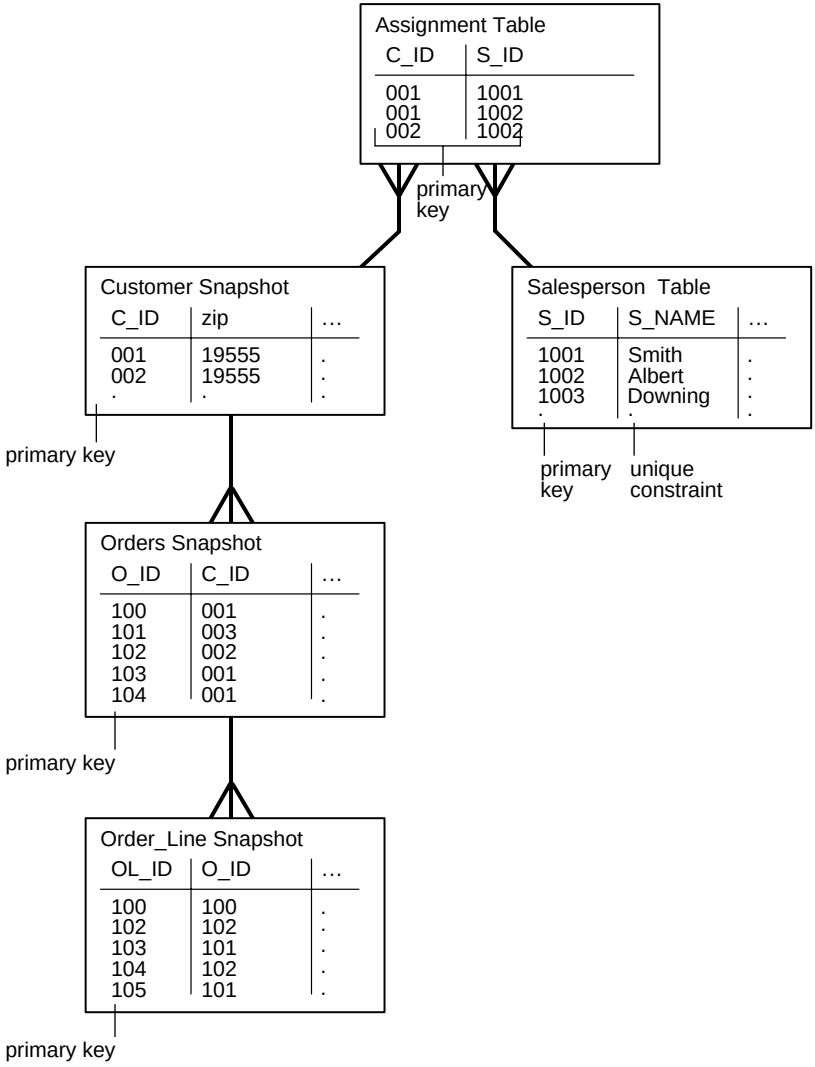
Note: Vertical partitioning is only available when you add a snapshot object to a deployment template using Oracle Replication Manager. Vertical partitioning is not available when using the replication management API.

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.

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, you can easily control changes to the snapshot environment.

For example, consider the customer / salesperson relationship in Figure 4–7. In this example, a salesperson is assigned 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, make this administration easy while ensuring security. For example, salesperson #1001 cannot view the customer information of other salespersons, which is very important if the customer information contains sensitive data.

Considering the relationships pictured in Figure 4–7 above, if the ORDERS snapshot's defining query was specified as the following:

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

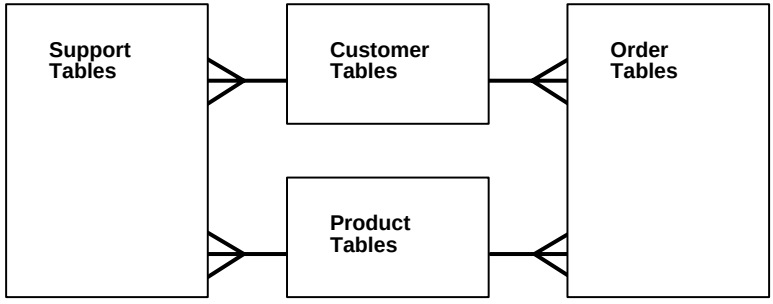
Then, the ORDERS snapshot is populated with order data for the customers that are assigned to the salesperson specified for the *:salesperson_id* variable. Notice the *:salesperson_id* variable in the last line of the CREATE SNAPSHOT statement.

With this flexibility, managers can easily control snapshot data sets by making simple changes to the assignment table, without requiring you 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 is 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 is removed from the snapshot site after the next refresh and the salesperson can no longer access that information.

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, both salespersons and technicians need customer information, but the technicians may not need sales information.

Figure 4–8 Some Users Require All Data



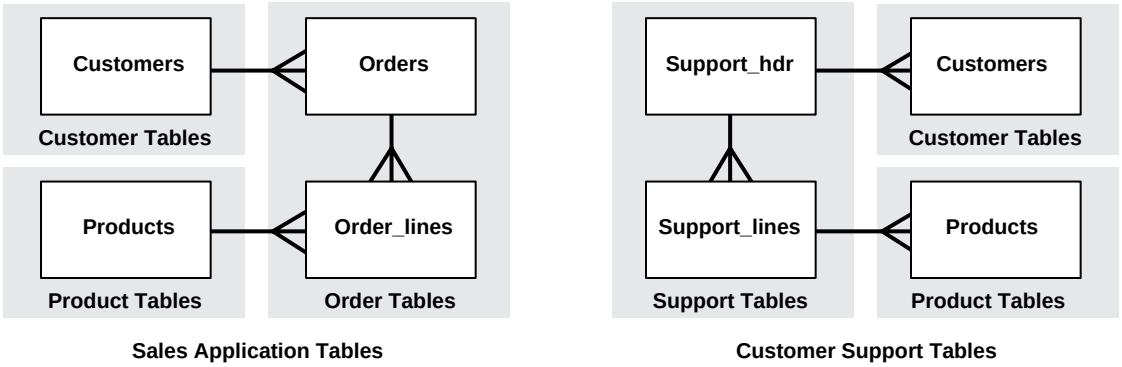
You do not want users to instantiate deployment templates that may contain extraneous data, because it will only require extra storage space and additional refresh times.

On the other hand, if you have users that require both sales and customer support information, you do not 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 templates 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 salespersons, one for customer service technicians, and one for users that require both sets of data.

To save time and effort, 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 4–9 The Different Needs of Salespersons And Customer Support Technicians



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 once, and it would be used for all objects during the instantiation process.

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.customer@hq.acme.com c
    WHERE o.c_id = c.c_id
      AND EXISTS
    ( SELECT * FROM sales.assignment@hq.acme.com a
     WHERE a.c_id = c.c_id
     AND EXISTS
     ( SELECT * FROM sales.salesperson@hq.acme.com s
       WHERE s.s_id = ':salesperson_id')));
```

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

Even though the two snapshots being created do not 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. Consider 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 does the snapshot environment require?
- Do any tables need to be created that store non-replicated data?
- Are any extra indexes required?

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 to only two or three sites, you still significantly reduce the amount of steps needed to build a snapshot environment by 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.

However, one question remains: 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 should build a snapshot environment using the Snapshot Group Wizard or

locally at the snapshot site when local control must 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 must retain some control.

Local control may be required if the snapshot site:

- Has an experienced DBA
- Is considered a trusted site
- Is a snapshot instead of a master site because of horizontal data partitioning requirements

Because snapshot groups are created with Replication Manager's Snapshot Group Wizard locally at the snapshot site by its DBA, or perhaps 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. Because multimaster replication does not allow for vertical and horizontal data partitioning, updateable snapshot sites are sometimes created primarily for their ability to partition data. These sites are typically secure, have experienced DBAs, and require the ability to maintain control locally to meet user and application requirements. Snapshot groups created with the Snapshot Group Wizard or with the replication management API allow for the localized control necessary to meet the requirements of the secure updateable snapshot sites.

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 fine for most installations, certain situations may require that the objects in a snapshot group are assigned to several different refresh groups.

5

Conflict Resolution Concepts & Architecture

This chapter covers the following topics:

- Conflict Resolution Concepts
- Conflict Resolution Architecture

Conflict Resolution Concepts

Replication conflicts can occur in a 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 a 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 is replicated. However, many applications require that some percentage of data be updateable at multiple sites at any time. If this is the case, you must address the possibility of replication conflicts.

The next few sections introduce information about replication conflicts, how to design a 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 its 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 (such as order-entry, shipping, billing, and so on). 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 a replication environment. Start by considering the general requirements of the applications that will work with the replicated data. For example, some applications might work fine with read-only 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

There are three types of data conflicts that you may encounter in a replicated database environment: update, uniqueness, and delete conflicts. As you will learn through the remainder of this chapter, you will most likely encounter update conflicts in your replication environment, though you should always prepare to handle uniqueness and delete conflicts. Oracle Corporation recommends that your database design works to avoid these types of 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, such as 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 and another transaction updating or deleting the same row, because in this case the row does not exist to be either updated or deleted.

Data Conflicts and Transaction Ordering

Conflicts

Ordering conflicts can occur in replication environments 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 are recorded in the error log and can be re-executed after the transactions they depend upon are propagated and applied. See Table 5–1 on page 5-9 to see an example of an ordering conflict.

To guarantee data convergence in replication environments with three or more master sites, you must select a conflict resolution method that can guarantee data convergence with any number of master sites (latest timestamp, minimum[1], maximum[1], priority group[1], additive[1]).

Referential Integrity

In addition to receiving a data conflict, replicated transactions that are applied out-of-order might experience referential integrity problems at a remote site if supporting data was not successfully propagated to that site. Consider the scenario where a new customer calls an order department; a customer record is created and an order is placed. If the order data is propagated to a remote site before the customer data, a referential integrity error is raised because the customer that the order references does not exist at the remote site.

If a referential integrity error is encountered, you can easily resolve the situation by re-executing the transaction in error after the supporting data has been propagated to the remote site.

[1] Conflict resolution method guarantees data convergence with any number of master sites as long as certain conditions exist. See the appropriate conflict resolution method in the "Conflict Resolution Architecture" section on page 5-10 for more information.

Detecting Conflicts

Each master site in a 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 can automatically detect if any replication conflicts exist.

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 a 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 values 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 "Performance Mechanisms" on page 5-26.

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 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.

Caution: 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

Once a conflict has been detected, it is important to resolve the conflict with the goal of data convergence across all sites. Oracle provides several prebuilt conflict resolution methods to resolve update conflicts and in many situations can guarantee data convergence across a variety of replication environments. Oracle also offers several conflict resolution methods to handle uniqueness conflicts, though these methods cannot guarantee data convergence.

Oracle does not provide any prebuilt conflict resolution methods to handle delete or ordering conflicts. Oracle does, however, allow you to build your own conflict resolution method to resolve data conflicts specific to your business rules. If you do build a conflict resolution method that cannot guarantee data convergence, which is likely for uniqueness and delete conflicts, you should also build a notification facility to notify the database administrator so that data convergence can be manually achieved.

Whether you use an Oracle prebuilt or user-defined conflict resolution method, it is applied as soon as the conflict is detected. If the defined conflict resolution method cannot resolve the conflict, the conflict is logged in the error queue.

To avoid a single point of failure for conflict resolution, you can define additional conflict resolution methods to backup the primary method. For example, in the unlikely event that the latest timestamp conflict resolution method cannot resolve a conflict because the timestamps are identical, you may want to define a site priority conflict resolution method, which breaks the timestamp tie and resolves the data conflict.

Avoiding Conflicts

Though Oracle provides powerful methods for resolving data conflicts, one of your highest priorities when designing a replicated database and front-end application should be to avoid data conflicts. 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 that 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 the 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.

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 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.

> **See Also:** "Using Dynamic Ownership Conflict Avoidance" on page 6-11 for more information about using dynamic ownership data models.

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 a replication environment to prevent the possibility of uniqueness conflicts. For example, you can create sequences at each site so that each sequence at each site generates a mutually exclusive set of sequence numbers. This solution, however, can become problematic as the number of sites increase or the number of entries in the replicated table grows.

Alternatively, you can append a unique site identifier as part of a composite primary key.

Finally, beginning with Oracle8*i*, release 8.1.5, you can select a globally unique value using the SYS_GUID () function; using the selected value as the primary key (or unique) value will globally avoid uniqueness conflicts.

Note: Sequences are not valid replication object types and you must therefore create the sequence at each site.

See Also: "Sequences" on page 2-18 for more information.

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 should mark rows for deletion and then configure the system to periodically purge logically deleted rows using procedural replication.

See Also: The "Create Conflict Avoidance Methods for Delete Conflicts" section in Chapter 6 of the *Oracle8i Replication Management API Reference* to learn how to prepare a table for delete avoidance and build a replicated procedure to purge marked rows.

Avoiding Update Conflicts After trying to eliminate the possibility of uniqueness and delete conflicts in a 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.

Avoiding Ordering Conflicts Whenever possible, however, it is best to avoid or automatically resolve ordering conflicts. For example, you should select conflict resolution methods that ensure convergence in multimaster configurations where ordering conflicts are possible.

The example in Table 5–1 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 5–1 Example: Ordering Conflicts With Site Priority Conflict Resolution

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

Conflict Resolution Architecture

There are very few architectural mechanisms and processes that are visible when implementing conflict resolution into your replication environment. This section describes the few supporting mechanisms involved in conflict resolution, but for the most part, the majority of this section describes the different aspects of Oracle's prebuilt conflict resolution methods.

Support Mechanisms

The most important mechanism that is involved in Oracle conflict resolution is the column group; all update conflict detection and resolution is based on the column group. Additionally, the error queue can provide you with important information to monitor the conflict detection activity of your replication environment.

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.

If your replicated table contains multiple column groups, each group is viewed independently when analyzing updates for conflicts. For example, consider that columns A1, A2, and A3 belong to Column Group A and columns B1, B2, and B3 belong to Column Group B. If user01 updates column A1 of row 1 at site01 and user02 updates column B2 of row 1 at site02, both sets of updates are replicated and do not result in a data conflict.

Because the same row was updated at multiple sites, how are both sets of updates replicated? Because each update occurred in different column groups and Oracle analyzes each column group independently, no conflict occurred. If user02 had updated column A2 of row 1 at site02, then a conflict would have occurred.

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 postal 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 postal 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. The shadow column group of a table 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.

Error Queue

If a conflict resolution method fails to resolve a data conflict, or if you have not defined any conflict resolution methods, the error queue contains information about the data conflict.

> **See Also:** "Error Queue" on page 2-20 for more information about the error queue.

Common Update Conflict Resolution Methods

Though Oracle provides eight prebuilt update conflict resolution methods, the *latest timestamp* and the *overwrite* conflict resolution methods are the most commonly implemented resolution methods.

These methods are the most common because they are conceptually easy to use and, in the proper environments, can guarantee data convergence. The latest timestamp and the overwrite conflict resolution methods are described in detail in the following two sections.

Table 5–2 Convergence Properties of Common Update Conflict Resolution Methods

Resolution Methods	One Master Site	Any Number of Master Sites
LATEST TIMESTAMP	YES (with backup method)	YES (with backup method)
OVERWRITE	YES	NO

Latest Timestamp

The *latest timestamp* method resolves a conflict based on the most recent update, as identified by the timestamp of when the update occurred.

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.

Target Environments The latest timestamp conflict resolution method works to converge replication environments with two or more master sites. Because time is always increasing, it is one of the few conflict resolution methods that can guarantee data convergence with multiple master sites. This resolution also works well with any number of snapshots.

Support Mechanisms 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.

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.

Timestamp 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.

Implement Latest Timestamp See the Replication Manager online help to learn how to define a latest timestamp conflict resolution method with Replication Manager. See the "Timestamp" section in Chapter 6 of the *Oracle8i Replication Management API Reference* book to learn how to define this type of conflict resolution method with the replication management API.

Overwrite

The *overwrite* method ignores the values from the originating 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. You can also use this form of conflict resolution with multiple master sites, though it does not guarantee data convergence and should be used 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.

Target Environments The overwrite conflict resolution method ensures data convergence for replication environments that have a single master site with any number of snapshots. With this in mind, the overwrite conflict resolution method is ideal for mass deployment environments.

If a conflict is detected, the value originating from the snapshot site is used, which means that priority is given to the most recently refreshed snapshots.

Support Mechanisms No additional support mechanisms are required for the overwrite conflict resolution method.

Implement Overwrite See the Replication Manager online help to learn how to define an overwrite conflict resolution method with Replication Manager. See the "Overwrite and Discard" section in Chapter 6 of the *Oracle8i Replication Management API Reference* book to learn how to define this type of conflict resolution method with the replication management API.

Additional Update Conflicts Resolution Methods

If the latest timestamp and/or the overwrite conflict resolution methods do not meet your needs to resolve data conflicts that are encountered in your replication environment, Oracle offers six additional pre-built update conflict resolution methods.

Table 5–3 Convergence Properties of Additional Update Conflict Resolution Methods

Resolution Methods	One Master Site	Any Number of Master Sites
ADDITIVE	YES	YES
AVERAGE	YES	NO
DISCARD	YES	NO
EARLIEST TIMESTAMP	YES (with backup method)	NO
MAXIMUM	YES	YES (column values must always increase)
MINIMUM	YES	YES (column values must always decrease)
PRIORITY GROUP	YES	YES (with ordered update values)
SITE PRIORITY	YES	NO

Additive

The *additive* method works with column groups consisting of a single numeric column only. If a conflict arises, instead of choosing one value over another, the difference of the two values is added to the current value.

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 and snapshot sites.

Target Environments The additive conflict resolution method is designed to conserve data rather than choose the most appropriate data. This method might be very useful in a financial environment where deposits and withdraws might happen so frequently that conflicts may arise; with a balance, it is important to conserve data rather than choose one value over another (though we might wish that deposits would always be chosen over withdraws).

Support Mechanisms No additional support mechanisms are required for the additive conflict resolution method.

Implement Additive See the Replication Manager online help to learn how to define an additive conflict resolution method with Replication Manager. See the "Additive and Average" section in Chapter 6 of the *Oracle8i Replication Management API Reference* book to learn how to define this type of conflict resolution method with the replication management API.

Average

Like the additive method, the *average* method works with column groups consisting of a single numeric column only. Instead of adding the difference to the current value, the average method resolves the conflict by computing the average of the current and the new value.

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 site.

Target Environments Because the average method cannot guarantee data convergence for replication environments with more than one master site, the average method is ideally implemented in mass deployment environment with a single master site and any number of updateable snapshots.

The average method might be useful for scientific applications that would rather average two values instead of choosing one value over another (for example, to compute the average temperature or weight).

Support Mechanisms No additional support mechanisms are required for the average conflict resolution method.

Implement Average See the Replication Manager online help to learn how to define an average conflict resolution method with Replication Manager. See the "Additive and Average" section in Chapter 6 of the *Oracle8i Replication Management API Reference* book to learn how to define this type of conflict resolution method with the replication management API.

Discard

The discard method ignores the values from the originating site and therefore can never guarantee convergence with more than one master site. This method is 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.

Target Environments The discard conflict resolution method is best suited for a mass deployment model having a single master site with any number of snapshot sites. If a conflict is detected, the value originating from the snapshot site is ignored, which means that priority is given to snapshots that refresh first.

Support Mechanisms No additional support mechanisms are required for the discard conflict resolution method.

Implement Discard See the Replication Manager online help to learn how to define a discard conflict resolution method with Replication Manager. See the "Overwrite and Discard" section in Chapter 6 of the *Oracle8i Replication Management API Reference* book to learn how to define this type of conflict resolution method with the replication management API.

Earliest Timestamp

The *earliest timestamp* methods resolves a conflict based on the earliest (oldest) update, as identified by the timestamp of when the update occurred.

Target Environments The earliest timestamp conflict resolution method works to converge replication environments with a single master site and any number of snapshots. Because time is always increasing, the earliest timestamp conflict resolution cannot guarantee data convergence in replication environments with more than one master site. This resolution also works well with any number of snapshots, if you have a backup conflict resolution method in the event that two transactions have the same timestamp.

Support Mechanisms 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. Be sure to review the "Timestamp Configuration Issues" discussion on page 5-13.

> **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.

Implement Earliest Timestamp See the Replication Manager online help to learn how to define an earliest timestamp conflict resolution method with Replication Manager. See the "Timestamp" section in Chapter 6 of the *Oracle8i Replication Management API Reference* book to learn how to define this type of conflict resolution method with the replication management API.

Maximum

When the replication facility detects a conflict with a column group and calls the *maximum* 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 *greater 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 less 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.

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 the column value is always increasing.

> **Note:** You should not enforce an always-increasing restriction by using a CHECK constraint because the constraint could interfere with conflict resolution.

Target Environments If you have defined the maximum conflict resolution method and the target column that is used to resolve the conflict is always increasing across all sites, this method guarantees data convergence with any number of master and snapshot sites.

Support Mechanisms No additional support mechanisms are required for the maximum conflict resolution method.

Implement Maximum See the Replication Manager online help to learn how to define a maximum conflict resolution method with Replication Manager. See the "Minimum and Maximum" section in Chapter 6 of the *Oracle8i Replication Management API Reference* book to learn how to define this type of conflict resolution method with the replication management API.

Minimum

When the 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.

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 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.

Target Environments If you have defined the minimum conflict resolution method and the target column that is used to resolve the conflict is always decreasing across all sites, this method guarantees data convergence with any number of master and snapshot sites.

Support Mechanisms No additional support mechanisms are required for the minimum conflict resolution method.

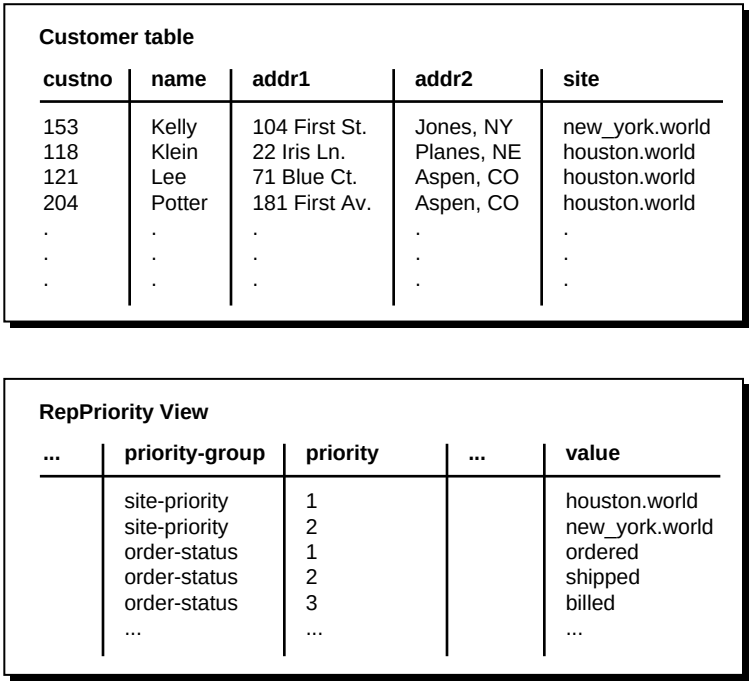
Implement Minimum See the Replication Manager online help to learn how to define a minimum conflict resolution method with Replication Manager. See the "Minimum and Maximum" section in Chapter 6 of the *Oracle8i Replication Management API Reference* book to learn how to define this type of conflict resolution method with the replication management API.

Priority Groups

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

As shown in Figure 5–1, the DBA_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 5–1 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 DBA_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. In the order-status priority group in this example, "billed" (priority 3) has a higher priority than "shipped" (priority 2), and "shipped" has a higher priority than "ordered" (priority 1).

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.

Target Environments The priority group conflict resolution method is useful for replicated environments that have been designed for a work flow environment. For example, once an order has reached the "shipping" status, updates from the "order entry" department are always over-written.

Support Mechanisms You need to define the priority of the values contained in the target column. This priority definition is required so that Oracle knows how to resolve a conflict based on the priority of the column value that has been designated to resolve a conflict. The priority definitions are stored in a priority group.

Implement Priority Groups See the Replication Manager online help to learn how to define a priority group conflict resolution method with Replication Manager. See the "Priority Groups" section in Chapter 6 of the *Oracle8i Replication Management API Reference* book to learn how to define this type of conflict resolution method with the replication management API.

Site Priority

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 DBA_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 DBA_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.

Target Environments Just like priority groups, site priority conflict resolution is commonly implemented in a work-flow environment. Additionally, when the site priority conflict resolution method is used in a mass deployment environment (which is a single master site and any number of snapshots), data convergence can be guaranteed.

The site priority conflict resolution method is also a good backup conflict resolution method should a primary conflict resolution method fail.

Support Mechanisms A column must be designated to store site information when a row is updated. Additionally, you need to create a trigger that populates this site column with the global name of the updating site when a row is either updated or inserted. A sample of this trigger is contained in the Replication Manager online help and in the "Site Priority" section in Chapter 6 of the *Oracle8i Replication Management API Reference* book.

You also need to define the priority of the sites that participate in your replication environment. This priority definition is required so that Oracle knows how to resolve a conflict based on the priority of the site that performed the update / insert. The site priority definitions are stored in a priority group.

Implement Site Priority See the Replication Manager online help to learn how to define a site priority conflict resolution method with Replication Manager. See the "Site Priority" section in Chapter 6 of the *Oracle8i Replication Management API Reference* book to learn how to define this type of conflict resolution method with the replication management API.

Uniqueness Conflicts Resolution Methods

Oracle provides three prebuilt methods for resolving uniqueness conflicts:

- Append the global site 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.

Append Site Name

The *append site name* method works by appending the global database name of the site originating the transaction to the replicated column value that is generating a DUP_VAL_ON_INDEX exception. Although this method allows the column to be inserted or updated without violating a unique integrity constraint, it does not provide any form of convergence between multiple master sites. The resulting discrepancies must be manually resolved; therefore, this method is 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.

This method 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.
- 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.

Append Sequence

The *append sequence* methods works by appending a generated sequence number to the column value that is generating a DUP_VAL_ON_INDEX exception. Although this method allows the column to be inserted or updated without violating a unique integrity constraint, it does not provide any form of convergence between multiple master sites. The resulting discrepancies must be manually resolved; therefore, this method is 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.

This method 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 sequence.
- Use a notification scheme to alert the appropriate person to resolve the duplication, instead of logging a conflict.

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.

Delete Conflict Resolution Methods

Oracle does not provide any prebuilt methods for resolving delete conflicts. As discussed in the "Avoiding Delete Conflicts" section on page 5-8, you should design your database and front-end application to avoid delete conflicts, which is achieved by marking rows for deletion and at regular intervals, using procedural replication to purge such marked rows.

See "Avoiding Delete Conflicts" on page 5-8 to learn how to avoid encountering delete conflicts. See Chapter 6, "Conflict Resolution" in the *Oracle8i Replication Management API Reference* to learn how to build conflict avoidance into your replication environment.

Performance Mechanisms

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.

Minimum Communication

When you create a replicated table and all participating sites are Oracle8 or greater databases, you can choose to minimize the amount of data that must be communicated to determine conflicts for each changed row in the table by enabling the *minimum communication* feature. When minimum communication is enabled, 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 transmits 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.

Send and Compare Old Values

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_OLD_VALUES procedure and the DBMS_REPCAT.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.

Note: 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 the SEND_OLD_VALUES and COMPARE_OLD_VALUES procedures to further reduce data propagation reduces protection against unexpected conflicts**.**

To further reduce data propagation, execute the following procedures:

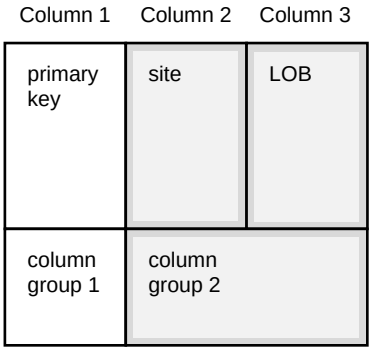
```
DBMS_REPCAT.SEND_OLD_VALUES(
   sname           IN  VARCHAR2,
   oname           IN  VARCHAR2,
   { column_list   IN  VARCHAR2,
   | column_table  IN  DBMS_REPCAT.VARCHAR2s,}
   operation       IN  VARCHAR2 := 'UPDATE',
   send            IN  BOOLEAN  := TRUE );

DBMS_REPCAT.COMPARE_OLD_VALUES(
   sname           IN  VARCHAR2,
   oname           IN  VARCHAR2,
   { column_list   IN  VARCHAR2,
   | column_table  IN  DBMS_REPCAT.VARCHAR2s,}
   operation       IN  VARCHAR2 := 'UPDATE',
   compare         IN  BOOLEAN  := TRUE );
```

After executing these procedures, you must use the DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT procedure to generate replication support 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 DBA_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 these procedures. Consider a table called SCOTT.REPORTS with three 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 use the DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT procedure to 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 calls:

```
DBMS_REPCAT.SEND_OLD_VALUES(
   sname           IN  'SCOTT',
   oname           IN  'REPORTS'
   column_list     IN  'C3',
   operation       IN  'UPDATE',
   send            IN  FALSE );

DBMS_REPCAT.COMPARE_OLD_VALUES(
   sname           IN  'SCOTT',
   oname           IN  'REPORTS'
   column_list     IN  'C3',
   operation       IN  'UPDATE',
   compare         IN  FALSE);
```

You must use the DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT procedure to generate replication support for SCOTT.REPORTS 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.

Note: Oracle conflict resolution does not support piecewise updates of LOBs.

See Also: The *Oracle8i Replication Management API Reference* for details about the DBMS_REPCAT. SEND_OLD_VALUES procedure and the DBMS_REPCAT.COMPARE_OLD_VALUES procedure.

6

Advanced Concepts & Architecture

This chapter describes several advanced techniques that you can use in implementing an Oracle replicated database environment:

- Using Procedural Replication
- Designing for Survivability
- Using Dynamic Ownership Conflict Avoidance
- Modifying Tables without Replicating the Modifications
- Understanding Replication Protection Mechanisms

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" on page 5-8.

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 helps you ensure that your tables remain consistent at all sites when you plan to use procedural replication.

- You must disable row-level replication within the body of the deferred procedure. See "Modifying Tables without Replicating the Modifications" on page 6-17.
- Only one replicated procedure should be executed at a time, as described in the next section, "Serialization of Transactions" on page 6-3.
- Deferred transactions should be propagated serially. For more information, see "Guidelines for Scheduled Links" on page 7-10.
- The replicated procedure must be packaged and the package cannot contain any functions. Standalone deferred procedures and standalone or packaged deferred functions are not currently supported.
- The deferred procedures must reference only locally owned data.
- The procedures should not use locally generated fields, values, or environmentally dependent SQL functions. For example, the procedure should not call SYSDATE.

- Your data ownership should be statically partitioned. That is, ownership of a row should not change between sites.
- If you have multiple master groups at a master site, and one or more master groups are quiesced, you cannot perform procedural replication on any master group at the master site. This restriction is enforced because a procedure in one master group may update objects in another master group. You can only perform procedural replication when all of the master groups on a master site are replicating data normally (that is, when none of the master groups is quiesced).

  For example, if you have a procedure named SAL_RAISE in master group A on master site DB1, you cannot execute the SAL_RAISE procedure if master group B on master site DB1 is quiesced, even if master group A is replicating normally.

Serialization of Transactions

Serial execution ensures that your data remains consistent. The replication facility propagates and executes replicated transactions one at a time. For example, assume that you have two procedures, A and B, that perform updates on local data. Now assume that you perform the following actions, in order:

1. Execute A and B locally.
2. Queue requests to execute other replicas of A and B on other nodes.
3. Commit.

The replicas of A and B on the other nodes are executed completely serially, in the same order that they were committed at the originating site. If A and B execute concurrently at the originating site, however, they may produce different results locally than they do remotely. Executing A and B serially at the originating site ensures that all sites have identical results. Propagating the transaction serially ensures that A and B are executing in serial order at the target site in all cases.

Alternatively, you could write the procedures carefully, to ensure serialization. For example, you could use SELECT... FOR UPDATE for queries to ensure serialization at the originating site and at the target site if you are using parallel propagation.

Generating Support for Replicated Procedures

You must disable row-level replication support at the start of your procedure, and then re-enable support at the end. This ensures that any updates that occur as a result of executing the procedure are not propagated to other sites. Row-level replication is enabled and disabled by calling the following procedures, respectively:

- DBMS_REPUTIL.REPLICATION_ON
- DBMS_REPUTIL.REPLICATION_OFF

See Also: "Disabling the Replication Facility" on page 6-18.

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.

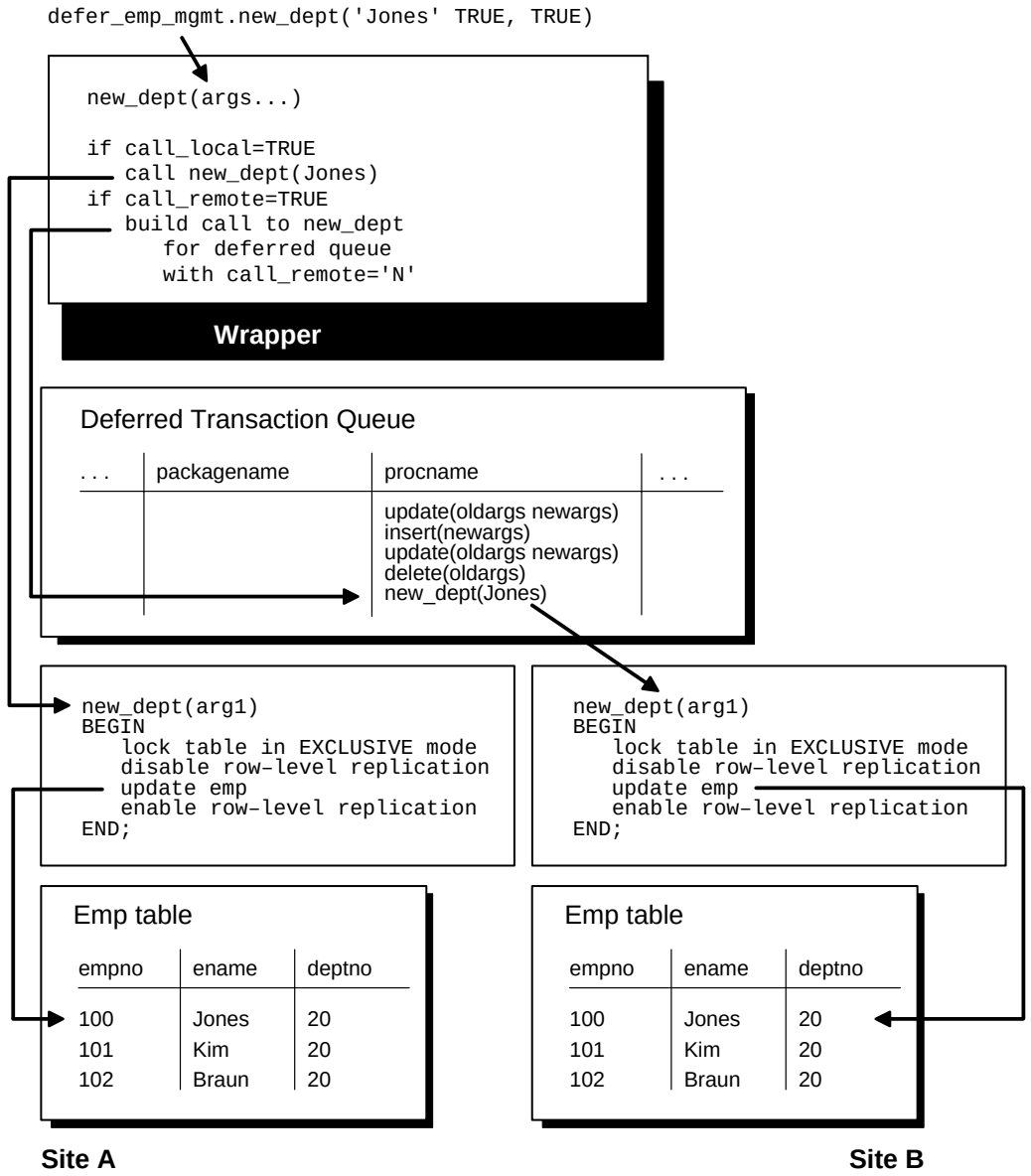
For example, suppose you create the package EMP_MGMT containing the procedure NEW_DEPT, which takes one argument, ENAME. To replicate this package to all master sites in your system, you can use Replication Manager to add the package to a master group and then generate replication support for the object. See the Replication Manager online help 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 6–1, 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 preserves the order of operations at the remote node, because both row-level and procedural replication use the same asynchronous queue.

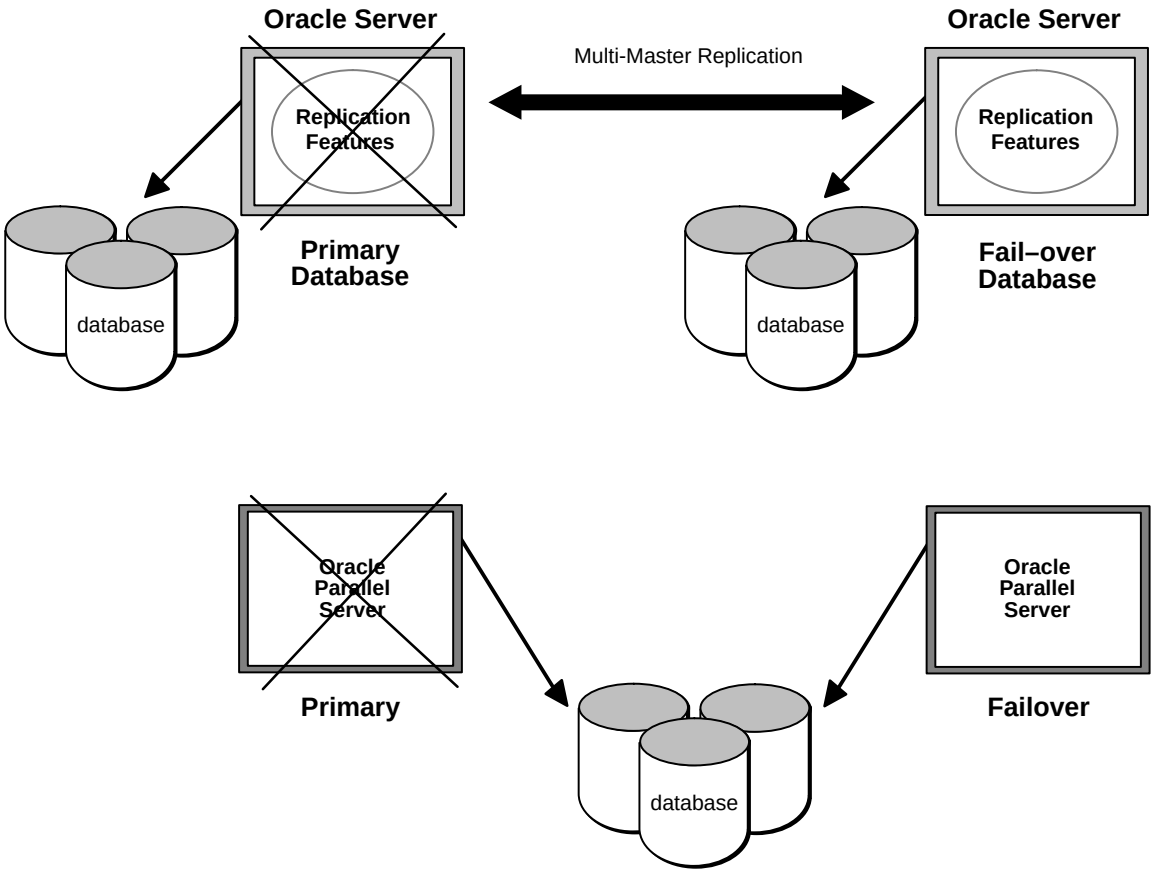
Figure 6–1 Asynchronous Procedural Replication



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 6–2, the Oracle server provides two different technologies for accomplishing survivability: the Oracle Parallel Server and the replication facility.

Figure 6–2 Survivability Methods: Replication vs. Parallel Server



Oracle Parallel Server versus 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 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. 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 replication facility for survivability, you should consider the following issues:

- The 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 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 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 failover site is used for read access only. That is, no updates are allowed at the failover site, even when the primary site fails.
- After a failure, the primary site is restored from the fail over site using export/import, or through 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 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.

You can use Net8 to configure automatic connect-time failover, which enables Net8 to fail over to a different master site if the first master site fails. You configure automatic connect-time failover in your `tnsnames.ora` file by setting the FAILOVER option to ON and specifying multiple connect descriptors.

> **See Also:** *Net8 Administrator's Guide* for more information about configuring connect-time failover.

Database Backup and Recovery in Replication Systems

Databases using replication are distributed databases. Follow the guidelines for distributed database backups outlined in the *Oracle8i Administrator's Guide* when creating backups of replicated databases. Follow the guidelines for coordinated distributed recovery in the *Oracle8i Administrator's Guide* when recovering a replication database.

If you fail to follow the coordinated distributed recovery guidelines, there is no guarantee that your 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 replication facility (for example, the DBA_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);
```

> **See Also:** The REPCAT_IMPORT_CHECK procedure in the *Oracle8i Replication Management API Reference* book for details

Using Dynamic Ownership Conflict Avoidance

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

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

Workflow

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

Take the simple example of separate sites for ordering, shipping, and billing. Here, the identifier columns are used to indicate the status of an order. The status determines which site can update the row. After a user at the ordering site has entered the order, 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 updates the status of this row to BILL, thus pushing ownership to the billing site, and so on.

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

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

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

Because the flow of work is ordered, ordering conflicts can be resolved by applying the change from the site that occurs latest in the flow of work. Any ordering conflicts can be resolved using a form of the PRIORITY conflict resolution method, where the priority value increases with each step in the work flow process.

The PRIORITY conflict resolution method successfully converges for more than one master 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.

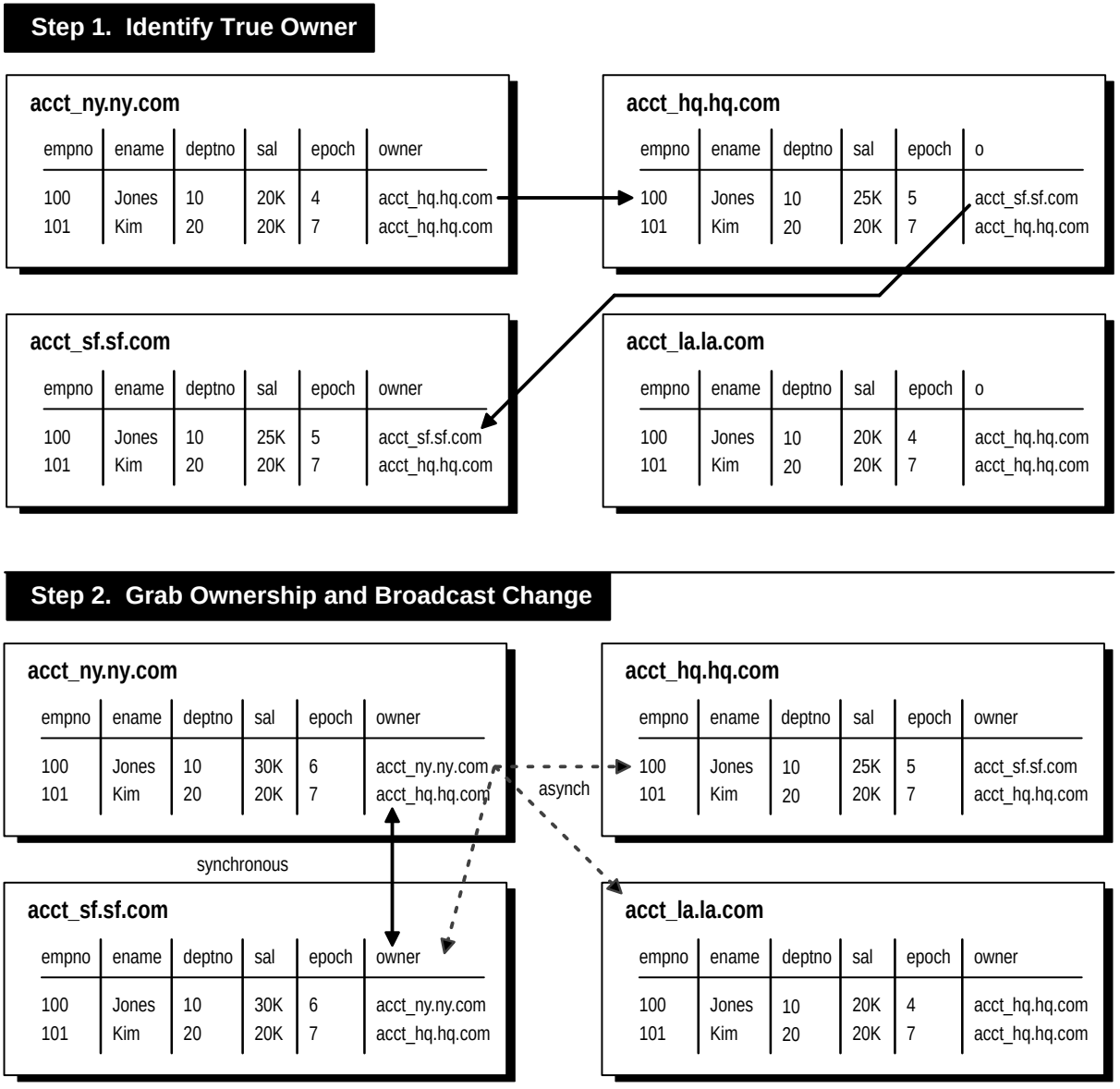
You should design your application to implement token passing for you automatically. You should not allow the owner or epoch columns to be updated outside this application.

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 6–3 illustrates how ownership of employee 100 passes from the ACCT_SF database to the ACCT_NY database.

Figure 6–3 Grabbing the Token



Locating the Owner of a Row

To obtain ownership, the ACCT_NY database uses a simple recursive algorithm to locate the owner of the row. The 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 are propagated to all other sites in the usual manner. When the SF change is propagated to NY, it is ignored because of the values of the epoch numbers, as described in the next bullet point.
4. Update the epoch number at the new owner site to be one greater than the value at the previous site. Perform this update at the new owner only, and then asynchronously propagate this update to the other master sites. Incrementing the epoch number at the new owner site prevents ordering conflicts.

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

   As another example, suppose the HQ site received the SF changes after receiving the NY changes, the HQ site would ignore the SF changes because the changes applied from the NY site would have the greater epoch number.

Applying the Change

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

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 re-execute 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 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.

> **Caution:** Turning replication on or off affects only the current session. That is, other users currently connected to the same server are not restricted from placing committed changes in the deferred transaction queue.

If you are using procedural replication, you should call REPLICATION_OFF at the start of your procedure, as shown in the following example. This ensures that the replication facility does not attempt to use row-level replication to propagate the changes that you make.

```
CREATE OR REPLACE PACKAGE update AS
  PROCEDURE update_emp(adjustment IN NUMBER);
END;
/
CREATE OR REPLACE PACKAGE BODY update AS
  PROCEDURE update_emp(adjustment IN NUMBER) IS
  BEGIN
   -- turn off row-level replication for set update
   dbms_reputil.replication_off;
   UPDATE emp . . .;
   -- re-enable replication
   dbms_reputil.replication_on;
  EXCEPTION WHEN OTHERS THEN
   . . .
   dbms_reputil.replication_on;
  END;
END;
```

Re-enabling the Replication Facility

After resolving any conflicts, or at the end of your replicated procedure, be certain to call DBMS_REPUTIL.REPLICATION_ON to resume normal replication of changes to your replicated tables or 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 only want the trigger to fire when the change is first made, and you do not want the remote trigger to fire when the change is replicated to the remote site.

You should check the value of the DBMS_REPUTIL.FROM_REMOTE package variable at the start of your trigger. The trigger should update the table only if the value of this variable is FALSE.

Alternatively, you can disable replication at the start of the trigger and re-enable it at the end of the trigger when modifying rows other than the one that caused the trigger to fire. Using this method, only the original change is replicated to the remote sites. Then the replicated trigger fires at each remote site. Any updates performed by the replicated trigger are not pushed to any other sites.

Using this approach, conflict resolution is not invoked. Therefore, you must ensure that the changes resulting from the trigger do not affect the consistency of the data.

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;
```

Understanding Replication Protection Mechanisms

In a multimaster replication environment, 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 the Replication Manager online help for more information about viewing and managing error transactions with Replication Manager.

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

- In the case of serial propagation, the replication facility uses two-phase commit.
- In the case of parallel propagation, the 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.

See Also: "Connection Qualifiers" on page 2-14.

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.

7

Planning Your Replication Environment

Before you begin to plan your replication environment, it is important to understand the replication concepts and architecture described in the previous chapters of this book. After you understand replication concepts and architecture, this chapter presents important considerations for planning a replication environment.

This chapter covers the following topics:

Considerations for Replicated Tables

The following sections discuss considerations for tables you plan to use in a replication environment.

Primary Keys

If possible, each replicated table should have a primary key. Where a primary key is not possible, each replicated table must have a set of columns that can be used as a unique identifier for each row of the table. If any of the tables that you plan to use in your replication environment do not have a primary key or a set of unique columns, alter these tables accordingly. In addition, if you plan to create any primary key snapshots based on a master table, that master table must have a primary key.

Datatype Considerations

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. You should convert LONG datatypes to LOBs in Oracle8*i*.

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 master tables will return an error message.

Initialization Parameters

Several initialization parameters are important for the operation, reliability, and performance of a replication environment. Table 7–1 lists these parameters.

See Also: *Oracle8i Reference* for more information about these initialization parameters.

Table 7–1 Initialization Parameters Important for Replication **(Page 1 of 5)**

Parameter	Values	Description
COMPATIBLE	**Default:** 8.0.0 **Range:** 8.0.0 to Current Release Number	COMPATIBLE lets you use a new release, while at the same time guaranteeing backward compatibility with an earlier release. This ability is helpful in case it becomes necessary to revert to the earlier release. This parameter specifies the release with which the Oracle server must maintain compatibility. When a snapshot site COMPATIBLE setting is less than 8.1.0, a snapshot view is created for each snapshot to maintain compatibility with releases prior to release 8.1.0. If COMPATIBLE is set to 8.1.0 or higher at a snapshot site, no view is created for snapshots.
ENQUEUE_RESOURCES	**Default:** Derived from SESSIONS parameter **Range:** 10 to unlimited	ENQUEUE_RESOURCES sets the number of resources that can be concurrently locked by the lock manager. An enqueue is a sophisticated locking mechanism that permits several concurrent processes to share known resources to varying degrees. Any object that can be used concurrently can be protected with enqueues. For example, Oracle allows varying levels of sharing on tables: two processes can lock a table in share mode or in share update mode. ENQUEUE_RESOURCES must be set to at least 1000 if the Oracle server supports Java RepAPI.
GLOBAL_NAMES	**Default:** FALSE **Range:** TRUE or FALSE	GLOBAL_NAMES specifies whether a database link is required to have the same name as the database to which it connects. GLOBAL_NAMES must be set to TRUE to use Oracle replication.

Table 7–1 Initialization Parameters Important for Replication **(Page 2 of 5)**

Parameter	Values	Description
JAVA_POOL_SIZE	**Default:** 0 **Range:** 0 to 4 GB	JAVA_POOL_SIZE specifies the size in bytes of the Java pool, from which the Java memory manager allocates most Java state during runtime execution. This memory includes the shared in-memory representation of Java method and class definitions, as well as the Java objects that are migrated to the Java session space at end-of-call. JAVA_POOL_SIZE must be set to at least 50000000 if the Oracle server supports Java RepAPI.
JOB_QUEUE_INTERVAL	**Default:** 60 (seconds) **Range:** 1 to 3600 (seconds)	JOB_QUEUE_INTERVAL specifies in seconds how frequently each SNP*n* background process of the instance wakes up. The setting for this parameter should be at least as frequent as the most frequent scheduled job.
JOB_QUEUE_PROCESSES	**Default:** 0 **Range:** 0 to 36	JOB_QUEUE_PROCESSES specifies the number of SNP*n* job queue processes per instance (SNP0, ... SNP9, SNPA, ... SNPZ). Job queue processes process requests created by DBMS_JOB. This parameter should be set to at least 1, and should be set to the same value as the maximum number of jobs that can run simultaneously. When JOB_QUEUE_PROCESSES is set to 0 at a site, you must apply administration requests manually for all groups at the site, and you must manually push and purge the deferred transaction queue.
MAX_ENABLED_ROLES	**Default:** 20 **Range:** 0 to 148	MAX_ENABLED_ROLES specifies the maximum number of database roles that users can enable, including roles contained within other roles. The actual number of roles user can enable is 2 plus the value of MAX_ENABLED_ROLES, because each user has two additional roles, PUBLIC and the user's own role. For example, if MAX_ENABLED_ROLES is set to 5, user SCOTT can have seven roles enabled: the five enabled by MAX_ENABLED_ROLES plus PUBLIC and SCOTT. MAX_ENABLED_ROLES must be set to at least 30 if the Oracle server supports Java RepAPI.

Table 7–1 Initialization Parameters Important for Replication **(Page 3 of 5)**

Parameter	Values	Description
MTS_DISPATCHERS	**Default:** None **Range:** Not Applicable	MAX_DISPATCHERS configures dispatcher processes in the multi-threaded server (MTS) architecture. The parsing software supports a name-value syntax to enable the specification of attributes in a position-independent case-insensitive manner. MAX_DISPATCHERS must be set to the following if the Oracle server supports Java RepAPI: `MTS_DISPATCHERS = "(PROTOCOL=TCP) (PRE=oracle.aurora.server.SGiopServer)"`
OPEN_CURSORS	**Default:** 50 **Range:** 1 to 4294967295 (4 GB -1)	OPEN_CURSORS specifies the maximum number of open cursors (handles to private SQL areas) a session can have at once. You can use this parameter to prevent a session from opening an excessive number of cursors. This parameter also constrains the size of the PL/SQL cursor cache, which PL/SQL uses to avoid having to reparse as statements are re-executed by a user. It is important to set the value of OPEN_CURSORS high enough to prevent your application from running out of open cursors. The number will vary from one application to another. Assuming that a session does not open the number of cursors specified by OPEN_CURSORS, there is no added overhead to setting this value higher than actually needed. OPEN_CURSORS must be set to at least 100 if the Oracle server supports Java RepAPI.
OPEN_LINKS	**Default:** 4 **Range:** 0 to 255	OPEN_LINKS specifies the maximum number of concurrent open connections to remote databases in one session. These connections include the schema objects called database links, as well as external procedures and cartridges, each of which uses a separate process. OPEN_LINKS must be set to at least 10 if the Oracle server supports Java RepAPI.

Table 7–1 Initialization Parameters Important for Replication **(Page 4 of 5)**

Parameter	Values	Description
PARALLEL_MAX_SERVERS	**Default:** Derived from the values of the following parameters: CPU_COUNT PARALLEL_AUTOMATIC_TUNING PARALLEL_ADAPTIVE_MULTI_USER **Range:** 0 to 3599	PARALLEL_MAX_SERVERS specifies the maximum number of parallel execution processes and parallel recovery processes for an instance. As demand increases, Oracle will increase the number of processes from the number created at instance startup up to this value. If you use parallel propagation, make sure the value of this parameter is set high enough to support it.
PARALLEL_MIN_SERVERS	**Default:** 0 **Range:** 0 to Value of PARALLEL_MAX_SERVERS	PARALLEL_MIN_SERVERS specifies the minimum number of parallel execution processes for the instance. This value is the number of parallel execution processes Oracle creates when the instance is started. If you use parallel propagation, make sure you have at least one process for each stream.

Table 7–1 Initialization Parameters Important for Replication **(Page 5 of 5)**

Parameter	Values	Description
PROCESSES	**Default:** Derived from PARALLEL_MAX_SERVERS **Range:** 6 to operating system dependent limit	PROCESSES specifies the maximum number of operating system user processes that can simultaneously connect to Oracle. Its value should allow for all background processes such as locks, job queue processes, and parallel execution processes.
REPLICATION_DEPENDENCY_TRACKING	**Default:** TRUE **Range:** TRUE or FALSE	REPLICATION_DEPENDENCY_TRACKING enables or disables dependency tracking for read / write operations to the database. Dependency tracking is essential for propagating changes in a replicated environment in parallel. TRUE: enables dependency tracking. FALSE: allows read / write operations to the database to run faster, but does not produce dependency information for Oracle to perform parallel propagation. Do not specify FALSE unless you are sure that your application will perform no read / write operations to the replicated tables.
SHARED_POOL_SIZE	**Default:** 16 MB (64 MB if 64 bit) **Range:** 300 KB to Operating System Dependent Limit	SHARED_POOL_SIZE specifies in bytes the size of the shared pool. The shared pool contains shared cursors, stored procedures, control structures, and other structures. Larger values improve performance in multi-user systems. Smaller values use less memory. Typically, the shared pool should be larger for an Oracle server in a replication environment than in a non-replication environment. You can monitor utilization of the shared pool by querying the view V$SGASTAT. SHARED_POOL_SIZE must be set to at least 50000000 if the Oracle server supports Java RepAPI.

Master Sites and Snapshot Sites

When you are planning your replication environment, you need to decide whether the sites participating in the replication environment will be master sites or snapshot sites. Consider the characteristics and advantages of both types of replication site when you are deciding whether a particular site in your replication environment should be a master site or a snapshot site. One replication environment can support both master sites and snapshot sites.

Table 7–2 Characteristics of Master Sites and Snapshot Sites

Master Sites	Snapshot Sites
■ Typically communicate with a small number of other master sites, and may communicate with a large number of snapshot sites	■ Communicate with one master site
■ Contain large amounts of data that are full copies of the other master sites' data	■ Contain small amounts of data that can be subsets of the master sites' data
■ Typically communicate continuously with short intervals between data propagation	■ Communicate periodically with longer intervals between bulk data transfers

Note: Master sites require either Oracle8*i* Enterprise Edition or Oracle8*i* Standard Edition. Oracle8*i* Standard Edition supports only single master site configurations. Snapshot sites require one of the following: Oracle8*i* Enterprise Edition, Oracle8*i* Standard Edition, Oracle8*i* Personal Edition, or Oracle8*i* Lite. In addition, replication must be installed on all master sites and snapshot sites.

Advantages of Master Sites

Master sites have the following advantages:

- Support for highly available data access by remote sites
- Provide better support for frequent data manipulation language (DML) changes because changes are propagated automatically and, typically, at short intervals
- Allow simultaneous DML changes and data propagation without locking tables
- Can provide failover protection

To set up a master site, you can use either the Replication Manager Setup Wizard or the replication management API.

See Also:

- The Replication Manager online help for instructions on using the Replication Manager Setup Wizard to set up a master site.
- "Setup Master Sites" in Chapter 2 of *Oracle8i Replication Management API Reference* for instructions on using the replication management API to set up a master site.
- "Designing for Survivability" on page 6-7 for information about designing your replication environment for failover protection.

Advantages of Snapshot Sites

Snapshot sites have the following advantages:

- Support disconnected computing
- Can contain a subset of its master site's data

To set up a snapshot site, you can use either the Replication Manager Setup Wizard or the replication management API.

See Also:

- The Replication Manager online help for instructions on using the Replication Manager Setup Wizard to set up a snapshot site.
- "Set Up Snapshot Sites" in Chapter 2 of *Oracle8i Replication Management API Reference* for instructions on using the replication management API to set up a snapshot site.

Guidelines for Scheduled Links

A scheduled link determines how a master site propagates its deferred transaction queue to another master site, or how a snapshot site propagates its deferred transaction queue 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.

You can configure a scheduled link to push information using serial or parallel propagation. In general, you should use parallel propagation, even if you set parallel propagation to 1.

Before creating the scheduled links for a replication system, carefully consider how you want replication to occur globally throughout the system. For example, you may choose to propagate deferred transactions at intervals, with time in between these intervals when the deferred transactions are not propagated. In this case, you must decide how often and when to schedule pushes. Alternatively, if you want to simulate real-time (or synchronous) replication, you might want to have each scheduled link constantly push a master site's deferred transaction queue to its destination.

Also, you might want to 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.

> **See Also:** "Serial and Parallel Propagation" on page 7-12 for more information about issues related to serial and parallel propagation.

Scheduling Periodic Pushes

You can schedule periodic intervals between pushes of a site's deferred transaction queue to a remote destination. Examples of periodic intervals are once an hour or once a day. To do so, 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 of 0. Then configure the interval to push the deferred transaction queue using the Next Date and Interval settings on the General page.

Scheduling Continuous Pushes

Even when using Oracle's asynchronous replication mechanisms, you can configure a scheduled link to simulate continuous, real-time replication. To do so, 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 1,200 and set Interval to a value less than the Delay Seconds setting. With this configuration, Oracle continues to push transactions that enter the deferred transaction queue for the duration of the entire interval. If the deferred transaction queue has no transactions to propagate for delay seconds, Oracle releases the resources used by the job and starts fresh when the next SNP process becomes available.

> **See Also:** "Delay Seconds" on page 2-39 for more information about setting delay seconds.

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 the Replication Manager Setup Wizard 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 a replication environment. For example, consider the following options:

- 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 is 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 Periodic Purges

You can schedule periodic purges of a site's deferred transaction queue. Examples of periodic purges are purges that occur once a day or once a week. When configuring a site's scheduled purge using the Replication Manager Setup Wizard, or the Purge Scheduling page of the Edit DB Connection property sheet, set Delay Seconds to the default value of 0. Then configure the interval to purge the deferred transaction queue using the Next Date and Interval settings.

Scheduling Continuous Purges

To configure continuous purging of a site's deferred transaction queue when using the Replication Manager Setup Wizard, or the Purge Scheduling page of the Edit DB Connection property sheet, set Delay Seconds to 500,000 and set Interval to a value less than the Delay Seconds setting.

Serial and Parallel Propagation

When you create the scheduled links for a replication environment, each link can asynchronously propagate changes to a destination using either serial or parallel propagation. Before you configure your replication environment, decide whether you want to use serial propagation 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. Typically, you should use serial propagation only when the replicated environment includes Oracle7 sites.
- 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).

  See Also: "Parallel Propagation" on page 2-36.

Deployment Templates

If you plan to include snapshot sites in your replication environment, consider using deployment templates to create the replicated objects at the snapshot sites.

> **See Also:** Chapter 4, "Deployment Templates Concepts & Architecture" for information about deployment templates.

Preparing Snapshot Sites for Instantiation of Deployment Templates

If you decide to use deployment templates, you need to prepare your snapshot sites for instantiation. If a deployment template has been designed well, little preparation is necessary at the remote snapshot site. This section describes the most common preparations that must 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 to assess which actions are necessary to prepare the remote snapshot site for instantiation:

- 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* Standard Edition, or Oracle8*i* Personal Edition release 8.1.5 or higher database? Or, does the snapshot site have an Oracle8*i* Lite release 4.0 or higher database?
- Has the remote snapshot site been set up to support snapshot replication?
- Do the schemas required by the deployment template exist at the snapshot site?
- If required database links are not part of the deployment template, do the required database links from the snapshot site to the master site exist?
- Will you use online instantiation or offline instantiation to instantiate the deployment template at the snapshot sites?
- Do the rollback segments required by the deployment template exist at the snapshot site and are they online?

The following sections provide guidance for the issues raised by each of these questions.

Network Connectivity

As with all replicated environments, network connectivity is a key component in Oracle replication. For Oracle8*i* Enterprise Edition, Oracle8*i* Standard Edition, and Oracle8*i* Personal Edition, verify that the remote snapshot site has a proper SQL*Net or Net8 connection to the target master site.

> **See Also:** *Net8 Administrator's Guide* for information about setting up an Oracle network connection with SQL*Net or Net8.

Oracle8*i* Lite snapshot sites using Java RepAPI must be able to use an Internet Inter-ORB Protocol (IIOP) connection to communicate with a master site.

> **See Also:** The Oracle8*i* Lite documentation for information about setting up snapshot clients with IIOP, and see Appendix C, "Configuring the Oracle8i Server for RepAPI" for information about setting up the server to accept these connections.

Database Version

The snapshot site must have an Oracle8*i* release 8.1.5 or higher database to instantiate a deployment template. If your snapshot site is using Oracle8*i* Lite, release 4.0 or higher must be installed. If your snapshot site does not meet the database version requirements, you need to upgrade your database at the snapshot site 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 Also:** The Replication Manager online help for instructions on setting up your snapshot site with 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 (such as 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 Also: "Set Up Snapshot Sites" in Chapter 2 of *Oracle8i Replication Management API Reference* for instructions on setting up your snapshot site with the replication management API.

Create Necessary Schema

If the deployment template that you are instantiating will create objects in multiple schemas, make sure that all of the necessary schemas have been created. Additionally, the user instantiating the deployment template must have the appropriate CREATE privileges on that schema. For example, if the deployment template will create a procedure in schema MARY and the user SCOTT is instantiating the template, SCOTT must have the CREATE ANY PROCEDURE privilege on schema MARY.

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 not required. If the database link DDL is not in the deployment template, manually create the database links to the target master site prior to instantiating the deployment template. The database links are required to populate the snapshot base tables during an online instantiation and are required for the proper maintenance of the snapshot environment.

Online or Offline Instantiation

You have the option of performing online or offline instantiation of deployment templates at snapshot sites. With online instantiation, the data in your snapshots is pulled from the master site during instantiation. With offline instantiation, the data in your snapshots is packaged in the template itself and is applied locally when you instantiate the template. In general, if your snapshots will contain a large amount of data, offline instantiation is preferred to minimize utilization of your network resources.

See Also: "Packaging and Instantiating Deployment Templates" on page 4-9 for more information about online and offline instantiation.

Create Necessary Rollback Segment

If the deployment template that you are instantiating will use specific rollback segments that do not currently exist at the remote snapshot site, create the necessary rollback segments. To see if your template objects use the default rollback segment or a specific rollback segment, query the DBA_REPCAT_TEMPLATE_OBJECTS view.

Conflict Resolution

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. If possible, plan your replication environment to avoid the possibility of conflicts. If data conflicts may occur in your replication environment, you need a mechanism to ensure that the conflict is resolved in accordance with your business rules and to ensure that the data converges correctly at all sites.

See Also: "Conflict Resolution Concepts & Architecture" for more information about avoiding conflicts and for information about the conflict resolution methods available to you if conflicts may occur.

Security

Security may be a concern in both multimaster and snapshot replication environments. You should plan your security strategy before you configure your replication environment.

See Also: Appendix A, "Security Options" in the *Oracle8i Replication Management API Reference* for information about security options in a replication environment.

8

Introduction to Replication Manager

Oracle Replication Manager provides a graphical user interface (GUI) for setting up, managing, and monitoring a replication environment. Replication Manager includes wizards that guide you through many important operations. You can use Replication Manager to manage both multimaster and snapshot replication environments. Replication Manager is integrated with Oracle Enterprise Manager.

This chapter introduces you to the features of Replication Manager. However, the primary documentation for using Replication Manager is the Replication Manager online help. This chapter covers the following topics:

- Usage Scenarios for Replication Manager
- Starting Replication Manager
- Replication Manager Interface
- Replication Manager Wizards
- Flowchart for Creating a Replicated Environment

> **See Also:** The Oracle Enterprise Manager documentation set and online help for information about using Oracle Enterprise Manager.

Usage Scenarios for Replication Manager

Using Replication Manager, you can:

- Set up master sites and snapshot sites
- Add master sites to and remove master sites from a replication environment
- Create and manage master groups
- Monitor master groups and master sites with a destination map
- Create and manage snapshot logs
- Create and manage snapshot groups
- Create and manage individual snapshots
- Create and manage refresh groups for snapshots
- Create and manage deployment templates
- Package deployment templates for offline instantiation (you must use the replication management API to package deployment templates for online instantiation)
- Configure conflict resolution methods
- Create, monitor, and manage scheduled links
- Monitor and manage administration requests
- Monitor and manage deferred transactions
- Monitor and manage error transactions
- Create, monitor, and manage local jobs

Starting Replication Manager

You can start Replication Manager using the Oracle Enterprise Manager Console or as an independent application.

Starting Replication Manager from the Oracle Enterprise Manager Console

On the Oracle Enterprise Manager Console, choose:

Tools > Extended Database Applications > Oracle Replication Manager

Starting Replication Manager as an Independent Application

The way you start Replication Manager as an independent application depends on your operating system.

On Windows platforms, use the Start menu to choose:

Programs > Oracle - *OEM_LABEL* > Extended Database Administration > Replication Manager

where *OEM_LABEL* is the label specified for Oracle Enterprise Manager during installation.

On UNIX systems, enter the following at a command prompt:

```
oemapp repmgr
```

Logging In To Replication Manager

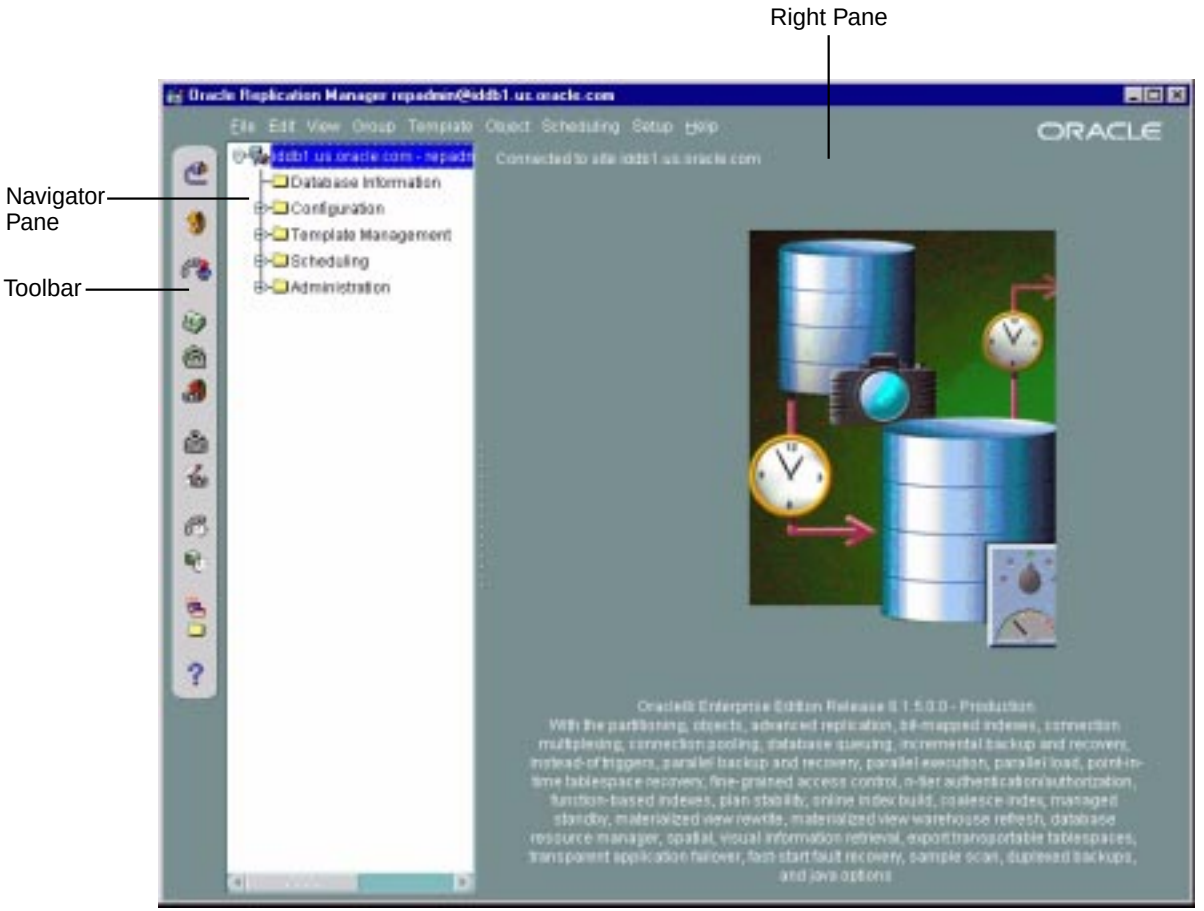
The first time you log in to Replication Manager, it should be as SYSTEM or SYS user. After you log in as SYSTEM or SYS user, use the Setup Wizard to set up your master definition site and, if you want a multimaster replication environment, your other master sites.

When setup is complete, log in to Replication Manager as the replication administrator you specified in the Setup Wizard. If you follow convention, the username of the Replication Administrator is REPADMIN. You should only log in to Replication Manager as the replication administrator, not as any other user, after setup is complete.

Replication Manager Interface

The Replication Manager interface includes two panes, a toolbar, and various menu items.

Figure 8–1 Replication Manager Interface



Panes

The Replication Manager interface has two panes: the Navigator pane and the right pane.

Navigator Pane

The Navigator pane in Replication Manager functions the same way as it does in other Oracle Enterprise Manager applications. That is, the Navigator pane lets you:

- Access all of the nodes in your replication environment (if you start Replication Manager from the Oracle Enterprise Manager Console).
- Expand and contract containers so that you can navigate to the object you want to monitor or manage. Examples of objects are master groups, snapshot groups, snapshots, snapshot logs, deployment templates, and so on.
- Right-click on an object to perform operations on the object.

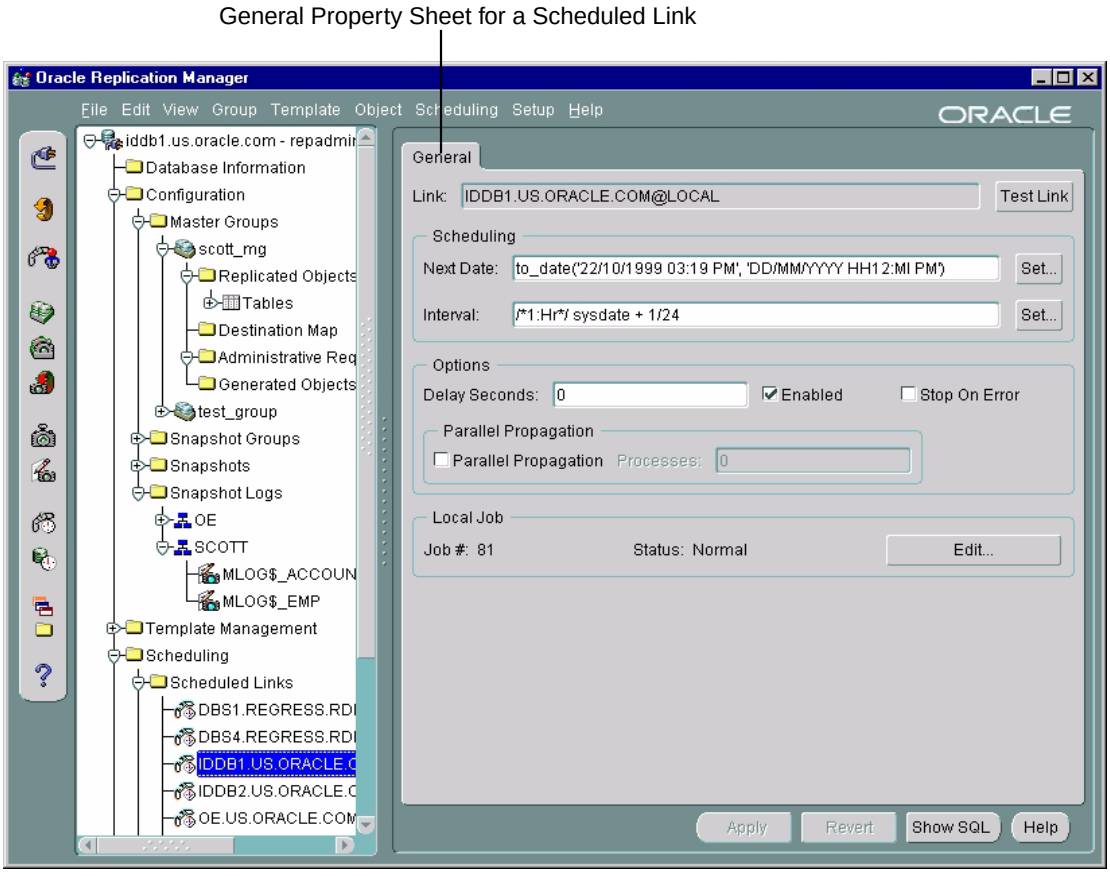
In Replication Manager, the Navigator pane has the following major containers:

Database Information	Select Database Information to view and edit the general properties of the selected database and the purge scheduling for the database's deferred transaction queue.
Configuration	Expand Configuration to manage master groups, snapshot groups, snapshots, and snapshot logs.
Template Management	Expand Template Management to manage deployment templates, authorized users of deployment templates, and sites that have instantiated deployment templates.
Scheduling	Expand Scheduling to manage scheduled links and refresh groups.
Administration	Expand Administration to manage deferred transactions, local errors, and local jobs.

Right Pane

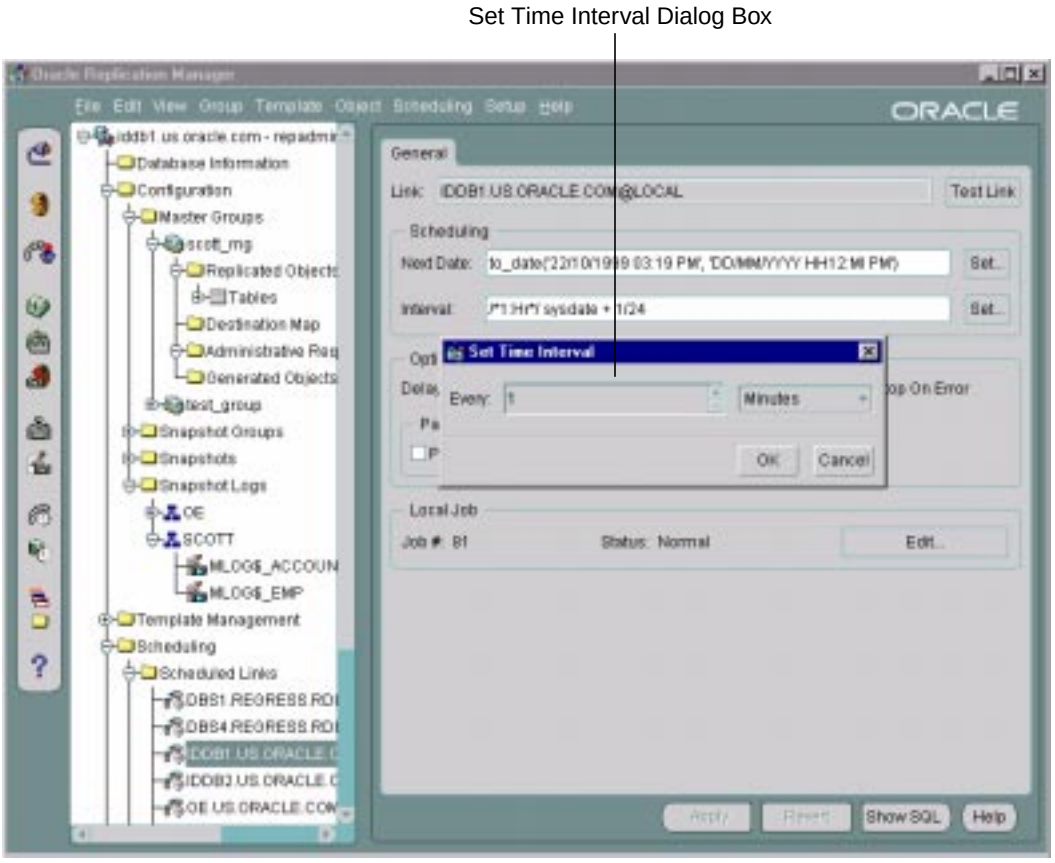
The right pane of replication manager contains both information that enables you to monitor your replication environment and property sheets that enable you to manage your replication environment. Figure 8–2 shows an example of a property sheet.

Figure 8–2 General Property Sheet for a Scheduled Link in Replication Manager



When you are working with a property sheet, you may click on a button that opens a new dialog box. For example, if you click the Set button associated with the Interval field, the Set Time Interval dialog box appears, as shown in Figure 8–3.

Figure 8–3 Set Time Interval Dialog Box in Replication Manager



You can also use the right pane of Replication Manager to monitor your replication environment. The Destination Map is an example of a monitoring tool that is available to you. The Destination Map shows the following information:

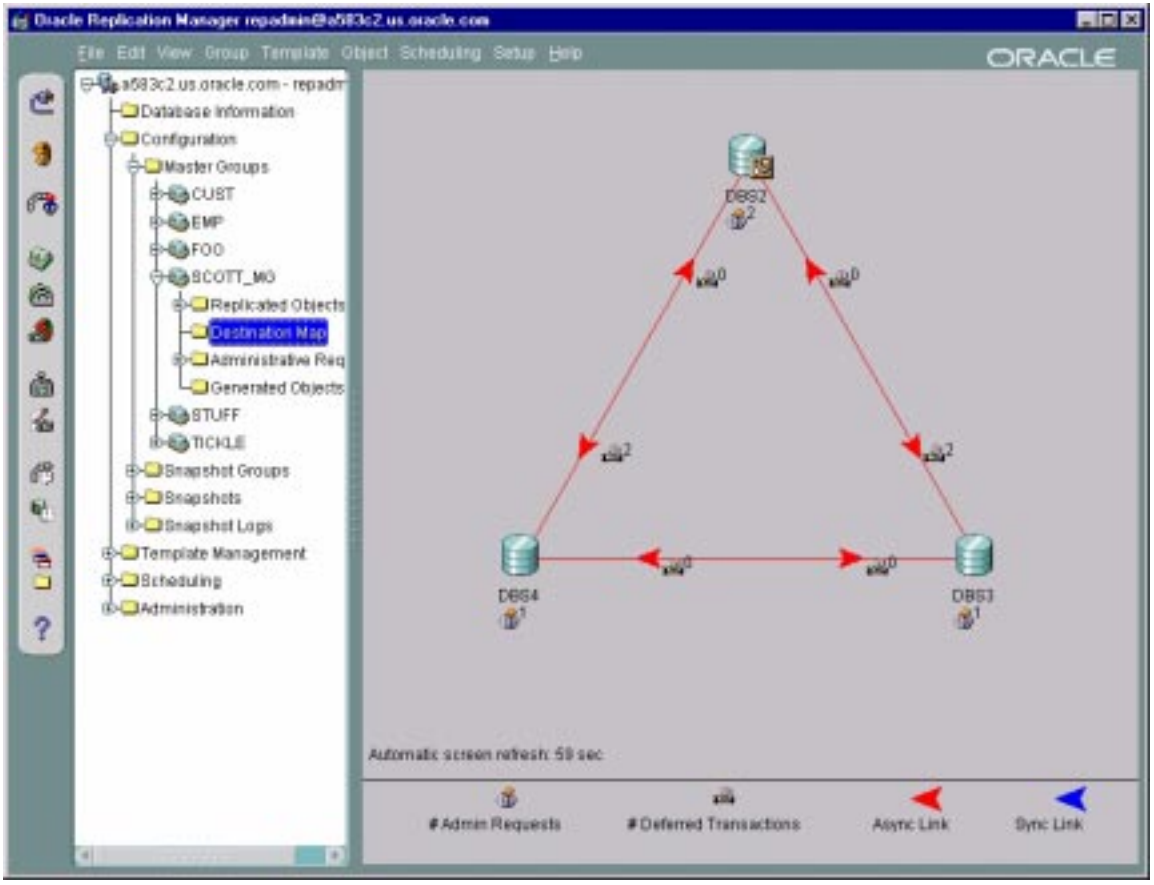
- All of the master sites in a selected master group
- The number of deferred transactions for each master site
- The number of administration requests for each master site
- Whether the master sites are using asynchronous or synchronous replication
- Whether local errors exist at a master site

For example, the Destination Map in Figure 8–4 shows the following:

- The database DBS2 has two administration request to process.
- The database DBS2 has two deferred transactions in its deferred transaction queue that will be applied at DBS3.
- The database DBS2 has two deferred transactions in its deferred transaction queue that will be applied at DBS4.
- The databases DBS3 and DBS4 both have one administration request to process.

In addition, you can right-click the Destination Map to open a menu that lets you perform actions such as refreshing the Destination Map and editing the master group shown.

Figure 8–4 Destination Map in Replication Manager



Toolbar

The Replication Manager toolbar lets you quickly perform the most common operations in a replication environment. The toolbar buttons are:

Change Database Connection 

Opens the Database Connect Information dialog box, which lets you connect to a different database.

> **See Also:** The "Changing the Database Connection" topic in the Replication Manager online help for detailed information about using the changing the database connection. To access this topic in the online help, open Replication Administration > Miscellaneous in the Help Contents.

Refresh 

Refreshes the Replication Manager interface so that you see the latest information. The Replication Manager interface does not refresh automatically, so refreshing manually may be important for monitoring various aspects of your replication environment, such as deferred transaction queues.

Setup Wizard 

Opens the Replication Manager Setup Wizard, which guides you through the process of setting up master sites and snapshot sites. When you run the Replication Manager Setup Wizard, make sure you are connected as SYS or SYSTEM user.

> **See Also:** "Setup Wizard" on page 8-18 for more information on the Setup Wizard.

Create Master Group 

Opens the Create Master Group dialog box, which lets you create a new master group. When you create a master group, the site to which you are connected becomes the master definition site for the master group.

> **See Also:** The "Flowchart for Creating a Master Group" topic in the Replication Manager online help for detailed information about using the Create Master Group dialog box to create a master group. To access this topic in the online help, open Multimaster Replication > Create Master Group in the Help Contents.

Create Snapshot Group 

Opens the Snapshot Group Wizard, which guides you through the process of setting up a snapshot group. When you create a snapshot group, make sure you are connected to the snapshot site that will contain the snapshots included in the snapshot group.

> **See Also:** "Snapshot Group Wizard" on page 8-20 for more information on the Snapshot Group Wizard.

Create Refresh Group 

Opens the Create Refresh Group dialog box, which lets you create a new refresh group. When you create a refresh group, make sure you are connected to the snapshot site that contains the snapshots that will be included in the refresh group.

> **See Also:** The "Creating a Refresh Group" topic in the Replication Manager online help for detailed information about using the Create Refresh Group dialog box to create a refresh group. To access this topic in the online help, open Snapshot Replication > Manage > Refresh Groups in the Help Contents.

Create Snapshot 

Opens the Create Snapshot dialog box, which lets you create a new snapshot. When you create a snapshot, make sure you are connected to the snapshot site where you want to create the snapshot.

> **See Also:** The "Creating a Snapshot" topic in the Replication Manager online help for detailed information about using the Create Snapshot dialog box to create a snapshot. To access this topic in the online help, open Snapshot Replication > Manage > Snapshots in the Help Contents.

Create Snapshot Log 

Opens the Create Snapshot Log dialog box, which lets you create a new snapshot log for a master table. When you create a snapshot log, make sure you are connected to the master site that contains the master table for which you want to create a snapshot log.

> **See Also:** The "Creating Snapshot Logs" topic in the Replication Manager online help for detailed information about using the Create Snapshot Log dialog box to create a snapshot log. To access this topic in the online help, open Snapshot Replication in the Help Contents.

Create Scheduled Link 

Opens the Create Scheduled Link dialog box, which lets you create a new scheduled link.

> **See Also:** The "Creating a Scheduled Link" topic in the Replication Manager online help for detailed information about using the Create Scheduled Link dialog box to create a scheduled link. To access this topic in the online help, open Replication Administration > Scheduled Links in the Help Contents.

Create Local Job 

Opens the Create Local Job dialog box, which lets you create a new local job.

> **See Also:** The "Creating a Local Job" topic in the Replication Manager online help for detailed information about using the Create Local Job dialog box to create a local job. To access this topic in the online help, open Replication Administration > Local Jobs in the Help Contents.

Create Template 

Opens the Deployment Template Wizard, which guides you through the process of creating a new deployment template. When you create a deployment template, make sure you are connected to a master site that contains the objects that will be included in the deployment template.

> **See Also:** "Deployment Template Wizard" on page 8-22 for more information on the Deployment Template Wizard.

Offline Instantiation File Generation 

Opens the Offline Instantiation Wizard, which guides you through the process of packaging the deployment template and generating offline instantiation files. You can generate offline instantiation files for Oracle8*i* Lite and for Oracle server (which includes Oracle8*i* Enterprise Edition, Oracle8*i* Standard Edition, and Oracle8*i* Personal Edition). When you generate offline instantiation files, make sure you are connected to the master site that contains the deployment template.

> **See Also:** "Offline Instantiation Wizard" on page 8-24 for more information on the Offline Instantiation Wizard.

Note: To generate online instantiation files, you must use the replication management API. See *Oracle8i Replication Management API Reference* for information about generating online instantiation files.

Help 

The Help button opens the Replication Manager online help.

Menus

You use Replication Manager's menus to perform actions, such as refreshing the Replication Manager interface, and to open wizards and dialog boxes. The following list describes the items available under each menu.

File Menu

The items in the File menu depend on whether you connect to Replication Manager using the Oracle Enterprise Manager Console or independently.

The File menu has the following menu items if you connected to Replication Manager using the Oracle Enterprise Manager Console:

Connect	Lets you connect to the selected database in the Navigator pane.
Disconnect	Lets you disconnect from the selected database in the Navigator pane.
Exit	Lets you exit Replication Manager.

The File menu has the following menu items if you connected to Replication Manager independently:

Change Database Connection	Lets you change your database connection. When you connect independently, you can connect to only one database at a time.
Exit	Lets you exit Replication Manager.

Edit Menu

The Edit menu has the following menu item:

Preferences Lets you edit the default settings for master groups, snapshots, refresh groups, scheduled links, and local jobs. The settings you specify as defaults are the settings used when you create a new object.

For example, when you create a new master group, Copy Row Data is one of the options. If you enable Copy Row Data as the default, Copy Row Data is enabled every time you create a new master group. However, you still can disable the option during master group creation if necessary.

Note: Changes to the Preferences menu only apply to a single Replication Manager session. If you close Replication Manager and then restart it, the items on the Preferences menu are returned to the defaults.

View Menu

The View menu has the following menu item:

Refresh Lets you refresh the Replication Manager interface. It performs the same action as the Refresh button on the toolbar.

Group Menu

The Group menu has the following submenus and menu items:

Master Group The Master Group submenu contains the following menu items:

Create: Opens the Create Master Group dialog box, which lets you create a new master group.

Add Object: Opens the Add Object(s) to Group dialog box, which lets you add objects to the selected master group.

Remove: Lets you remove the selected master group.

Snapshot Group	The Snapshot Group submenu contains the following menu items: **Create:** Opens the Snapshot Group Wizard, which lets you create a new snapshot group. **Add Object:** Opens the Snapshot Group Edit Wizard, which lets you add objects to the selected snapshot group. **Remove:** Lets you remove the selected snapshot group.

Template Menu

The Template menu has the following menu items and submenus:

Create	Opens the Deployment Template Wizard, which guides you through the process of creating a deployment template.
Copy	Opens the Copy Template Wizard, which lets you copy a deployment template to the same master site or to different master sites.
Compare	Opens the Compare Deployment Templates dialog box, which lets you compare two deployment templates.
Add	The Add submenu contains the following menu items: **Object:** If the selected deployment template has not yet been instantiated, lets you add objects to it. **Parameter:** If the selected deployment template has not yet been instantiated, lets you add parameters to it. **Authorized User:** Lets you add authorized users to private deployment templates.
Instantiate	Opens the Offline Instantiation Wizard, which guides you through the process of generating offline instantiation files for a deployment template.
Remove	Lets you remove the selected deployment template. Removing a deployment template does not affect any remote snapshot sites that have already instantiated the template.
Drop Instantiated Site	If an instantiated snapshot site is selected, lets you drop the instantiated site. To select an instantiated site, expand Template > *template_name* > Sites in the Navigator pane, where *template_name* is the name of the deployment template.

Object Menu

The Object menu has the following menu items:

Create Snapshot	Opens the Create Snapshot dialog box, which lets you create a snapshot at a snapshot site.
Create Snapshot Log	Opens the Create Snapshot Log dialog box, which lets you create a snapshot log on a master table at a master site.
Remove	Lets you remove the selected snapshot or snapshot log.

Scheduling Menu

The Scheduling menu has the following menu items and submenus:

Create Scheduled Link	Opens the Create Scheduled Link dialog box, which lets you create a new scheduled link.
Create Local Job	Opens the Create Local Job dialog box, which lets you create a new local job.
Refresh Group	The Add submenu contains the following menu items: **Create:** Opens the Create Refresh Group dialog box, which lets you create a new refresh group. **Add Object:** Opens the Add Snapshot(s) to Refresh Group dialog box, which lets you add snapshots to the selected refresh group. **Remove:** Lets you remove the selected refresh group.
Remove	Lets you remove the selected scheduled link or local job.

Setup Menu

The Setup menu has the following menu items:

Master Sites	Opens the Replication Manager Setup Wizard, which guides you through the process of setting up master sites.
Snapshot Sites	Opens the Replication Manager Setup Wizard, which guides you through the process of setting up snapshot sites.

Help Menu

The Help menu has the following menu items:

Contents	Opens Replication Manager online help.
About Replication Manager	Opens a window that displays the Replication Manager version number and copyright information.

Replication Manager Wizards

The Replication Manager wizards provide step-by-step guidance for tasks that require many steps. The wizards simplify complex tasks by guiding you through the task in manageable steps. The following sections describe the Replication Manager wizards:

- Setup Wizard
- Snapshot Group Wizard
- Deployment Template Wizard
- Offline Instantiation Wizard
- Copy Template Wizard

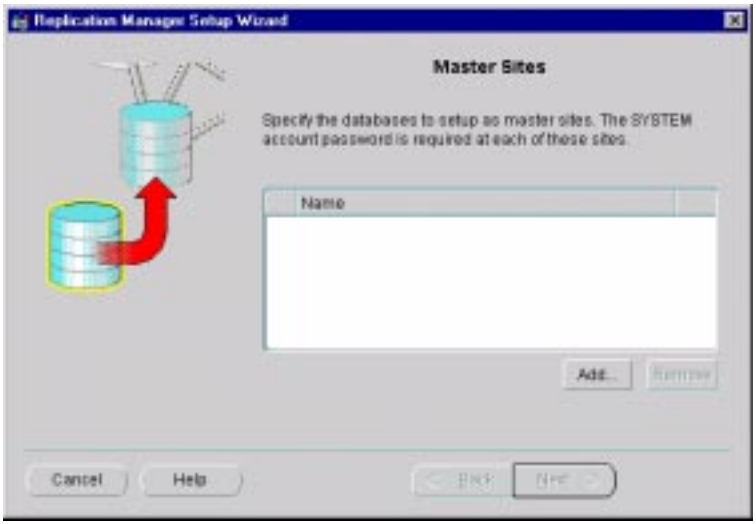
Setup Wizard

The Setup Wizard guides you through setting up master sites and snapshot sites for replication. Preparing sites for replication is a simple process using the Setup Wizard. At each site that you specify, the Setup Wizard performs the following steps:

- Creates a database account to serve as a replication administrator. By default, the Setup Wizard creates this account to serve also as the replication propagator and receiver. The default username for the replication administrator at a master site is REPADMIN, and the default username at a snapshot site is SNAPADMIN, but you can change the username if you wish.
- Grants the necessary privileges to the replication administrator account.
- Creates database links to correspond to new replication administrator accounts at each site.
- For master sites, schedules a job to push changes from the master site to each other master site.
- Schedules a job to purge the deferred transaction queue of completed transactions for all sites in the system.

The Setup Wizard is slightly different for master sites than for snapshot sites. To open the Setup Wizard to set up master sites, choose Setup > Master Sites from the menu bar.

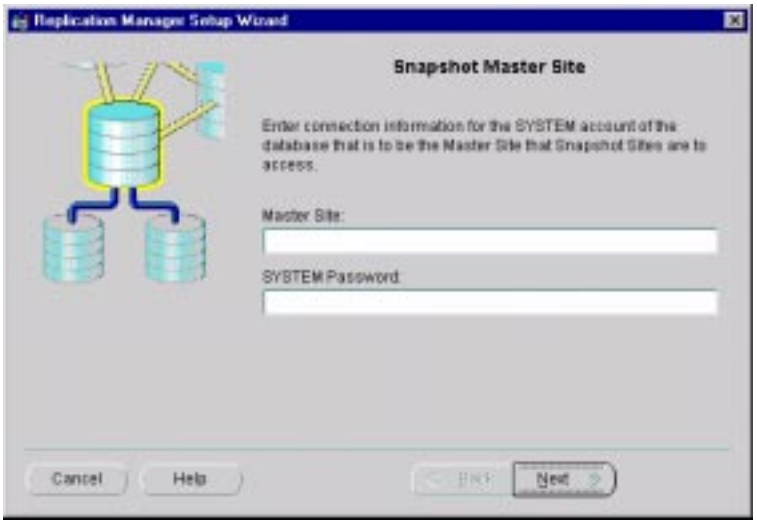
Figure 8–5 Opening Screen of Setup Wizard for Master Sites



See Also: The "Setup Master Site: Overview" topic in the Replication Manager online help for detailed information about using the Setup Wizard to set up a master site. To access this topic in the online help, open Setup Replication Sites > Master Site in the Help Contents.

To open the Setup Wizard to set up snapshot sites, choose Setup > Snapshot Sites from the menu bar.

Figure 8–6 Opening Screen of Setup Wizard for Snapshot Sites



See Also: The "Set Up Snapshot Site: Overview" topic in the Replication Manager online help for detailed information about using the Setup Wizard to set up a snapshot site. To access this topic in the online help, open Setup Replication Sites > Snapshot Site in the Help Contents.

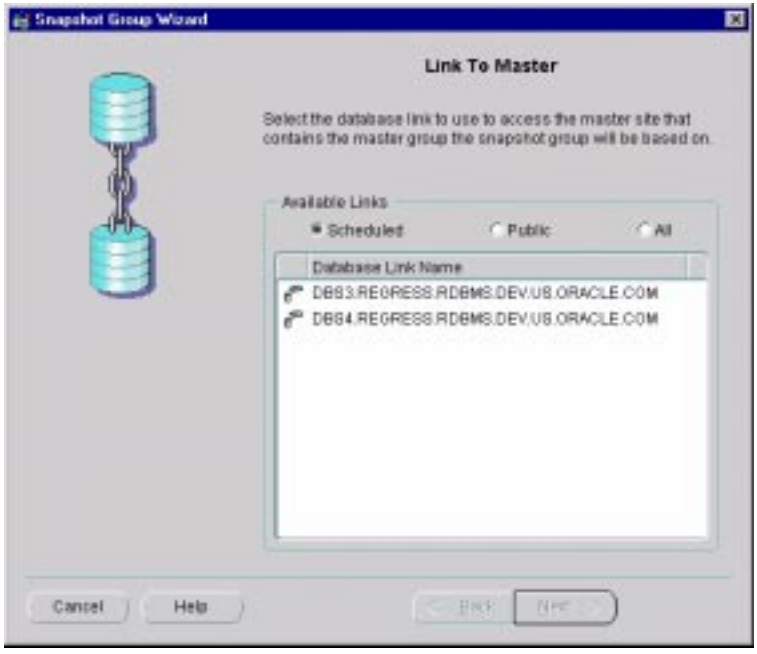
See Also: Chapter 2, "Master Concepts & Architecture" for more information about master sites, and see Chapter 3, "Snapshot Concepts & Architecture" for more information about snapshot sites.

Snapshot Group Wizard

The Snapshot Group Wizard guides you through creating a group of snapshots based on a master group. Each snapshot can be a partial or complete copy of the master table from its source master group. Snapshot groups are located at remote snapshot sites and are based on a single, target master group at a master site.

Run the Snapshot Group Wizard at the snapshot site where you want to create the snapshot group. To open the Snapshot Group Wizard, click the Create Snapshot Group button on the Replication Manager toolbar.

Figure 8–7 Opening Screen of the Snapshot Group Wizard



See Also: The "Create Snapshot Group: Overview" topic in the Replication Manager online help for detailed information about using the Snapshot Group Wizard to create a snapshot group. To access this topic in the online help, open Setup Snapshot Replication > Create in the Help Contents.

See Also: "Snapshot Groups" on page 3-23 for more information about snapshot groups.

Deployment Template Wizard

Deployment templates simplify the task of deploying and maintaining many remote snapshot sites. Using deployment templates, you can define a collection of snapshot definitions at a master site and use parameters in these definitions to customize the snapshots for individual users or types of users.

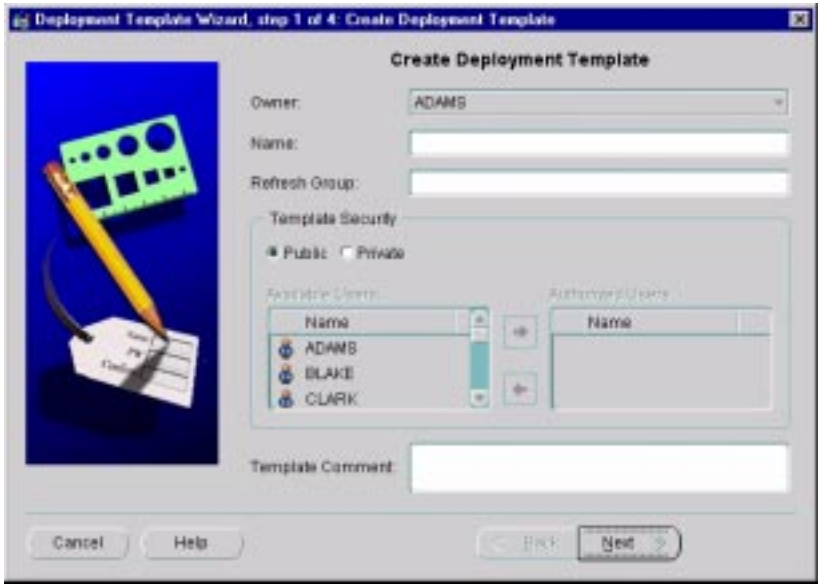
For example, you might create one template for the sales force and another template for field service representatives. In this case, a parameter value might be the sales territory or the customer support level. When a user instantiates a template, the appropriate snapshots are created and populated at the remote site.

The Deployment Template Wizard guides you through creating a deployment template. Separate screens in the Deployment Template Wizard let you:

- Name the deployment template and specify whether it is public or private. If it is private, you can specify authorized users.
- Add objects to the deployment template.
- Specify parameters for the deployment template.

Run the Deployment Template Wizard from the master site where you want to create the deployment template. To open the deployment template wizard, click the Create Template button in the Replication Manager toolbar.

Figure 8–8 Opening Screen of the Deployment Template Wizard



See Also: The "Overview of Creating a Deployment Template" topic in the Replication Manager online help for detailed information about using the Deployment Template Wizard to create a deployment template. To access this topic in the online help, open Deployment Templates > Create in the Help Contents.

See Also: Chapter 4, "Deployment Templates Concepts & Architecture" for more information about deployment templates.

Offline Instantiation Wizard

Offline instantiation allows end users to use a generated file to instantiate a template without being connected to the master site through a network. Offline instantiation allows you to write all of the necessary DDL and data to a file that you then transfer and run at your snapshot site. This solution is best suited for laptop users with low-speed WAN connections, or other cases where connectivity is unstable or slow.

This solution is best suited for laptop users with low-speed WAN connections, which could make online instantiation difficult.

The Offline Instantiation Wizard guides you through packaging a deployment template for offline instantiation. The Offline Instantiation Wizard generates offline instantiation files that you use to build snapshots and other objects at your snapshot sites.

Run the Offline Instantiation Wizard at the master site that contains the template for which you want to generate the offline instantiation files. To run the Offline Instantiation Wizard, click the Offline Instantiation File Generation button in the Replication Manager toolbar.

Figure 8–9 Opening Screen of the Offline Instantiation Wizard

Note: To generate online instantiation files, you must use the replication management API. See *Oracle8i Replication Management API Reference* for information about generating online instantiation files.

See Also: The "Package for Offline Instantiation: Overview" topic in the Replication Manager online help for detailed information about using the Offline Instantiation Wizard to package a deployment template for offline instantiation. To access this topic in the online help, open Deployment Templates > Packaging and Instantiation in the Help Contents.

See Also: "Packaging and Instantiation Process" on page 4-16 for more information on packaging and instantiating deployment templates.

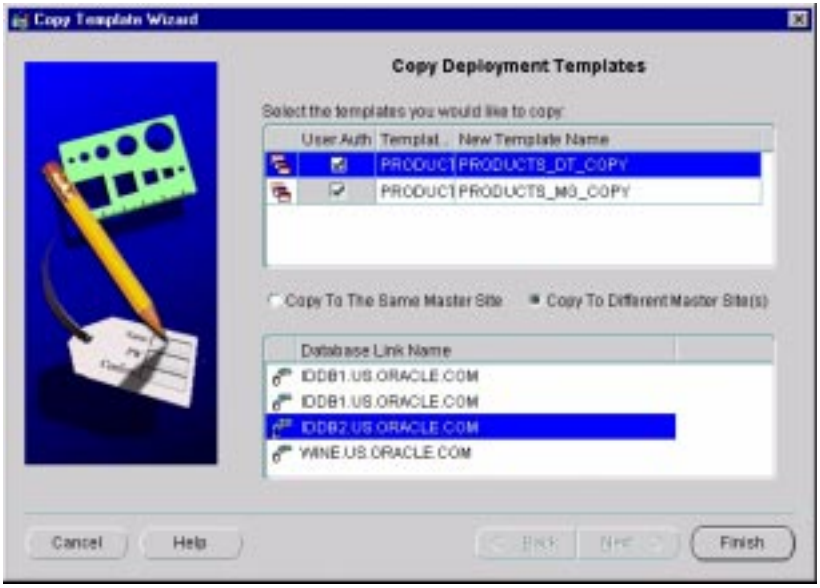
Copy Template Wizard

The Copy Template Wizard guides you through copying a deployment template. You may need to copy them to multiple master sites for various reasons. For example, you may want to:

- Distribute Network Load: Before allowing users to instantiate the template, you need to locate the template at the master site of the target snapshot sites. If you have a large network, you may want to copy the template definition to multiple master sites, thereby distributing the network load of multiple snapshot sites.
- Make Changes to a Template: After building a template, you may need to create another template that has many of the same characteristics of the first template; instead of building an entirely new template, copy the template and modify it as necessary.

Run the Copy Template Wizard from the master site that contains the deployment template. To open the Copy Template Wizard, choose Template > Copy from the menu bar.

Figure 8–10 Copy Template Wizard

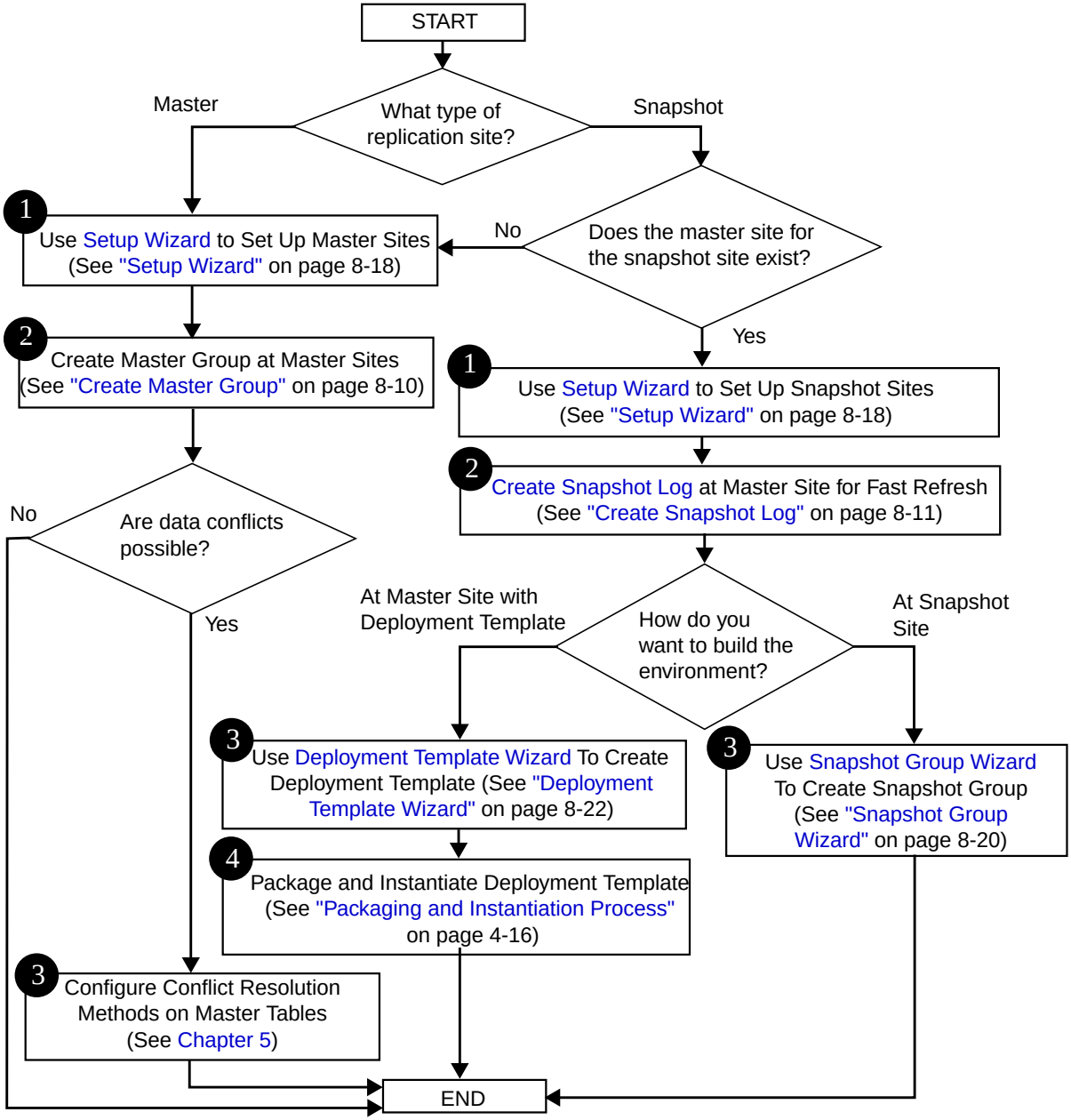


See Also: The "Copy Deployment Template" topic in the Replication Manager online help for detailed information about using the Copy Template Wizard to copy a deployment template. To access this topic in the online help, open Deployment Templates > Manage in the Help Contents.

Flowchart for Creating a Replicated Environment

The flowchart in Figure 8–11 displays the major decisions and tasks that are involved when you create a replication environment using Replication Manager. The flowchart shows the major decisions and steps for building both multimaster and snapshot environments. Each task in the flowchart includes a cross reference to a section in this book that provides more information about the task. Detailed instructions about for completing these tasks are in the Replication Manager online help.

Figure 8–11 Create Replicated Environment Using Replication Manager



A

New Features

This appendix briefly describes the new replication features of Oracle8*i* and provides pointers to where you can get additional information. This chapter also retains Oracle8 (release 8.0) new features to help those users migrating from Oracle7 to Oracle8*i* (release 8.1).

New features include:

Oracle8*i*:

- Performance Improvements
- Improved Mass Deployment Support
- Improved Security
- Replication Manager
- Improved Oracle8i Lite Integration

Oracle8:

- Performance Enhancements
- Data Subsetting Based on Subqueries
- Enhanced, System-Based Security Model
- New Replication Manager Features

Oracle8*i* New Replication Features

The Oracle8*i* (release 8.1) replication features and enhancements described below comprise the overall effort to optimize replication performance and make snapshot environment distribution and security more effective.

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.

See Also: Chapter 2, "Master Concepts & Architecture".

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.

See Also: Chapter 3, "Snapshot Concepts & Architecture".

Improved Mass Deployment Support

With Oracle8*i*, Oracle has improved replication support for the growing market of front office applications, in particular mass deployment.

Parameterized Snapshot Deployment Templates

Deployment templates 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, and then deploy the 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 Template Wizard guides the DBA through the selection of schema objects to add to the template, the selection of parameters, and defining authorizations. Deployment templates can be used to create snapshot environments at the following types of snapshot sites: Oracle8*i* Enterprise Edition, Oracle8*i* Standard Edition, Oracle8*i* Personal Edition, and Oracle8*i* Lite.

> **See Also:** Chapter 4, "Deployment Templates Concepts & Architecture".

Column Level Snapshot Subsetting

Updateable snapshots can now be subsetted horizontally (selected rows) or vertically (selected columns). Previous release allow 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.

> **See Also:**
>
> - "Data Subsetting with Snapshots" on page 3-8.
> - "Vertical Partitioning" on page 4-22.
> - "Horizontal Partitioning with Assignment Tables" on page 4-24.

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 to online instantiation is that the data will be current at instantiation time. However, this is at the cost of requiring a live connection, possibly of long duration.

Offline instantiation allows the DBA to package the deployment templates and required data onto some type of storage media (tape, CD-ROM, and so on) 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 are instantiating the target template.

> **See Also:** "Packaging and Instantiating Deployment Templates" on page 4-9.

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 snapshots. 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.

> **See Also:** Appendix A, "Security Options" in *Oracle8i Replication Management API Reference*.

Replication Manager

A number of improvements have been made to the Oracle Replication Manager. A major 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 Windows platforms. There is also a replication class of events in OEM which, for example, can be used to monitor errors or delinquent snapshot refreshes.

Replication Manager has the following new wizards in Oracle8*i*:

- Deployment Template Wizard for creating deployment templates
- Offline Instantiation Wizard for packaging a deployment template for offline instantiation

> **See Also:** Chapter 8, "Introduction to Replication Manager" and the Replication Manager online help.

Improved Oracle8*i* Lite Integration

The integration between the Oracle8*i* server and Oracle8*i* Lite has been improved to provide better performance and increased functionality. All of the replication changes previously described are supported, and Oracle8*i* Lite users with laptops at remote sites will especially benefit from reduced connection time.

Java RepAPI

Java RepAPI is a runtime library that enables clients to replicate data with Oracle servers. Java RepAPI provides replication services to thin client databases, such as Oracle8*i* Lite and those running front-office automation applications. With the new deployment template feature in Oracle8*i* as its foundation, Java RepAPI provides support for LOBs and vertical subsetting of updateable snapshots, using the CORBA Internet Inter-ORB Protocol (IIOP) for communication. Oracle8*i* supports instantiation of Oracle8*i* Lite sites while disconnected from the server. Java RepAPI is available only with Oracle release 8.1.6 and higher.

> **See Also:** Appendix C, "Configuring the Oracle8i Server for RepAPI" for more information.

Oracle8 New Replication Features

The Oracle8 (release 8.0) replication 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. Manageability and security were also enhanced for Oracle8. All of the features described in the following sections also apply to Oracle8*i*.

Performance Enhancements

Oracle8 provides significant performance improvements based on the following new features.

Parallel Propagation of Deferred Transactions

Oracle8 dramatically improves throughput performance by parallelizing the propagation of a replication transaction stream while maintaining consistency and transaction dependencies.

> **See Also:** "Guidelines for Scheduled Links" on page 7-10.

Internalized Replication Triggers

Oracle8 internalized replication triggers that were external in past releases are now internalized. Internal triggers:

- Improve response time performance
- Reduce processing overhead
- Require less administration

See Also: Chapter 2, "Master Concepts & Architecture".

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
- Primary key values

This feature is especially important for performance when replicating LOBs.

See Also: "Min Communication" on page 2-38.

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 sales force automation and branch automation.

See Also: "Data Subsetting with Snapshots" on page 3-8.

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)

> **See Also:** The Replication Manager online help for information on replicating object definitions to master sites and "Datatype Considerations for Snapshots" on page 3-22.

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 process updates while other master groups are quiesced.

> **See Also:** "Replication Modes of Operation" on page 2-20.

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.

> **See Also:** "Available Snapshots" on page 3-4. Also, see Appendix G, "Migration and Compatibility for Replication Environments" in *Oracle8i Migration* for information about using ROWID snapshots in version 8.

Snapshot Registration at Master Sites

Oracle8 automatically registers information about a snapshot at its associated master site. This facilitates monitoring and distributed administration.

> **See Also:** "Snapshot Registration at a Master Site" on page 3-16.

Reorganizing Tables With Capability of Fast Refresh

Oracle8 provides utilities to enable you to reorganize master tables while preserving the consistency of master snapshot logs.

> **See Also:** *Oracle8i Replication Management API Reference* for information about reorganizing master tables.

Support for Offline Instantiation Using Export/Import

Offline instantiation of schemas and database using Export/Import is now more automatic.

> **See Also:** The "Performing an Offline Instantiation Using Export/Import" section in Chapter 7 of *Oracle8i Replication Management API Reference.*

Deferred Constraints for Updateable Snapshots

Updateable snapshots now support declarative referential and uniqueness constraints.

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

The Oracle8 system-based security model:

- Improves consistency because a single system-level model operates the same way in both synchronous and asynchronous environments
- Improves reliability because transactions are less likely to fail due to the lack of privileges at the receiving sites
- Simplifies links because a single user can act as REPADMIN user and REPSYS user
- Eliminates the need for a “user-level” model
- Allows one or more snapshot owners to perform snapshot refresh

> **See Also:** Appendix A, "Security Options" in *Oracle8i Replication Management API Reference.*

New Replication Manager Features

Replication Manager now includes several wizards to help you configure your system quickly.

- Setup Wizard for set up of multimaster configurations, including account, schema, and link creation
- Setup Wizard for set up of snapshot site configurations, including account, schema, and link creation
- Snapshot Group Wizard for set up of snapshot groups.

Oracle Replication Manager includes support for most new features of Oracle8 replication, including:

- Automatic recognition of each site's database version and dynamic configuration to manage both release 8.x and release 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
- Support for the validation of a master group at all master sites

> **See Also:** Chapter 8, "Introduction to Replication Manager" and the Replication Manager online help.

B

Troubleshooting Replication Problems

This appendix contain troubleshooting guidelines for managing a replication environment. This appendix covers the following topics:

- Diagnosing Problems with Database Links
- Diagnosing Problems with Master Sites
- Diagnosing Problems with the Deferred Transaction Queue
- Diagnosing Problems with Snapshots

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, SQL*Plus, 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 dbs1.world@myethernet CONNECT TO repadmin
  IDENTIFIED BY secret USING 'connect_string_myethernet'
```

All the sites, whether masters or snapshots, associated with dbs1.world@myethernet must include myethernet as the connection qualifier.

Diagnosing Problems with Master Sites

There are a number problems that might arise in a multimaster 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.

Note: 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.
- If a scheduled link's interval has passed and corresponding deferred transactions have not been pushed, check the corresponding job for the link.
- Even after propagating a deferred transaction to a destination, it might not execute because of an error.

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. EXECUTE privileges for most of the packages involved with 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.

Problems With ON DELETE CASCADE and Integrity Constraints

ON DELETE CASCADE is not supported for replicated tables with a configuration similar to the following example. This example assumes there are three tables named A, B, and C.

- Table B contains a record dependent on a record in Table A because of a foreign key constraint.
- Table C contains a record dependent on a record in Table B because of a foreign key constraint.
- Table C also contains a record dependent on a record in Table A because of a foreign key constraint.

If you use ON DELETE CASCADE in such a configuration, the deletes may not be propagated in the correct order when you delete a record. In these cases, the following error is returned:

```
ORA-02292: integrity constraint (SCOTT.FK_CB) violated - child record found
```

If you encounter this error, create triggers that are aware of replication to perform this functionality, instead of ON DELETE CASCADE.

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 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 of problems that might happen with snapshot sites in a 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 updateable snapshot, check that the associated master table has a snapshot log.
- Make sure that you have the necessary privileges to create the object. For a snapshot, you need SELECT privilege on the master table and its snapshot log. See "Privileges" on page 3-33 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.
- If you are trying to create a fast refresh primary key or subquery snapshot, then make sure that the snapshot log on the master table logs primary keys.
- If you are trying to create a fast refresh rowid snapshot, then make sure that the snapshot log on the master table logs rowids.
- Check if the snapshot log has the required filter columns for subquery snapshots.
- Check if the snapshot log exists for all tables that are involved in a fast refresh snapshot. If the snapshot contains a subquery, each table referenced in the subquery should have a snapshot log.

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.

> **See Also:** The name of the snapshot trace file is of the form SNP*n*, where *n* is operating system-specific. See the Oracle documentation for your operating system 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 snapshot log or 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 diagnose the problem with job queues.
- Check if their are job queue process running. Check the JOB_QUEUE_ PROCESSES initialization parameter, query the DBA_JOBS_RUNNING view, and use your operating system to check if the SNP background processes are still running.
- 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 is 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 *Oracle8i Replication Management API Reference* for information about managing snapshot logs.

- Snapshots in the same refresh groups 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 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.
- 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 have incorrect references to master table ROWIDs.
- If while refreshing you see an ORA-942 (table or view does not exist), then check your database links and make sure you still have the required privileges on the master table and the snapshot log.
- If a fast refresh was succeeding but then fails, then check the following:
  - If the snapshot log was truncated, purged, or dropped
  - If you still have the required privileges on the snapshot log
- If a force refresh takes an inordinately long time, then check if the snapshot log used by the refresh has been dropped.
- If the snapshot was created with BUILD DEFERRED, and its first fast refresh fails, make sure a previous complete refresh was done successfully before checking for other problems.

See Also: *Oracle8i Replication Management API Reference* for information about managing snapshot logs.

C

Configuring the Oracle8*i* Server for RepAPI

This appendix describes configuring an Oracle8*i* server for access by RepAPI clients. This appendix covers the following topics:

What Is RepAPI?

RepAPI is a runtime library that enables clients to replicate data with Oracle servers. RepAPI provides replication services to thin client databases, such as Oracle8*i* Lite and those running front-office automation applications. The RepAPI server runtime library is installed in the *ORACLE_HOME*/rdbms/jlib directory. For information about the location of the installed RepAPI files on Oracle8*i* Lite clients, see the Oracle8*i* Lite documentation.

In past releases, RepAPI was written in the C programming language, but it has been completely rewritten in Java for Oracle8*i*. This Java-based RepAPI facility requires fewer resources and is easier to use than its C-language predecessor. Also, in Oracle8*i*, round-trip network communication between the client and the server is kept to a minimum by reducing the amount of metadata maintained on the client and by keeping the SQL snapshot queries entirely on the master side.

Administration and configuration of client sites is simplified by utilizing the new Oracle8*i* deployment templates, which are based on refresh groups, as the sole mechanism for manipulating snapshots. In addition, in Oracle8*i* the RepAPI facility supports LOBs as well as offline and online instantiation of deployment templates.

Offline instantiation involves two parts: packaging and instantiation. The database administrator (DBA) for the master site typically packages a deployment template for offline instantiation at a particular site. Packaging results in an offline instantiation file for each client site. When the DBA completes this packaging, the related snapshot logs begin logging updates for the snapshots that were packaged in the template. Offline instantiation files are unique for each client, and multiple templates can be instantiated on a single client.

After the offline instantiation file is made available to the client site through file transfer protocol (FTP), compact disk, or some other delivery mechanism, a user at the client site loads the file into the Oracle8*i* Lite database using the Oracle Client Replication Tool. Offline instantiation is very useful for large-scale, rapid deployment, and does not require a network connection between the client and the server.

Online instantiation can be completed in one step using the using the Oracle Client Replication Tool. The Oracle Client Replication Tool combines packaging and instantiation into one single step. Online instantiation requires a network connection between the client and the server.

> **See Also:** Chapter 4, "Deployment Templates Concepts & Architecture" for more information about deployment templates. Also, see the Oracle8*i* Lite documentation for information about the Oracle Client Replication Tool.

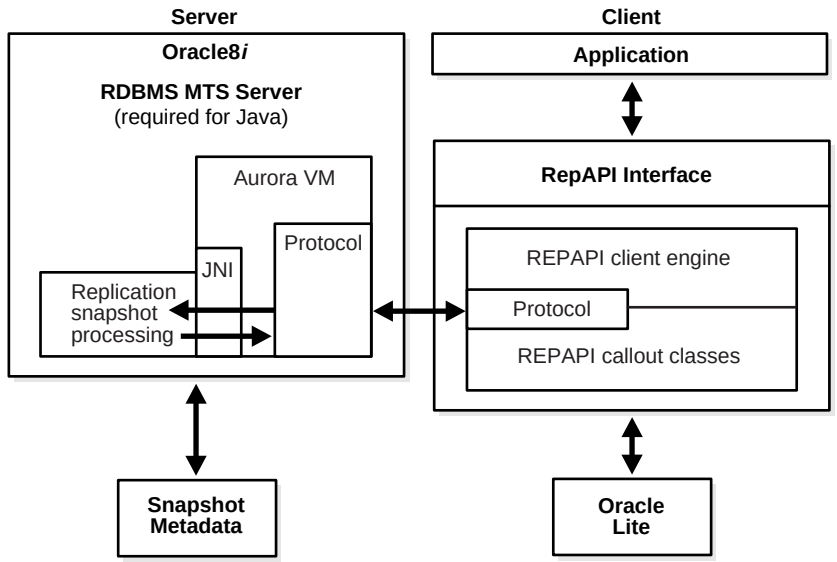
RepAPI Concepts and Architecture

RepAPI includes three components:

- A small server component residing in the Java virtual machine of Oracle8*i*
- A set of callout classes implemented by Oracle8*i* Lite and integrated with the Java Access Class
- A client engine

All of these components are written in Java, and the client and server components communicate through Common Object Request Broker Architecture (CORBA) stubs. The Java RepAPI protocol encapsulates row data, which is essentially a byte stream, into row sets. Each row set sends a small amount of descriptive data. Every row set exchanged by the client and the server is associated with a snapshot. When data is pushed from a RepAPI client to a master site, each row set is bound to a transaction. When data is pulled from a master site to a RepAPI client, each row set is bound to a particular snapshot and DML operation (INSERT, UPDATE, or DELETE, which are indicated by a flag, not by SQL, in the row set).

Figure C–1 RepAPI Architecture



The following sections contain detailed information about the components of the RepAPI architecture.

Server Component

The server component of RepAPI is a CORBA interface supporting the following basic distributed operations:

- Listing available refresh groups
- Packaging a deployment template for instantiation of a refresh group
- Initiating a synchronization of a refresh group
- Dropping a refresh group

All of these operations are purely at the refresh group level. A refresh group name is submitted as a parameter to the requests for these operations. Distributed operations on individual snapshots are not supported. Refresh group names and descriptions are pre-configured by a DBA in the form of deployment templates. A refresh group name is the same as its deployment template name.

The DBA loads the server Java object into the Virtual Machine (VM) using the loadjava utility supplied with Oracle8*i*. The DBA makes the server object known to clients through the Oracle 8*i* publish utility. Specifically, the DBA loads and publishes a Java object named oracle.repapi.server.RServer so that it can be activated by a service URL. Here is an example of a typical URL referring to this object:

```
sess_iiop://myserver:2481:RNDM/etc/repapi
```

The URL has the following parts:

- myserver is the host name
- 2481 is the Internet Inter-ORB Protocol (IIOP) listener port number
- RNDM is the SID for the database
- /etc/repapi is the published name for the object

The DBA can configure the published name of this object. By default, the name is /etc/repapi. The end-user simply specifies the URL in the application; no other client configuration is necessary. When a RepAPI client connects, the Object Request Broker (ORB) instantiates a transient server object, which interacts with the snapshot processing code. Oracle removes the transient object when the session disconnects.

The server object interacts with the Oracle8*i* server through the Java Native Interface (JNI). The majority of the RepAPI snapshot processing occurs on the server and is executed after the JNI callouts. The server runs the pre-fabricated snapshot queries, defined by the appropriate deployment template. When the server has row data to send to the client, it makes callbacks into the VM through JNI to deliver messages back to the RepAPI client engine. The RepAPI server component therefore takes advantage of CORBA's full-duplex messaging ability.

> **See Also:** *Oracle8i Java Developer's Guide*for more information about the Java Virtual Machine in Oracle8*i*.

Callout Classes

The callout classes are the glue between the RepAPI client engine and Oracle8*i* Lite. They provide a generic set of database and table operations that can interact with the client engine. The callout classes are not directly called by Oracle8*i* Lite; the RepAPI client engine invokes the callout classes native to Oracle8*i* Lite. Therefore, the RepAPI callout classes hide specific implementation of the foreign data store from the client engine, enabling the RepAPI client engine to be generic and flexible.

Client Engine

The RepAPI client engine is the foundation of RepAPI. It connects the server component and callout classes. The client engine:

- Dispatches requests to the server with the CORBA IIOP-based protocol
- Manages snapshot refresh groups on the client
- Drives synchronization (refresh) operations
- Pushes changes to the master
- Applies refresh changes at the client
- Provides an interface for application development

Two objects (RepAPI and Refresh Group) provide public methods for application developers. The RepAPI object provides public methods for the application-driven refresh group operations, including the following operations:

- Show available refresh groups on the client
- Drop and create refresh group
- Offline instantiate

The Refresh Group object provides a public synchronization method, which includes the standard refresh options: fast, complete, and force. In addition, you can use synchronization to perform a "push-only" operation, which propagates changes from the client to the server only.

The RepAPI client engine maintains a small set of metadata tables in the Oracle8*i* Lite database. This metadata includes the following:

- The URL of the server object, to be used during the next connect attempt
- The transaction ID of the last successfully pushed transaction (if updateable snapshots were involved)
- A list of refresh groups, including their names, identification numbers, and their refresh sequence numbers. The refresh sequence numbers act as a verification check between the two sides
- A list of snapshots, including their names, identification numbers, and a flag indicating whether they are updateable

In addition to this metadata, RepAPI maintains a queue table for updateable snapshot transactions at the Oracle8*i* Lite client. This transaction queue is bound to a particular master site and is populated when triggers capture any updateable snapshot updates. When a refresh is initiated, the transaction queue is always completely pushed and sent to the master associated with the refresh group, even if the refresh request was for a group that had no updateable snapshots. The transaction boundaries are preserved when the queue is pushed to the Oracle8*i* server.

Before Using RepAPI on the Oracle8*i* Server

Complete the following steps before you begin using RepAPI on your Oracle8*i* server:

1. Edit the initialization parameters in your initialization parameter file:

   a. Make sure following initialization parameters are set to at least the following values (the settings can be higher if necessary, but not lower):

      SHARED_POOL_SIZE= 50000000

      JAVA_POOL_SIZE=50000000

      OPEN_LINKS=10

   b. If the following initialization parameters do not exist in your initialization parameter file, add them. If they do exist, make sure they are set to at least the following values (the settings can be higher if necessary, but not lower):

      ENQUEUE_RESOURCES=1000

      OPEN_CURSORS=100

      MAX_ENABLED_ROLES=30

   c. Confirm that the MTS_DISPATCHERS initialization parameter is set to the following:

      MTS_DISPATCHERS =
      "(PROTOCOL=TCP)(PRE=oracle.aurora.server.SGiopServer)"

2. Verify that the `listener.ora` file in the *`ORACLE_HOME`*`/network/admin` directory includes a line in the LISTENER configuration similar to the following, where the hostname is myserver:

```
(DESCRIPTION =
     (PROTOCOL_STACK =
        (PRESENTATION = GIOP)
        (SESSION = RAW))
     (ADDRESS =
        (PROTOCOL = TCP)
        (HOST = myserver)
         (PORT = 2481)))
```

Note: If you are using Secure Sockets Layer (SSL), set the PROTOCOL to TCPS, and set the PORT to 2482. Also, 2481 and 2482 are the default PORT settings for a "Typical" installation; however, your settings can be different.

---

3. Shutdown and restart the Oracle8*i* server so that the changes to the initialization parameters take effect.

Loading and Publishing the RepAPI Server Object

For RepAPI clients to access the server component of RepAPI, you can load the appropriate RepAPI server object into the Oracle8*i* Virtual Machine (VM) using the loadjava utility. Then, you can publish the object using the publish utility.

RepAPI Server Object Loaded and Published at Install: If you installed the Java option for Oracle8*i*, then the RepAPI server object should have been loaded and published automatically during your installation. The procedure below is provided for your reference. If you want to verify that the RepAPI server object was loaded and published successfully, go to Step 3 on page C-12 for verification instructions.

See Also: This section describes configuring your Oracle8i server to support RepAPI.

- For information about using RepAPI on your Oracle8*i* Lite client, see the Oracle8*i* Lite documentation.
- For more information about the loadjava utility and the publish utility, see *Oracle8i Enterprise JavaBeans and CORBA Developer's Guide.*

To load and publish the RepAPI server object, complete the following steps:

1. Run the loadjava utility to load the `repapi_s.jar` file into the database.

   The syntax for loading the `repapi_s.jar` file at a command prompt follows:

```
loadjava
   -synonym
   -resolve
   -thin
   -user sys/SYS_PASSWORD@HOST:LISTENER_PORT:ORACLE_SID
   -grant public
   ORACLE_HOME/rdbms/jlib/repapi_s.jar
```

The syntax includes the following variables:

- *SYS_PASSWORD* is the password for SYS user.
- *HOST* is the hostname on which the Oracle8*i* server is installed.
- *LISTENER_PORT* is the port for your listener. The default is 1521.
- *ORACLE_SID* is the System Identifier (SID) specified on the Oracle8i server.
- *ORACLE_HOME* is the directory in which the Oracle8*i* server is installed.

> **Note:** This loadjava syntax is appropriate for both UNIX systems and Windows platforms. Make the appropriate adjustments for file path naming.

For example, suppose the following is true for your environment:

- *SYS_PASSWORD* is change_on_install.
- *HOST* is myserver.
- *LISTENER_PORT* is 1521.
- *ORACLE_SID* is DB1.
- *ORACLE_HOME* is the `/oracle/product/8.1` directory.

For this environment, the following is the correct loadjava utility syntax:

```
loadjava -synonym -resolve -thin
   -user sys/change_on_install@myserver:1521:DB1
   -grant public /oracle/product/8.1/rdbms/jlib/repapi_s.jar
```

2. Run the publish utility to publish the RepAPI object with a service URL.

```
publish
   -u system
   -password SYSTEM_PASSWORD
   -s sess_iiop://HOST:ORB_PORT:ORACLE_SID/PUBLISHED_OBJECT
   oracle.repapi.server.RServer
   oracle.repapi.serverstub.RSHelper
   -schema SYS
   -grant public
```

The syntax includes the following variables:

- *SYSTEM_PASSWORD* is the password for SYSTEM user.
- *HOST* is the hostname on which the Oracle8*i* server is installed.
- *ORB_PORT* is the listener port number for the IIOP. The default is 2481 for TCP connections and 2482 for TCP connections using Secure Sockets Layer (SSL).
- *ORACLE_SID* is the System Identifier (SID) specified on the Oracle8*i* server.
- *PUBLISHED_OBJECT* is the name of the published object (default is /etc/repapi).

Note: This publish syntax is appropriate for both UNIX systems and Windows platforms. Make the appropriate adjustments for file path naming.

For example, suppose the following is true for your environment:

- SYSTEM_PASSWORD is manager.
- *ORB_PORT* is 2481.
- *HOST* is myserver.
- *ORACLE_SID* is DB1.
- *PUBLISHED_OBJECT* is /etc/repapi.

For this environment, the following is the correct publish utility syntax:

```
publish -u system -password manager
   -s sess_iiop://myserver:2481:DB1/etc/repapi
   oracle.repapi.server.RServer
   oracle.repapi.serverstub.RSHelper
   -schema SYS -grant public
```

3. Verify that the object was published successfully by completing the following steps:

   a. Create a session to view the published objects by entering the following command:

   ```
   sess_sh
      -user system
      -password SYSTEM_PASSWORD
      -service sess_iop://HOST:AURORA_ORB_PORT:ORACLE_SID
   ```

   The syntax includes the following variables:

   - *SYSTEM_PASSWORD* is the password for SYSTEM user.
   - *HOST* is the hostname on which the Oracle8*i* server is installed.
   - *AURORA_ORB_PORT* is the port number specified when the java object was published. By default, this port number is set to 2481 for TCP connections and 2482 for TCP with SSL connections. Aurora is the Java Virtual Machine for the Oracle8*i* server.
   - *ORACLE_SID* is the System Identifier (SID) specified on the Oracle8*i* server.

   For example, suppose the following is true for your environment:

   - *SYSTEM_PASSWORD* is manager.
   - *HOST* is myserver.
   - *AURORA_ORB_PORT* is 2481.
   - *ORACLE_SID* is DB1.

   For this environment, the following is the correct publish utility syntax:

   ```
   sess_sh -user system -password manager
      -service sess_iop://myserver:2481:DB1
   ```

b. Wait for the shell to become visible. The following appears on your screen when the prompt becomes visible:

```
AURORA ORB  shell
$
```

Note: Do not continue until you see this prompt. It may take several minutes for the prompt to appear.

c. Check the permissions on the published object, etc/repapi, by entering the following:

```
cd test
ls -l
```

The permissions for Read, Write, and Execute are displayed at the beginning of the line, and the word PUBLIC should be under the Read and Execute columns, as in the following output:

```
Read     Write    Exec
PUBLIC   SYSTEM   PUBLIC
```

If your output is different for the Read and Execute columns, enter the following commands:

```
chmod +r public repapi
chmod +e public repapi
```

Then, check the permissions again and verify that repapi has PUBLIC permission for READ and EXECUTE:

```
ls -l
```

d. Exit the session:

```
exit
```

Generating Offline Instantiation Files

Use the DBMS_REPCAT_INSTANTIATE.INSTANTIATE_OFFLINE_REPAPI function of the replication management API to generate offline instantiation files for RepAPI clients.

The naming convention for the offline instantiation file is the following:

user_templatename_siteid.oli

So, if the username is SCOTT, the template name is MYTEMPLATE, and the site identification number is 1234, the filename is the following:

scott_mytemplate_1234.oli

You can specify the site identification number by using the SITE_ID parameter when you run the function. If you do not specify a site identification number, Oracle generates a number automatically. The site identification number is a temporary placeholder. Oracle8*i* Lite replaces this value with the globally unique Oracle8*i* Lite site identifier.

A user can generate an offline instantiation file, as in the following example:

```
DECLARE
  i number;
BEGIN
  i := dbms_repcat_instantiate.instantiate_offline_repapi(
       refresh_template_name => 'PRODUCTS_DT',
       trace_vector => 0);
END;
/
```

The default location for the offline instantiation file is the *ORACLE_HOME*/dbs directory on the server. Also, if the deployment template contains template parameters, the parameter values for the connected user are applied.

A user with DBA privileges can specify a username and a path for the offline instantiation file, as in the following example:

```
DECLARE
  i number;
BEGIN
  i := dbms_repcat_rgt.instantiate_offline_repapi(
       refresh_template_name => 'PRODUCTS_DT',
       user_name => 'SCOTT',
       offline_dirpath => '/mydirectory',
       trace_vector => 0);
END;
/
```

Here, the deployment template name is PRODUCTS_DT, the directory path for the offline instantiation file is `/mydirectory`, and the trace level for debugging is 0 (zero). Also, if the deployment template contains template parameters, the parameter values for the specified user are applied; so, the parameter values for user SCOTT are applied in the example above.

> **See Also:** *Oracle8i Replication Management API Reference* for more information about using the DBMS_REPCAT_INSTANTIATE.INSTANTIATE_OFFLINE_REPAPI function.

Note: You can also use Replication Manager to generate offline instantiation files for RepAPI clients. See the Replication Manager online help for instructions.

Administering RepAPI

The following tools are available for administering and monitoring RepAPI.

Replication Manager

You can use Replication Manager to create deployment templates and offline instantiation files for RepAPI client sites. Also, when client sites are available over the network, you can use Replication Manager to monitor the snapshots at the client site and refresh them on demand.

> **See Also:** The Replication Manager online help for more information about using the Replication Manager.

Replication Management API

The replication management API includes the following PL/SQL functions and procedures:

- DBMS_REPCAT_INSTANTIATE.INSTANTIATE_OFFLINE_REPAPI
- DBMS_REPCAT_RGT.INSTANTIATE_OFFLINE_REPAPI
- DBMS_REPCAT_RGT.DROP_SITE_INSTANTIATION

> **See Also:** *Oracle8i Replication Management API Reference* for more information about using these PL/SQL functions and procedures to administer RepAPI client sites.

Replication Catalog

You can use the DBA_REPCAT_TEMPLATE_SITES static data dictionary view to monitor the status of template instantiation at RepAPI client sites.

> **See Also:** *Oracle8i Replication Management API Reference* for more information about the DBA_REPCAT_TEMPLATE_SITES static data dictionary view.

Oracle Client Replication Tool

The Oracle Client Replication Tool is a Java application with a graphical user interface that is installed with Oracle8*i* Lite. Users can use the Oracle Client Replication Tool to instantiate deployment templates, and application developers can use the Oracle Client Replication Tool as sample code to build custom applications.

In addition, the client Jar files are included with an Oracle server installation for completeness. These Jar files are in the *ORACLE_HOME*/rdbms/jlib directory on the server.

> **See Also:** Your Oracle8*i* Lite documentation for information about using the Oracle Client Replication Tool.

Client Sample Program

A sample Java program for RepAPI is included with your Oracle8*i* server distribution. This sample program is provided so that you can test your Oracle8*i* Lite client environment and to help you learn the integration techniques for building an application using RepAPI in an Oracle8*i* Lite client environment.

Installing the Sample Program

Complete the following steps to install the sample program:

1. Copy the RepSample.java file in the *ORACLE_HOME*/rdbms/demo directory to a directory of your choice.
2. Make sure the directory to which you copied the files in Step 1 is included in the CLASSPATH environment variable.
3. Compile RepSample.java by entering the following command at a command prompt:

   ```
   javac RepSample.java
   ```

   This command creates a class file named RepSample.class.

Using the Sample Program

The sample program uses Oracle's Java RepAPI replication library to replicate data between an Oracle8*i* server and an Oracle8*i* Lite client. This code assumes that the username and password on the Oracle8*i* Lite site is SYSTEM/MANAGER. If your username and password are different, make the appropriate edits to the sample program. A minimum of nine arguments are expected with this code, and the arguments must be specified in the correct order.

The sample program demonstrates the two modes for performing work: offline (not connected to the master site) and online (connected to the master site). The offline mode only allows for the instantiation of an existing file (`*.oli` extension type) containing the metadata and initial data for a particular deployment template (refresh group), user, and site. The online mode demonstrates the ability to create a refresh group while connected to the master site and to synchronize the refresh group with the master site.

Following is the full syntax for the sample program:

```
java RepSample ( -offline | -online )
   -user USER_NAME
   -pass USER_PASSWORD
   -master MASTER_DATABASE_NAME
   -dbid ORACLE_LITE_DATABASE
   [-url SERVICE_URL]
   [-offlineinstantiate OFFLINE_FILE_NAME]
   [-create TEMPLATE_NAME]
   [-sync | syncf TEMPLATE_NAME]
   [-drop TEMPLATE_NAME]
   [-list]
```

The syntax includes the following variables:

- *USER_NAME* is the identity of the authorized user on the Oracle8*i* server.
- *USER_PASSWORD* is the password for the above *USER_NAME* on the Oracle8*i* server.
- *MASTER_DATABASE_NAME* is a label for the Oracle8*i* server stored within the Oracle8*i* Lite client. This label is used to name the transaction queue table for an individual Oracle8*i* server.

  Caution: The value for *MASTER_DATABASE_NAME* cannot contain any special characters, including hyphens or dashes. This is a table-naming restriction within Oracle8*i* Lite.

- *ORACLE_LITE_DATABASE* is the fully qualified filename for the Oracle8*i* Lite database.

- *SERVICE_URL* is in the following form:

  sess_iiop://*HOST:AURORA_ORB_PORT:ORACLE_SID/PUB_OBJECT*

  where:

  - *HOST* is the hostname on which the Oracle8*i* server is installed.
  - *AURORA_ORB_PORT* is the port number specified when the Java object was published. By default, this port number is set to 2481 for TCP connections and 2482 for TCP with SSL connections.
  - *ORACLE_SID* is the System Identifier (SID) specified on the Oracle8*i* server.
  - *PUB_OBJECT* is the published object name as configured in the publish command as part of the RepAPI server configuration. By default, /etc/repapi is published object name. This name can be configured by the DBA.
  - *OFFLINE_FILE_NAME* is the fully qualified filename of the offline instantiation file. This file is the serialized file generated by the DBA using either Replication Manager or the DBMS_REPCAT_INSTANTIATE.INSTANTIATE_OFFLINE_REPAPI function at the master site. An *OFFLINE_FILE_NAME* has the following file type identifier:

    .oli

- *TEMPLATE_NAME* is the name of the deployment template (and refresh group) that is to be manipulated. The template name and refresh group name must be identical. These names *are not* configurable by the end user. The template name is specified by the DBA.

Example Usage

This section contains examples of using the sample program.

Example 1 To load the data from a template which has been offline instantiated earlier, enter the following command:

```
java RepSample -offline -user scott -pass tiger -master sales
   -dbid c:\orant\oldb40\polite.odb -offlineinstantiate SCOTT_T1_CLNT2.oli
```

Example 2 To online instantiate a deployment template named T2 by connecting online, enter the following command:

```
java RepSample -online -user scott -pass tiger -master sales
   -dbid c:\orant\oldb40\polite.odb
   -url sess_iiop://salespc.us.oracle.com:2481:db1/etc/repapi
   -create T2
```

Example 3 To perform a complete refresh of group T2 created in Example 2 above, enter the following command:

```
java RepSample -online -user scott -pass tiger -master sales
   -dbid c:\orant\oldb40\polite.odb
   -url sess_iiop://salespc.us.oracle.com:2481:db1/etc/repapi
   -sync T2
```

Example 4 To perform a force refresh of group T2 created in Example 2 above, enter the following command, using the syncf parameter instead of the sync parameter:

```
java RepSample -online -user scott -pass tiger -master sales
   -dbid c:\orant\oldb40\polite.odb
   -url sess_iiop://salespc.us.oracle.com:2481:db1/etc/repapi
   -syncf T2
```

Example 5 To online instantiate a deployment template named T3 and synchronize the refresh group T2 created in Example 2 above, enter the following command:

```
java RepSample -online -user scott -pass tiger -master sales
   -dbid c:\orant\oldb40\polite.odb
   -url sess_iiop://salespc.us.oracle.com:2481:db1/etc/repapi
   -create T3  -sync T2
```

Example 6: To drop the refresh group T2 created in Example 2 above by connecting online, enter the following:

```
java RepSample -online -user scott -pass tiger -master sales
   -dbid c:\orant\oldb40\polite.odb
   -url sess_iiop://salespc.us.oracle.com:2481:db1/etc/repapi
   -drop T2
```

Example 7 To list all the available and instantiated refresh groups, enter the following:

```
java RepSample -online -user scott -pass tiger -master sales
   -dbid c:\orant\oldb40\polite.odb
   -url sess_iiop://salespc.us.oracle.com:2481:db1/etc/repapi
   -list
```

Index

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U

V

W