Oracle® Database

Advanced Replication

11*g* Release 2 (11.2)

E10706-01

July 2009

ORACLE®

Oracle Database Advanced Replication, 11*g* Release 2 (11.2)

E10706-01

Primary Author: Randy Urbano

Contributors: Nimar Arora, Yuen Chan, Alan Downing, Curt Elsbernd, Yong Feng, Jairaj Galagali, Lewis Kaplan, Jonathan Klein, Jing Liu, Edwina Lu, Pat McElroy, Maria Pratt, Arvind Rajaram, Neeraj Shodhan, Wayne Smith, Jim Stamos, Janet Stern, Mahesh Subramaniam, Lik Wong, David Zhang

Contents

3 Materialized View Concepts and Architecture

4 Deployment Templates Concepts and Architecture

5 Conflict Resolution Concepts and Architecture

6 Planning Your Replication Environment

List of Figures

List of Tables

Preface

Oracle Database Advanced Replication describes the features and functionality of Advanced Replication. Specifically, *Oracle Database Advanced Replication* contains conceptual information about Advanced Replication, as well as information about planning your replication environment and troubleshooting replication problems.

This Preface contains these topics:

Audience

Oracle Database Advanced Replication is intended for database administrators and application developers who develop and maintain replication environments. These administrators and application developers perform one or more of the following tasks:

- Plan for a replication environment
- Configure the following types of replication environments:
  - Read-only materialized view
  - Updatable materialized view
  - Single master replication
  - Multimaster replication
- Use deployment templates to create a materialized view environment
- Configure conflict resolution
- Administer a replication environment
- Perform troubleshooting activities when necessary
- Manage job queues
- Manage deferred transactions
- Use the Advanced Replication interface in Oracle Enterprise Manager to create, monitor, and manage replication environments

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

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

Related Documents

For more information, see these Oracle resources:

- *Oracle Database Advanced Replication Management API Reference*
- The Advanced Replication online Help in Oracle Enterprise Manager
- *Oracle Database Concepts*
- *Oracle Database Administrator's Guide*
- *Oracle Database SQL Language Reference*
- *Oracle Database PL/SQL Language Reference*
- *Oracle Streams Replication Administrator's Guide* if you want to migrate your Advanced Replication environment to Oracle Streams

Many of the examples in this book use the sample schemas of the sample database, which is installed by default when you install Oracle Database. Refer to *Oracle*

Database Sample Schemas for information about how these schemas were created and how you can use them.

Printed documentation is available for sale in the Oracle Store at

http://oraclestore.oracle.com/

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

http://www.oracle.com/technology/membership/

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

http://www.oracle.com/technology/documentation/

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Introduction to Advanced Replication

This chapter explains the basic concepts and terminology related to Advanced Replication.

This chapter contains these topics:

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

Overview of 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. Advanced 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 `employees` table resides at only the `ny.example.com` database in a distributed database system that also includes the `hk.example.com` and `la.example.com` databases. Replication means that the same data is available at multiple locations. For example, the `employees` table is available at `ny.example.com`, `hk.example.com`, and `la.example.com`.

Some of the most common reasons for using replication are described as follows:

Availability

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 different

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.

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 different servers, thereby reducing the load at all servers.

Disconnected Computing

A **materialized view** is a complete or partial copy (replica) of a target table from a single point in time. Materialized views 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) materialized views on demand. When users refresh materialized views, they update the central database with all of their changes, and they receive any changes that happened while they were disconnected.

Network Load Reduction and Mass Deployment

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.

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

Note: The Advanced Replication feature is automatically installed and upgraded in every Oracle Database installation.

See Also: *Oracle Database Administrator's Guide* for more information about distributed databases

Applications that Use Replication

Replication supports a variety of applications that often have different requirements. Some applications allow for relatively autonomous individual materialized view 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.

Advanced Replication can be used for each of the types of applications described in the previous paragraphs, and for systems that combine aspects of both types of applications. In fact, Advanced Replication can support both mass deployment and server-to-server replication, enabling integration into a single coherent environment. In such an environment, for example, sales force automation and customer service call centers can share data.

Advanced Replication can replicate data in environments that use different releases of Oracle and in environments that run Oracle on different operating systems. Therefore, applications that use data in such an environment can use Advanced Replication.

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. Advanced Replication enables you to replicate the following types of objects:

- Tables
- Indexes
- Views and Object Views
- Packages and Package Bodies
- Procedures and Functions
- User-Defined Types and Type Bodies
- Triggers
- Synonyms
- Indextypes
- User-Defined Operators

Regarding tables, replication supports advanced features such as partitioned tables, index-organized tables, tables containing columns that are based on user-defined types, and object tables.

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.

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, each 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 **materialized view sites**. One site can be both a master site for one replication group and a materialized view site for a different replication group. However, one site cannot be both the master site and the materialized view site for the same replication group.

The differences between master sites and materialized view sites are the following:

- A replication group at a master site is more specifically referred to as a **master group**. A replication group at a materialized view site is based on a master group and is more specifically referred to as a **materialized view 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 materialized views at a materialized view site can contain all or a subset of the table data within a master group. For example, if the `hr_repg` master group contains the tables `employees` and `departments`, then all of the master sites participating in a master group must maintain a complete copy of `employees` and `departments`. However, one materialized view site might contain only a materialized view of the `employees` table, while another materialized view site might contain materialized views of both the `employees` and `departments` tables.
- All master sites in a multimaster replication environment communicate directly with one another to continually propagate data changes in the replication group. Materialized view sites contain an image, or materialized view, of the table data from a certain point in time. Typically, a materialized view is refreshed periodically to synchronize it with its master site. You can organize materialized views into **refresh groups**. Materialized views in a refresh group can belong to one or more materialized view groups, and they are refreshed at the same time to ensure that the data in all materialized views in the refresh group correspond to the same transactionally consistent point in time.

Types of Replication Environments

Advanced Replication supports the following types of replication environments:

- Multimaster Replication
- Materialized View Replication
- Multimaster and Materialized View Hybrid Configurations

Multimaster Replication

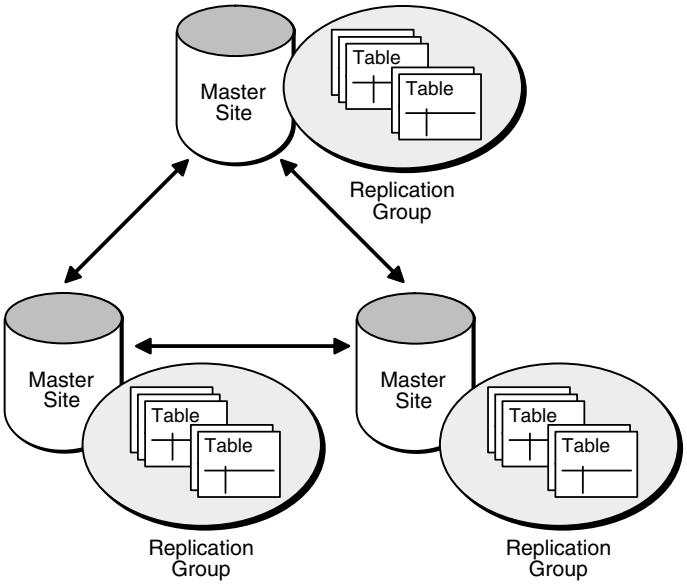
Multimaster replication (also called peer-to-peer or *n*-way replication) enables multiple sites, acting as equal peers, to manage groups of replicated database objects. Each site in a multimaster replication environment is a master site, and each site communicates with the other master sites.

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, information about a data manipulation language (DML) change on a table is stored in the deferred transactions 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 these intervals.

Using asynchronous replication means that data 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 prebuilt mechanisms that can be implemented to resolve them. Information about unresolved conflicts is stored in an error log.

Figure 1–1 Multimaster Replication



Master Group Quiesce

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 add a new master group object. Stopping all replication activity for a master group is called **quiescing** the group. When a master group is quiesced, users cannot issue DML statements on any of the objects in the master group. Also, all deferred transactions must be propagated before you can quiesce a master group. Users can continue to query the tables in a quiesced master group.

Materialized View Replication

A materialized view contains a complete or partial copy of a target **master** from a single point in time. The target master can be either a master table at a master site or a master materialized view at a materialized view site. A **master materialized view** is a materialized view that functions as a master for another materialized view. A **multitier materialized view** is one that is based on another materialized view, instead of on a master table.

Materialized views provide the following benefits:

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

A materialized view can be read-only, updatable, or writeable, and these types of materialized views provide benefits in addition to those listed previously.

Overview of Read-Only Materialized Views

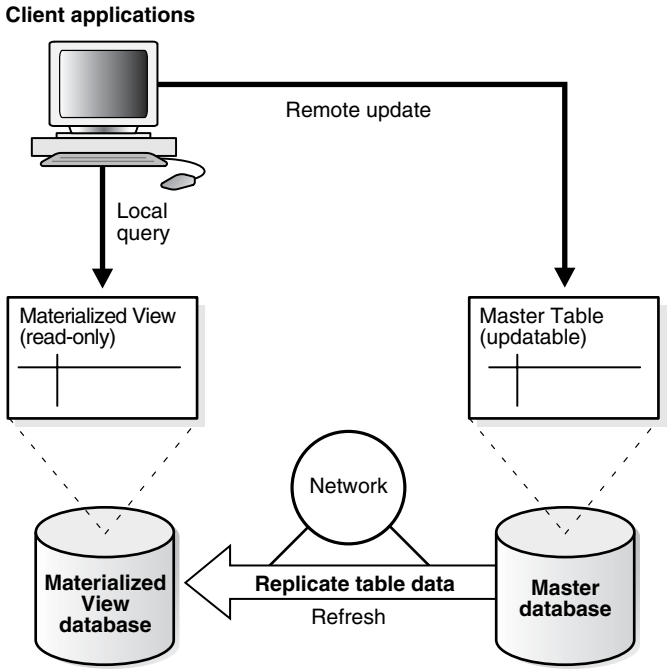
In a basic configuration, materialized views can provide read-only access to the table data that originates from a master site or master materialized view site. Applications can query data from **read-only materialized views** to avoid network access to the master site, regardless of network availability. However, applications throughout the system must access data at the master site to perform data manipulation language changes (DML). Figure 1–2 illustrates basic, read-only replication. The master tables and master materialized views of read-only materialized views do not need to belong to a replication group.

Read-only materialized views provide the following benefits:

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

> **See Also:** "Available Materialized Views" on page 3-5 for more information about complex materialized views

Figure 1–2 Read-Only Materialized View Replication



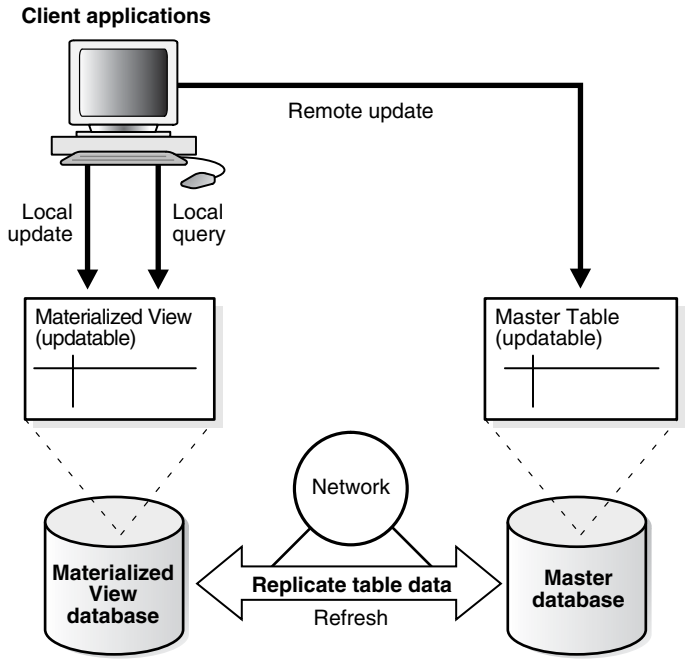
Overview of Updatable Materialized Views

In a more advanced configuration, you can create an **updatable materialized view** that allows users to insert, update, and delete rows of the target master table or master materialized view by performing these operations on the materialized view. An updatable materialized view can also contain a subset of the data in the target master. Figure 1–3 illustrates a replication environment using updatable materialized views.

Updatable materialized views are based on tables or other materialized views that have been set up to support replication. In fact, updatable materialized views must be

part of a **materialized view group** that is based on another replication group. For changes made to an updatable materialized view to be pushed back to the master during refresh, the updatable materialized view must belong to a materialized view group.

Figure 1–3 Updatable Materialized View Replication



Updatable materialized views have the following properties.

- They are always based on a single table, although multiple tables can be referenced in a subquery.
- They can be incrementally (or fast) refreshed.
- Oracle propagates the changes made to an updatable materialized view to the materialized view's remote master table or master materialized view. The updates to the master then cascade to all other replication sites.

Updatable materialized views provide the following benefits:

- Allow users to query and update a local replicated data set even when disconnected from the master site or master materialized view site.
- Require fewer resources than multimaster replication, while still supporting data updates. Materialized views can reduce the amount of stress placed on network resources because materialized views can be refreshed on demand, while multimaster replication propagates changes at regular intervals. In addition, because materialized views can reside in a database that contains far less data, the disk space and memory requirements for materialized view clients can be less than the requirements for an Oracle server containing master sites.

Overview of Writeable Materialized Views

You can create a materialized view using the `FOR UPDATE` clause during creation but then never add the materialized view to a materialized view group. In this case, users can perform data manipulation language (DML) changes on the materialized view,

but these changes cannot be pushed back to the master and are lost if the materialized view refreshes. Such materialized views are called **writeable materialized views**.

Overview of Row and Column Subsetting with Materialized Views

Both row and column subsetting enable you to create materialized views that contain a partial copy of the data at a master table or master materialized view. Such materialized views can be helpful for regional offices or sales forces that do not require the complete data set.

Row subsetting enables you to include only the rows that are needed from the masters in the materialized views by using a `WHERE` clause. Column subsetting enables you to include only the columns that are needed from the masters in the materialized views. You do this by specifying particular columns in the `SELECT` statement during materialized view creation.

Note: Column subsetting of updatable materialized views is supported only through the use of deployment templates and the Advanced Replication interface in Oracle Enterprise Manager. This restriction does not apply to column subsetting of read-only materialized views.

See Also:

- "Data Subsetting with Materialized Views" on page 3-12
- "Column Subsetting with Deployment Templates" on page 4-15

Materialized View Refresh

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

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

Refresh Groups

When it is important for materialized views 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 materialized views in the refresh group correspond to the same transactionally consistent point in time. Both read-only and updatable materialized views can be included in a refresh group. A materialized view in a refresh group still can be refreshed individually, but doing so nullifies the benefits of the refresh group because refreshing the materialized view individually does not refresh the other materialized views in the refresh group.

Materialized View Log

A **materialized view log** is a table at the materialized view's master site or master materialized view site that records all of the DML changes to the master table or master materialized view. A materialized view log is associated with a single master table or master materialized view, and each of those has only one materialized view

log, regardless of how many materialized views refresh from the master. A fast refresh of a materialized view is possible only if the materialized view's master has a materialized view log. When a materialized view is fast refreshed, entries in the materialized view's associated materialized view log that have appeared since the materialized view was last refreshed are applied to the materialized view.

Deployment Templates

Deployment templates simplify the task of deploying and maintaining many remote materialized view sites. Using deployment templates, you can define a collection of materialized view definitions at a master site, and you can use parameters in the definitions so that the materialized views 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 materialized views are created and populated at the remote site. The 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 materialized view site, the object DDL (for example, `CREATE MATERIALIZED VIEW` or `CREATE TABLE` statements) is executed to create the schema objects at the materialized view 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).

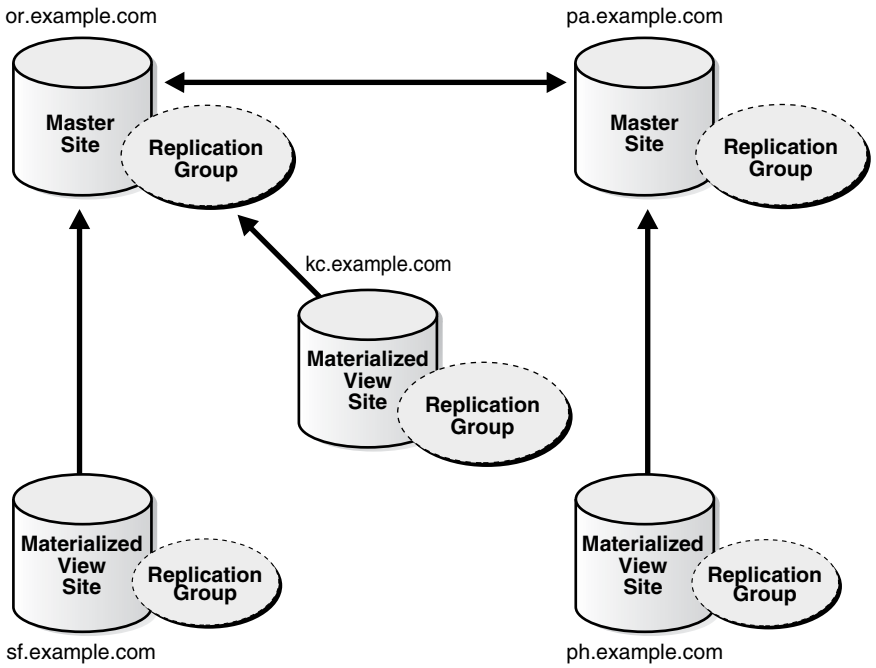
Offline instantiation is often used to decrease server loads during peak usage periods and to reduce remote connection times. To instantiate a template offline, you package the template and required data on some type of storage media, such as tape, CD-ROM, and so on. Then, instead of pulling the data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

Multimaster and Materialized View Hybrid Configurations

Multimaster replication and materialized views can be combined in **hybrid** or "mixed" configurations to meet different application requirements. Hybrid configurations can have any number of master sites and multiple materialized view 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. Materialized views 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 materialized views and replicated master tables include the following:

- Replicated master tables must contain data for the full table being replicated, whereas materialized views can replicate subsets of master table data.
- Multimaster replication enables you to replicate changes for each transaction as the changes occur. Materialized view refreshes are set oriented, propagating changes from multiple transactions in a more efficient, batch-oriented operation, but at less frequent intervals.
- If conflicts occur because of changes made to multiple copies of the same data, then detection and resolution of conflicts always occurs at a master site or a master materialized view site.

Scheduled Links

Both master sites and materialized view sites use **scheduled links** to propagate data changes to other sites. A scheduled link is a database link with a user-defined schedule to push deferred transactions. A scheduled link determines how a master site propagates its deferred transaction queue to another master site, or how a materialized view 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.

Administration Tools for a Replication Environment

Several tools are available for configuring, administering, and monitoring your replication environment. The Advanced Replication interface in Oracle Enterprise Manager provides a powerful graphical user interface (GUI) to help you manage your environment, while the replication management application programming interface (API) provides you with a familiar API to build customized scripts for replication

administration. Additionally, the replication catalog keeps you informed about your replication environment.

Advanced Replication Interface in Oracle Enterprise Manager

To help configure and administer replication environments, Oracle provides a sophisticated Advanced Replication interface in Oracle Enterprise Manager. To access the Advanced Replication interface, go to the Data Movement subpage in Enterprise Manager. The Advanced Replication interface online Help is the primary documentation source for this tool. Figure 1–5 shows the Advanced Replication interface in Enterprise Manager.

Figure 1–5 Advanced Replication Interface in Enterprise Manager

See Also: The Advanced Replication interface online Help in Oracle Enterprise Manager

Replication Management API

The replication management API is a set of PL/SQL packages that encapsulate procedures and functions that you can use to configure an Advanced Replication environment. The replication management API is a command-line alternative to the Advanced Replication interface in Oracle Enterprise Manager. In fact, the Advanced Replication interface uses the procedures and functions of the replication management API to perform its work. For example, when you use the Advanced Replication interface to create a new master group, the interface completes the task by making a call to the `CREATE_MASTER_REPGROUP` procedure in the `DBMS_REPCAT` package. The replication management API makes it easy for you to create custom scripts to manage your replication environment.

See Also: *Oracle Database Advanced Replication Management API Reference* for more information about using the replication management API

Replication Catalog

Every master site and materialized view 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:** *Oracle Database Advanced 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 the Advanced Replication interface in the Oracle Enterprise Manager or the `DBMS_REPCAT` package in the replication management API. Specifically, if you use the `DBMS_REPCAT` package, then use the `CREATE_MASTER_REPOBJECT` procedure to add objects to a master group, and use `ALTER_MASTER_REPOBJECT` to modify replicated objects. You can also use the `EXECUTE_DDL` procedure.

When you use either the Advanced Replication interface or the `DBMS_REPCAT` package, all DDL statements are replicated to all of the sites participating in the replication environment. In some cases, you can also use export/import to create replicated objects.

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

Replication Conflicts

Asynchronous multimaster and updatable materialized view replication environments must address the possibility of replication conflicts that can 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 can occur in your replication environment, Advanced Replication offers a variety of prebuilt 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 prebuilt conflict resolution methods cannot resolve, then you have the option of building and using your own conflict resolution methods.

See Also:

- Chapter 5, "Conflict Resolution Concepts and Architecture" for information about how to design your database to avoid data conflicts and how to build conflict resolution methods that resolve such conflicts when they occur
- The Advanced Replication interface online Help in Oracle Enterprise Manager for instructions on using the interface to configure conflict resolution methods
- *Oracle Database Advanced Replication Management API Reference* for a description of how to build conflict resolution methods 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, then 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, then users can still query replicated tables, but no transactions can be completed until communication is reestablished. Also, it is possible to configure asynchronous replication so that it simulates synchronous replication.

See Also: "Scheduling Continuous Pushes" on page 6-18 for information about simulating synchronous replication in an asynchronous replication environment

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 replication environment. Procedural replication can occur asynchronously or synchronously.

Conflict Detection and Procedural Replication

When replicating 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:** *Oracle Database Advanced Replication Management API Reference* for more information about procedural replication

2

Master Replication Concepts and Architecture

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

This chapter contains these topics:

- Master Replication Concepts
- Master Replication Architecture

Master Replication Concepts

To understand the architectural details of master replication, you need to understand the 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.

This section contains these topics:

- What is Master Replication?
- Why Use Multimaster Replication?
- Multimaster Replication Process
- Conflict Resolution Concepts
- How Replication Works with Object Types and Collections

What is Master Replication?

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

Materialized views can be based on master tables at master sites or on materialized views at materialized view sites. When materialized views are based on materialized views, you have a multitier materialized view environment. In such an environment, materialized views that have other materialized views based on them are called master materialized views.

See Also: Chapter 3, "Materialized View Concepts and Architecture" for more information about multitier materialized views

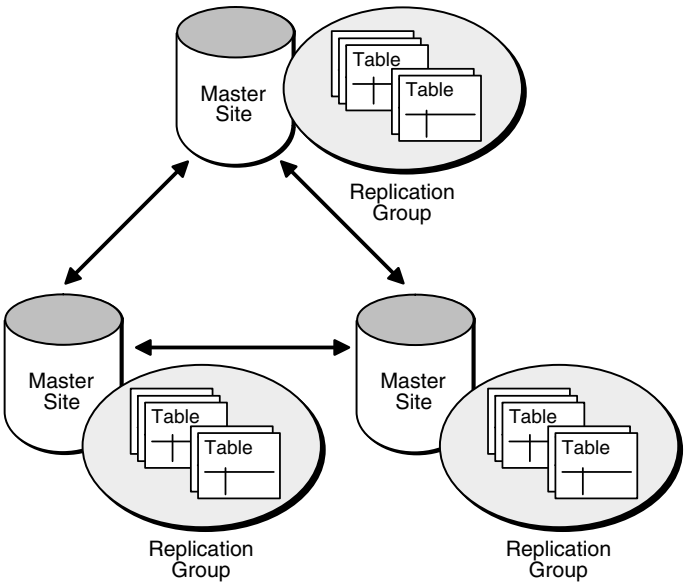
Multimaster Replication

Multimaster replication, also known as peer-to-peer or *n*-way replication, is comprised of multiple master sites equally participating in an update-anywhere model. Updates made to an individual master site are propagated (sent) to all other participating master sites. 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. 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.

If the replication environment is a hybrid environment (it has multiple master sites supporting one or more materialized view sites), then the target master site propagates any of the materialized view updates to all other master sites in the multimaster replication environment. Then each master site propagates changes to their materialized views when the materialized views refresh.

Figure 2–1 Multimaster Replication



Single Master Replication

A single master site can also function as the target master site for one or more materialized view sites. Unlike multimaster replication, where updates to a single site are propagated to all other master sites, materialized views update only their target master site.

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

> **See Also:** Chapter 3, "Materialized View Concepts and Architecture" for more information about materialized view replication with a master site

Master Sites

A master site can be both a node in a multimaster replication environment and the master for one or more materialized view sites in a single master or multimaster 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, one 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.

Each master group can have only one master definition site, 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 materialized view replication is by default the master definition site.

Why Use Multimaster Replication?

From a very basic point of view, replication is used to ensure that data is available when and where you need it. The following sections describe several different environments that have different information delivery requirements. Your replication environment might have 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 data in your database to establish a failover site should the primary site become unavailable due to system or network outages. Such a failover site can also serve as a fully functional database to support application access when the primary site is concurrently operational.

You can use Oracle Net to configure automatic connect-time failover, which enables Oracle Net 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_MODE` parameter to `on` and specifying multiple connect descriptors.

> **See Also:** *Oracle Database Net Services Administrator's Guide* for more information about configuring connect-time failover

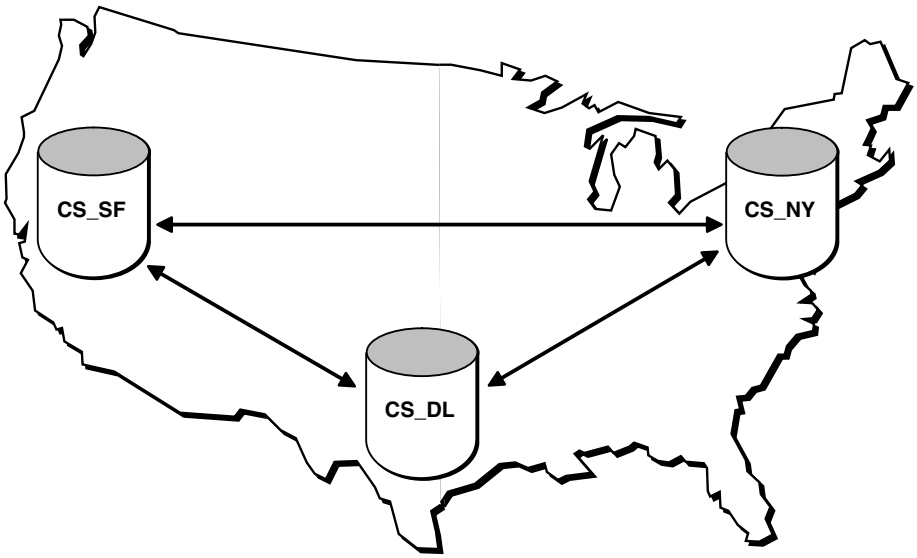
Load Balancing

Multimaster replication is useful for transaction processing applications that require multiple points of access to database information for the following purposes:

- Distributing a heavy application load
- Ensuring continuous availability
- 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



Support for Disconnected Materialized View Environments

Materialized view replication enables 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 materialized view environment. Master site support can be:

- A single master site supporting all of the materialized views, which reduces the possibility of divergent data because conflict resolution is performed only at master sites or master materialized view sites (in a multitier materialized view environment).
- A combination of multimaster and materialized view replication where groups of materialized views 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.

Oracle Real Application Clusters Compared with Replication

The two major areas where you need to consider whether Advanced Replication or Oracle Real Application Clusters (Oracle RAC) better serves your needs are load balancing and survivability.

- **Load Balancing**: Advanced Replication provides read load balancing over multiple databases, while Oracle RAC provides read and write load balancing over multiple instances. Because each write must be performed at each replication site, replication does not offer write load balancing.
- **Survivability**: Replication provides greater survivability protection with regards to natural disasters, power outages, or sabotage, or both because the remaining replication sites can be positioned in a geographically different region. Oracle

RAC 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.

- **Interoperability**: Advanced Replication can replicate data between different platforms and operating systems that are running Oracle. The instances in an Oracle RAC environment must run on the same platform.

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 data manipulation language (DML) statement or procedure fails at any site, then 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, then you must regenerate replication support for all master group objects. When you regenerate replication support, Oracle then activates the internal triggers and regenerates the internal packages to support replication of the objects at all master sites. Also, a multimaster replication environment can contain a mixture of both synchronous and asynchronous replication.

See Also: "Understanding Mixed-Mode Multimaster Systems" on page 2-32 for more information

Asynchronous Replication

Asynchronous replication independently propagates 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 can be different for a period of time before the changes have been propagated. Also, data conflicts can occur in an asynchronous replication environment.

The following describes the process of asynchronous replication:

1. A user issues DML statement or executes a wrapper for a replicated procedure.

   After a table has been set up for replication, any DML that a user commits 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)** and places it in the deferred transaction queue. The deferred transaction queue contains all deferred RPCs.

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

2. The 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. There is one deferred transaction queue for each site, and this one queue can be used by multiple replication groups.

3. 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. Each destination can have a different interval.

4. The deferred transaction queue entry applied at a remote destination.

   As a deferred transaction is being propagated to a target destination, each deferred RPC is applied at the destination site by calling an internal package. If the deferred transaction cannot be successfully applied at the destination site, then 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, then it is logged at the remote location and, optionally, a conflict resolution method is invoked.

5. When a deferred transaction has been successfully pushed to all remote master sites, it is not purged from the deferred transaction queue at the originating site immediately. It can be purged later by a purge job, which runs at a user-defined interval.

> **See Also:** Chapter 5, "Conflict Resolution Concepts and 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, then 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 2-32 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. For example, if a single master site of a six node multimaster environment is unavailable, then a transaction cannot be completed at any master site.

However, in asynchronous replication, the deferred transaction is held at the originating site until the downed site becomes available. Meanwhile, the transaction can be successfully propagated and applied at other replication sites.

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

> **See Also:** *Oracle Database Administrator's Guide* for more information about two-phase commit

The following describes the process of synchronous replication:

1. User issues DML statement or executes a wrapper for a replicated procedure.

   After a table has been set up for replication, any DML that a user commits 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, then the procedure call is captured for replication.

2. 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 roll back 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 materialized view site) that causes a data conflict at a destination site is automatically detected by the Oracle server at the destination site. Any data conflicts introduced by a materialized view site are detected and resolved at the target master site or master materialized view site of the materialized view.

For example, if the following master group is scheduled to propagate changes once an hour, then 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, then a conflict occurs.

In addition to the update conflict described previously, 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.

Conflict Type	Summary
Update conflict	Two or more DML statements are applied to the same row at different replication sites before the DML statement 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 can 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 and Architecture" for more information about the different types of data conflicts

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

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

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

If you choose to use Oracle-supplied or user-defined conflict resolution methods, then the Oracle server automatically tries to resolve the data conflict. The conflict resolution methods that you implement should conform to the business rules defined for your replication environment and should work to guarantee data convergence. You might need to modify tables to meet the needs of the conflict resolution methods you implement. For example, the latest time stamp conflict resolution method requires a time stamp column in the table on which it is implemented.

How Replication Works with Object Types and Collections

Oracle **object types** are user-defined data types that make it possible to model complex real-world entities such as customers and orders as single entities, called **objects**, in the database. You create object types using the `CREATE TYPE ... AS OBJECT` statement. You can replicate object types and objects between master sites in a multimaster replication environment.

An Oracle object that occupies a single column in a table is called a **column object**. Typically, tables that contain column objects also contain other columns, which can be built-in data types, such as `VARCHAR2` and `NUMBER`. An **object table** is a special kind of table in which each row represents an object. Each row in an object table is a **row object**.

You can also replicate **collections**. Collections are user-defined data types that are based on `VARRAY` and nested table data types. You create varrays with the `CREATE`

TYPE . . . AS VARRAY statement, and you create nested tables with the CREATE TYPE . . . AS TABLE statement.

Note: Advanced Replication does not support type inheritance or type evolution, and Advanced Replication does not support types created with the NOT FINAL clause. If a column of a replicated table or a replicated object table is based on a user-defined type, then you cannot alter the user-defined type.

See Also: *Oracle Database Object-Relational Developer's Guide* for detailed information about user-defined types, column objects, object tables, and collections. This section assumes a basic understanding of the information in that book.

Type Agreement at Replication Sites

User-defined types include all types created using the CREATE TYPE statement, including object, nested table, and VARRAY. To replicate schema objects based on user-defined types, the user-defined types themselves must exist, and must be exactly the same, at all replication sites.

When replicating user-defined types and the schema objects on which they are based, the following conditions apply:

- All replication sites must have the same object identifier (OID), schema owner, and type name for a replicated user-defined type.
- If the user-defined type is an object type, then all replication sites must agree on the order and data type of the attributes in the object type. You establish the order and data types of the attributes when you create the object type. For example, consider the following object type:

```
CREATE TYPE cust_address_typ AS OBJECT
     (street_address     VARCHAR2(40),
      postal_code        VARCHAR2(10),
      city               VARCHAR2(30),
      state_province     VARCHAR2(10),
      country_id         CHAR(2));
/
```

  At all replication sites, street_address must be the first attribute for this type and must be VARCHAR2(40), postal_code must be the second attribute and must be VARCHAR2(10), city must be the third attribute and must be VARCHAR2(30), and so on.

- All replication sites must agree on the hashcode of the user-defined type. Oracle examines a user-defined type and assigns the hashcode. This examination includes the type attributes, order of attributes, and type name. When all of these items are the same for two or more types, the types have the same hashcode. You can view the hashcode for a type by querying the DBA_TYPE_VERSIONS data dictionary view.

To ensure that a user-defined type is exactly the same at all replication sites, you must create the user-defined type in one of the following ways:

- Use the Replication Management API
- Use a CREATE TYPE Statement

- Use Export/Import

Use the Replication Management API Oracle recommends that you use the replication management API to create, modify, or drop any replicated object at a replication site, including user-defined types. If you do not use the replication management API for these actions, then replication errors can result. For example, to add a user-defined type that meets the conditions described previously to all replication sites in a master group, create the type at the master definition site and then use the `CREATE_MASTER_REPOBJECT` procedure in the `DBMS_REPCAT` package to add the type to a master group.

> **See Also:** *Oracle Database Advanced Replication Management API Reference*

Use a CREATE TYPE Statement You can use a `CREATE TYPE` statement at a replication site to create the type. It might be necessary to do this if you want to precreate the type at all replication sites and then add it to a replication group.

If you choose this option, then you must ensure the following:

- The type is in the same schema at all replication sites.
- The type has exactly the same attributes in exactly the same order at all replication sites.
- The type has exactly the same data type for each attribute at all replication sites.
- The type has the same object identifier at all replication sites.

You can find the object identifier for a type by querying the `DBA_TYPES` data dictionary view. For example, to find the object identifier (OID) for the `cust_address_typ`, enter the following query:

```
SELECT TYPE_OID FROM DBA_TYPES WHERE TYPE_NAME = 'CUST_ADDRESS_TYP';

TYPE_OID
--------------------------------
6F9BC33653681B7CE03400400B40A607
```

Or, if you are creating a new type at a number of different replication sites, then you can specify the same OID at each site during type creation. In this case, you can identify a globally unique OID by running the following query:

```
SELECT SYS_GUID() OID FROM DUAL;
```

When you know the OID for the type, complete the following steps to create the type at the replication sites where it does not exist:

1. Log in to the replication site as the user who owns the type. If this user does not exist at the replication site, then create the user.
2. Issue the `CREATE TYPE` statement and specify the OID:

```
CREATE TYPE oe.cust_address_typ OID '6F9BC33653681B7CE03400400B40A607'
     AS OBJECT (
     street_address     VARCHAR2(40),
     postal_code        VARCHAR2(10),
     city               VARCHAR2(30),
     state_province     VARCHAR2(10),
     country_id         CHAR(2));
/
```

The type is now ready for use at the replication site.

> **See Also:** *Oracle Database Advanced Replication Management API Reference*

Use Export/Import You can use the Export and Import utilities to maintain type agreement between replication sites. When you export object tables based on user-defined types, or tables containing column objects based on user-defined types, the user-defined types are also exported automatically, if the user performing the export has access to these types. When you import these tables at another replication site, the user-defined types are exactly the same as the ones at the site where you performed the export.

Therefore, you can use export/import to precreate your replication tables at new replication sites, and then specify the "use existing object" option when you add these tables to a replication group. This practice ensures type agreement at your replication sites.

> **See Also:** *Oracle Database Utilities* for information about export/import

Object Tables and Replication

When you replicate object tables, the following conditions apply:

- The OID of an object table must be the same at all replication sites.
- The OID of each row object in an object table must be the same at all replication sites.

You can meet these conditions by using the replication management API to add object tables to a replication group, modify object tables, and drop object tables from a replication group. For example, if you use the `CREATE_MASTER_REPOBJECT` procedure in the `DBMS_REPCAT` package to add an object table to a master group, then Oracle ensures that these conditions are met. You can also use export/import to precreate object tables at replication sites to meet these conditions.

Another option is to specify the OID for an object table when you create the object table at multiple replication sites. Complete the following steps if you want to use this option:

1. Query the `DUAL` view for a globally unique OID:

```
SELECT SYS_GUID() OID FROM DUAL;

OID
--------------------------------
81D98C325D4A45D0E03408002074B239
```

2. Create the `categories_tab` object table with the OID returned in Step 1 at each replication site:

```
CREATE TABLE oe.categories_tab5 OF oe.category_typ
   OID '81D98C325D4A45D0E03408002074B239'
   (category_id PRIMARY KEY);
```

Tables with Collection Columns

Collection columns are columns based on `VARRAY` and nested table data types. Oracle supports the replication of collection columns. When you add a table with a collection column to a replication group, the data in the collection column is replicated

automatically. If the collection column is a varray, then a varray larger than four kilobytes is stored as a BLOB.

If the collection column is a nested table, then Oracle performs row-level replication for each row in the nested table's storage table. For example, changes in five rows of a storage table result in five distinct remote procedure calls (RPCs), and five distinct conflict detection and optional resolution phases. The storage table can be stored as an index-organized table.

In addition, DML on a row that contains a nested table results in separate RPCs for the parent table and for each affected row in the nested table's storage table. Oracle does not perform referential integrity checks between the rows in the parent table and the rows in the storage table unless you explicitly specified integrity constraints during the creation of the parent table. Oracle recommends that you specify such constraints for replicated tables to detect all conflicts.

To ensure conflict detection between a nested table and its storage table, Oracle recommends that you define a deferrable foreign key constraint between them. Without a deferrable foreign key constraint, a conflict can insert rows in the storage table that cannot be accessed. A deferrable foreign key constraint causes an error to be raised in these situations so that the conflict is detected. You use the DEFERRED clause of the SET CONSTRAINTS statement to defer a constraint.

The following actions are not allowed directly on the storage table of a nested table in a replicated table:

- Adding the storage table to a replication group
- Altering the storage table
- Dropping the storage table
- Generating replication support on the storage table

These actions can occur indirectly when they are performed on the parent table of the storage table. In addition, you cannot replicate a subset of the columns in a storage table.

Tables with REF Columns

A REF is an Oracle built-in data type that is a logical "pointer" to a row object in an object table. A scoped REF is a REF that can only contain references to a specified object table, while an unscoped REF can contain references to any object table in the database. A scoped REF requires less storage space and provides more efficient access than an unscoped REF. Oracle supports the replication of tables with REFs.

Scoped REFs If a table with a scoped REF is replicated and the object table referenced by a REF is not replicated, then you must create the referenced object table at the sites where it does not exist before you begin replicating the table containing the scoped REF. Otherwise, replicating this table results in an error when the scoped REF cannot find the referenced object table. Typically, in this situation, it is best to replicate the referenced object table as well because it can become out of sync at the various replication sites if it is not replicated.

Unscoped REFs If a table with an unscoped REF is replicated and the object table referenced by the REF is not replicated, then a dangling REF might result at replicated sites if the REF cannot find the referenced object. For a replicated REF to be valid, the referenced object table must exist at each replication site.

REFs Created Using the WITH ROWID Option If the `WITH ROWID` option is specified for a `REF` column, then Oracle maintains a hint for the rowid of the row object referenced in the `REF`. Oracle can find the object referenced directly using the rowid contained in the `REF`, without the need to fetch the rowid from the OID index. The `WITH ROWID` option is not supported for scoped `REF`s.

Replicating a `REF` created using the `WITH ROWID` option results in an incorrect rowid hint at each replication site, except the site where the `REF` was first created or modified. The `ROWID` information in the `REF` is meaningless at the other sites, and Oracle does not correct the rowid hint automatically. Invalid rowid hints can cause performance problems. In this case, you can use the `VALIDATE STRUCTURE` option of the `ANALYZE TABLE` statement to determine which rowid hints at each replication site are incorrect.

> **See Also:** *Oracle Database SQL Language Reference* for more information about the `ANALYZE TABLE` statement

Master Replication Architecture

Although you can build a replication environment by following the procedures and examples described in the online Help for the Advanced Replication interface in Oracle Enterprise Manager and in the *Oracle Database Advanced 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.

This section contains these topics:

- Master Site Mechanisms
- Administrative Mechanisms
- Organizational Mechanisms
- Propagation Mechanism
- Performance Mechanisms
- Replication Protection Mechanisms
- Conflict Resolution Mechanisms

Master Site Mechanisms

To support a replication 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 can 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 requirements, then you can 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 or individual privileges or both.

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 has the following responsibilities:

- Building and maintaining the individual master replication groups
- Adding and removing participating master sites
- Managing the queues
- Controlling the state of the replication environment (normal and quiesced)

The default user name for this administrator is `repadmin`, but you can use any user name you wish.

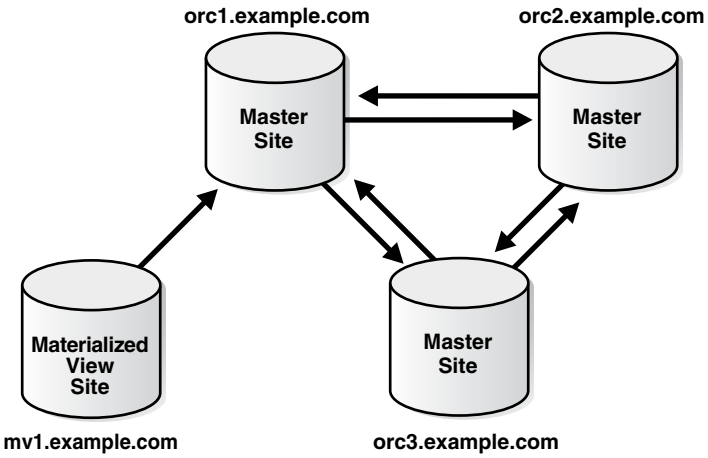
Propagator The propagator performs the task of propagating each transaction contained in the deferred transaction queue to the transaction's destinations. There is a single propagator for the database. In other words, it is possible for you to have multiple replication administrators to manage different schemas, but there can only be a single propagator for each 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, then the entire deferred transaction is placed in the error queue at the destination. You can register the receiver by using the `REGISTER_USER_REPGROUP` procedure in the `DBMS_REPCAT_ADMIN` package.

Database Links and Replication

Database links provide the conduit to replicate data between master sites and materialized view 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. Notice that for a materialized view site, only a link from the materialized view site to the master site is required. The master site does not need a database link to the materialized view site.

The most basic setup has 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 create public database links between all of the participating master sites, without specifying a `CONNECT TO` clause. The public database links specify the target of each database link with the `USING` clause, which specifies the service name of a remote database.

After creating the public database links, you can create the private replication administrator database links. When creating private database links, you specify the `CONNECT TO` clause, but the associated public database link 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:

- Multiple sets of private database links can share the same public link, further simplifying the administration of database links.
- If the target database of a database link changes but the service name for the target database remains the same, then you only need to change the `tnsnames.ora` file entry for the target database. Remember that the `USING` clause specifies the service name for the remote target database. All private database links for the same target point to the destination defined in the `USING` clause in the public database link.

  For example, if a database is moved to a different server but keeps the same service name, then you can update the `tnsnames.ora` file entry for the remote database at each replication site, and you do not need to re-create the database link.

As previously described, the replication administrator usually performs the tasks of administration and propagation in a multimaster environment. Because a single user

performs these tasks, only one 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 each of these alternate users.

See Also:

- *Oracle Database Administrator's Guide* for detailed information about database links and for information about creating database links
- *Oracle Database Net Services Administrator's Guide* for information about service names and the `tnsnames.ora` file

Database Links Created by the Advanced Replication Interface If you use a wizard in the Advanced Replication interface in the Oracle Enterprise Manager to set up your replication sites, then, by default, the wizard creates database links with a `USING` clause that contains the description of the service name in the `tnsnames.ora` file or the Oracle Management Server.

For example, suppose the `tnsnames.ora` file entry for a site is the following:

```
HQ.MYCOMPANY.COM =
'(DESCRIPTION=
   (ADDRESS=(PROTOCOL=TCP)(HOST=server1)(PORT=1521))
   (CONNECT_DATA=(SID=hqdb)(SERVER=DEDICATED)))'
```

Here, the service name is `HQ.MYCOMPANY.COM` and the description is the text after the first equal sign. The following statement shows an example of a database link to the `HQ.MYCOMPANY.COM` site created by the wizard:

```
CREATE PUBLIC DATABASE LINK "HQ.MYCOMPANY.COM" USING
'(DESCRIPTION=
   (ADDRESS=(PROTOCOL=TCP)(HOST=server1)(PORT=1521))
   (CONNECT_DATA=(SID=hqdb)(SERVER=DEDICATED)))'
```

The wizard uses the description of the service name and not the service name itself because different sites can have different information in their `tnsnames.ora` files. For example, if the wizard only used the service name and not the service name description, then the user would be required to ensure that the same service name exists and has the same information in the `tnsnames.ora` file at all sites, because there is no way for the Advanced Replication interface to check for this requirement.

By using the description for the service name, the wizard ensures that the database link is valid for all replication sites. The drawback to this type of database link is that, in the rare cases when service name description of a database changes, you must drop and re-create the database link. If the database link is created only with the service name and not the description, then you could change the `tnsnames.ora` file at all sites and retain the same database link.

Note: You can override the default behavior of the wizard by editing the customization screens of the wizard.

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 `ny` can have two public database links named `ny.example.com` that connect to the remote database using different paths.

- `ny.example.com@ethernet`, a link that connects to `ny` using an Ethernet link
- `ny.example.com@modem`, another link that connects to `ny` 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, then the scheduling characteristics for the propagation of the deferred transaction queue is the same for all master groups. This can 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:** *Oracle Database Administrator's Guide* 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.

Note: If you plan to use connection qualifiers, then you probably need to increase the value of the `OPEN_LINKS` initialization parameter at all master sites. The default is four open links for each process. Estimate the required value based on your usage. See "Initialization Parameters" on page 6-4, and see the *Oracle Database Reference* for more information about `OPEN_LINKS`.

Replication Objects

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

- Tables
- Indexes
- Packages and Package Bodies

- Procedures and Functions
- User-Defined Types and Type Bodies
- Triggers
- Views, Object Views, and Synonyms
- Indextypes
- User-Defined Operators

Tables In most cases, replicated tables are the foundation of your replication environment. After 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 Also:** "Internal Triggers" on page 2-21

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, then you have the option of using the existing object in your replication environment.

Note:

- On tables with self-referential integrity constraints, Advanced 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 *Oracle Database Advanced Replication Management API Reference* for information.
- When adding a master site to a master group that contains tables with circular dependencies or a table that contains a self-referential constraint, you must precreate the table definitions and manually load the data at the new master site. The following is an example of a circular dependency: Table A has a foreign key constraint on table B, and table B has a foreign key constraint on table A.
- When you drop a function-based index from a replicated table, or add a function-based index to a replicated table, you must regenerate replication support for the table.

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, then the table (or other objects) and the data remain 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.

> **See Also:** "Considerations for Replicated Tables" on page 6-1

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. You do not need to generate replication support for indexes.

Oracle supports the replication of domain indexes. You can replicate the definition of storage tables for domain indexes, but you cannot replicate the storage tables themselves because they typically contain `ROWID` information.

> **See Also:**
>
> - "Foreign Keys and Replicated Tables" on page 6-2 for information about replicating the index on a foreign key column
> - *Oracle Database Data Cartridge Developer's Guide* for more information about extensible indexes

Packages and 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 replication environment.

All parameters for a procedure with replication support must be `IN` parameters and must meet the data type requirements described in "Data Type Considerations for Replicated Tables" on page 6-2. `OUT` and `IN/OUT` modes are not supported.

A replicated procedure must be declared in a package. Standalone procedures cannot have replication support.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* for detailed information about using procedural replication

---

> **Note:** Similar to the concepts presented in the "Tables" on page 2-18, 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 and Functions Procedures and functions not declared as part of a package cannot have replication support. Though you cannot create a procedural replication environment with standalone procedures and functions, you can still use replication to distribute these standalone procedures and functions to the sites in your replication environment. When the standalone 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.

User-Defined Types and Type Bodies To replicate schema objects with user-defined types, the user-defined types must exist on all replication sites and be exactly the same at all replication sites.

> **See Also:** "How Replication Works with Object Types and Collections" on page 2-8

Triggers To ensure 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 time stamp 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 OR REPLACE TRIGGER hr.insert_time
2)    BEFORE
3)       INSERT OR UPDATE ON hr.employees FOR EACH ROW
4)    BEGIN
5)       IF DBMS_REPUTIL.FROM_REMOTE = FALSE THEN
6)          :NEW.TIMESTAMP := SYSDATE;
7)       END IF;
8)    END;
9) /
```

If the `DBMS_REPUTIL.FROM_REMOTE` function determines that the insert or update was locally initiated, then the defined action (that is, assign time stamp) occurs. If this function determines that the insert or update is from a remote site, then the time stamp value is not assigned. This example assumes that a `timestamp` column was added to the `hr.employees` table.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* for more information about creating replicated triggers

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

Indextypes Oracle supports the replication of indextypes. You must explicitly replicate the type and type body functions that you use to implement an indextype, either using the Advanced Replication interface in Enterprise Manager or the `CREATE_MASTER_REPOBJECT` procedure in the `DBMS_REPCAT` package.

> **See Also:** *Oracle Database Data Cartridge Developer's Guide* for more information about extensible indexes

User-Defined Operators Developers of object-oriented applications can extend the list of built-in relational operators (for example, `+`, `-`, `/`, `*`, `LIKE`) with domain specific operators (for example, `Contains`, `Within_Distance`, `Similar`) called user-defined operators. When you replicate a user-defined operator, you are simply

using replication to distribute the operator to the other master sites that are involved in the replication environment. After the object is replicated to the other sites, it operates as if the operator was created locally. No internal trigger or package monitors the object to capture any changes. Because it is a replicated object, though, you can still drop or modify it using the replication management API.

See Also: *Oracle Database Data Cartridge Developer's Guide*

Alternatives to Replicating Sequences

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

Three alternatives to replicating sequences guarantee the generation of unique values and avoid any uniqueness data conflicts. You can retrieve a unique identifier by executing the following select statement:

```
SELECT SYS_GUID() OID FROM DUAL;
```

This SQL statement 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 2-7.

Additionally, you can create a sequence at each of the master sites so that each site generates a unique value in your replication environment. You can accomplish this by using a combination of starting, incrementing, and maximum values in the `CREATE SEQUENCE` statement. 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.

Internal Triggers

Oracle uses internal triggers to capture and store information about updates to replicated data. Internal triggers build remote procedure calls (RPCs) to reproduce data changes made 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 previously. 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 Advanced 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 queuing mechanism.

Note: When the restricted session is enabled by the SQL statement `ALTER SYSTEM` with the `ENABLE RESTRICTED SESSION` clause, deferred transactions are not propagated. When the restricted session is disabled, they are propagated.

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, you can either reexecute 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 materialized view refresh groups.

Administrative Mechanisms

Several mechanisms are required to handle the administrative tasks that are often performed to support a replication environment. These mechanisms enable 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 replication environment.

Replication Modes of Operation

There are three modes of operation for a replication environment.

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

Quiescing Quiescing is the mode that transfers a replication 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 (such as adding and removing replicated objects). Replication is "stopped" in this mode. 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 after 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, then materialized views based on those master tables cannot propagate their changes to the target master tables, but local changes to the materialized view can continue.

A replication environment is quiesced on a master group level. All master sites participating in the 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. Users can still query tables that belong to a quiesced master group.

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), then you should verify that the administrative request queue (`DBA_REPCATLOG`) is empty before executing the resume mechanism.

Administrative 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. The Advanced Replication interface in Oracle Enterprise Manager also uses the procedures and functions of each site's replication management API to perform work.

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

The Administrative Request Mechanisms When you use the Advanced Replication interface in Enterprise 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 synchronously. If a synchronous broadcast fails for any reason, then Oracle returns an error message and rolls back the encompassing transaction.

When an Oracle server receives an administrative 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 administrative requests for a master group at a master site, you might observe requests that are waiting for a callback from another master site. These requests are called AWAIT_CALLBACK requests. Master replication activity cannot resume until all of the administrative requests in the DBA_REPCATLOG view have been applied and any errors resolved.

Whenever you use the Advanced Replication interface in Enterprise Manager to create an administrative 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. Oracle automatically determines how often the background process wakes up. You can experience a delay if you do not have enough background processes available to execute the outstanding jobs.

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 administrative 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 administrative 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, then DO_DEFERRED_REPCAT_ADMIN stops executing administrative 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 administrative request at the master definition site, can be deferred and does not prevent DO_DEFERRED_REPCAT_ADMIN from executing additional requests.

The success or failure of an administrative request at each master site is noted in the DBA_REPCATLOG view at each site. For each master group, the Advanced Replication interface in Enterprise Manager displays the corresponding status of each administrative request. Ultimately, each master site propagates the status of its administrative requests to the master definition site. If a request completes successfully at a master site, then 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, then 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, then Oracle removes the request at the master site and updates the corresponding AWAIT_CALLBACK request at the master definition site with ERROR status and 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 replication support, then 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. The Advanced Replication interface in Enterprise Manager displays the status of all administrative requests. Additionally, the DBA_REPCATLOG view contains interim status and any asynchronous error messages generated by the request.

Any materialized view sites that are affected by a DDL change are updated the next time you perform a refresh of the materialized view site. While all master sites can communicate with one another, materialized view sites can communicate only with their associated master site.

If you must alter the shape of a materialized view as the result of a change to its master, then you must drop and re-create the materialized view.

Administrative Request Queue

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

You can view the administrative request queue by querying the DBA_REPCATLOG view or viewing the Administrative Requests page in the Advanced Replication interface in Enterprise Manager.

Each request has a status that displays the state of the request. Here are the possible states:

- **READY**: The READY state indicates that the request is ready to be executed. If you monitor the administrative request queue and a request remains in the READY state for a long time, then a request in front of the ready request might be waiting for a callback. Typically, administrative requests in the READY state are waiting for a job to execute them. You can execute them manually by using the DO_DEFERRED_REPCAT_ADMIN procedure in the DBMS_REPCAT package.
- **AWAIT_CALLBACK**: The AWAIT_CALLBACK state indicates that the request is waiting for a request to be executed at another site and is waiting for confirmation of the request execution. After the request receives the callback, the request is either removed or has its status changed. The request is removed from the queue if it was applied successfully, or its status is changed to ERROR if it failed. This state is only possible at the master definition site.
- **ERROR**: If a request cannot be successfully executed, then it is placed in the ERROR state. The error number appears in the ERRNUM column and the error message appears in the MESSAGE column of the administrative request queue (or ERROR in the **Status** field on the Administrative Requests page when using the Advanced Replication interface in Oracle Enterprise Manager).

  Note: If a request is in the ERROR state, then resolve the error condition as described by the error number and resubmit the request.

- **DO_CALLBACK**: If a request at a master site is in the DO_CALLBACK state, then it means that the master site must inform the master definition site about the success or failure of the request. This state is only possible at a master site that is not the master definition site.

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

Note: When the restricted session is enabled by the SQL statement ALTER SYSTEM with the ENABLE RESTRICTED SESSION clause, administrative requests are not executed. When the restricted session is disabled, they are executed.

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

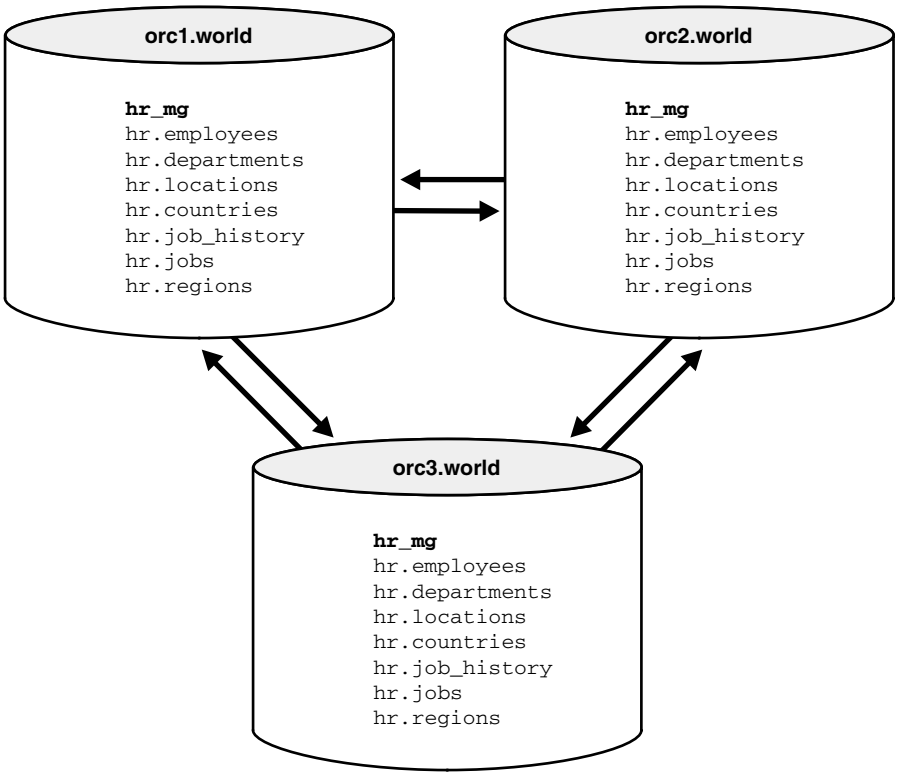
In a replication environment, Oracle manages replication objects using **replication groups**. A replication group is a collection of replication objects that are always updated in a transactionally consistent manner.

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 2-17). Figure 2–4 illustrates that master group `hr_mg` contains an exact replica of the replicated objects at each master site.

Figure 2–4 Master Group hr_mg Contains Same Replication Objects at All Sites

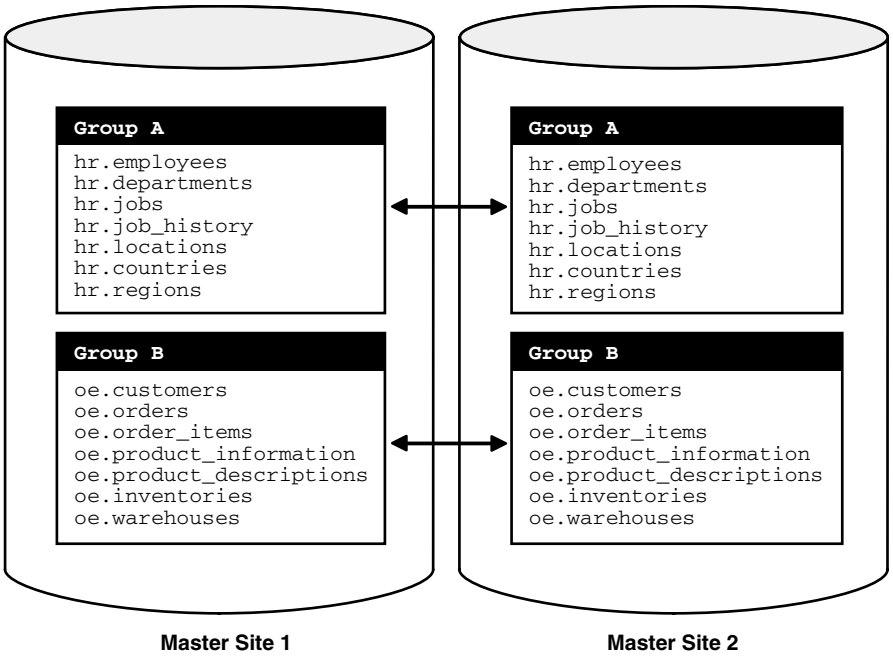


The master group organization at the master site plays an integral role in the organization of replication objects at a materialized view site.

> **See Also:** "Organizational Mechanisms" on page 3-41 for more information about the organizational mechanisms at a materialized view site

Additionally, Figure 2–5 illustrates that each site can 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 organizational 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, then the remainder of the group's columns can 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, then the `timestamp` field can 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. Although this makes setup and administration easier, it might decrease the performance of your replicated table and might increase the potential for data conflicts. As described in the "Performance Mechanisms" on page 2-33, if a conflict occurs in one column group of a table, then the minimum communication feature does not send data from other column groups in the table. Therefore, placing all columns into a single column group might negate the advantages of the minimum communication feature, unless you use the `SEND_OLD_VALUES` and `COMPARE_OLD_VALUES` procedures in the `DBMS_REPCAT` package.

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

Propagation Mechanism

Propagation 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

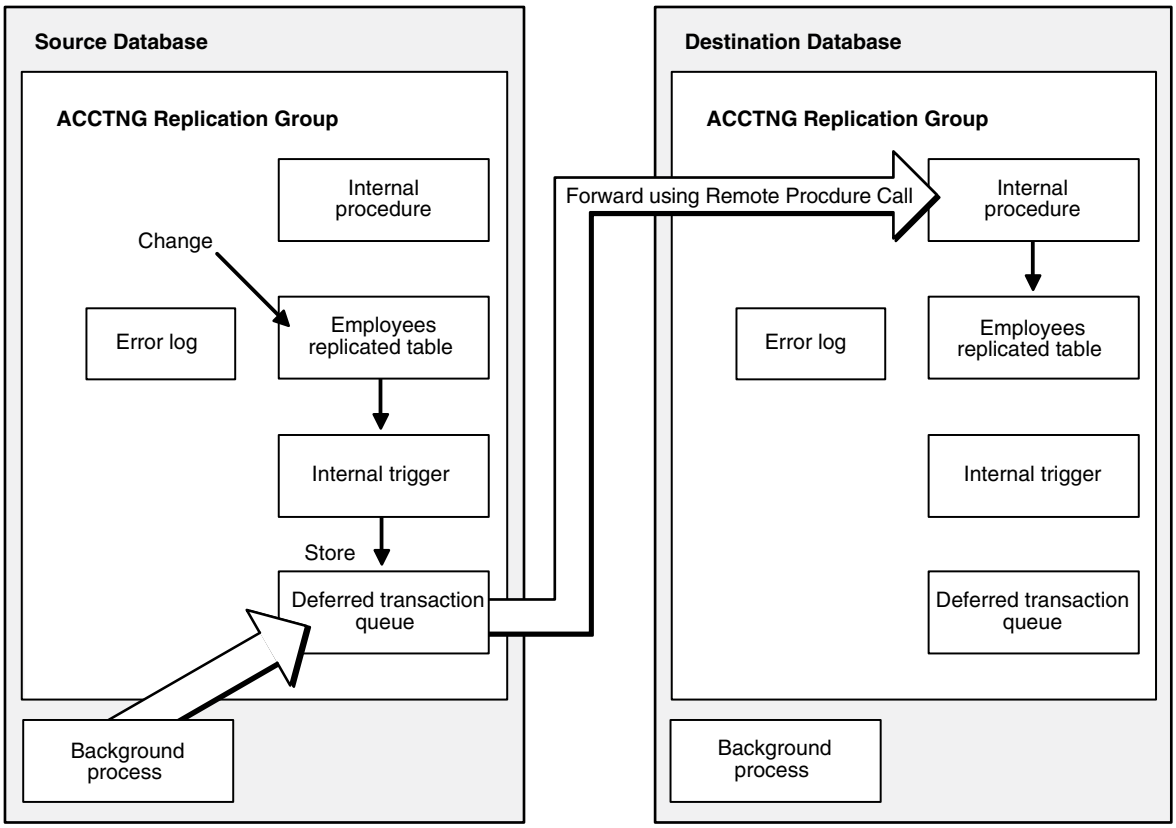
As the internal trigger captures any DML applied to a replicated table, the DML must be **propagated** (or sent) to the other master sites in the replication environment. Internal triggers are described in the section "Internal Triggers" on page 2-21.

Advanced 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**.

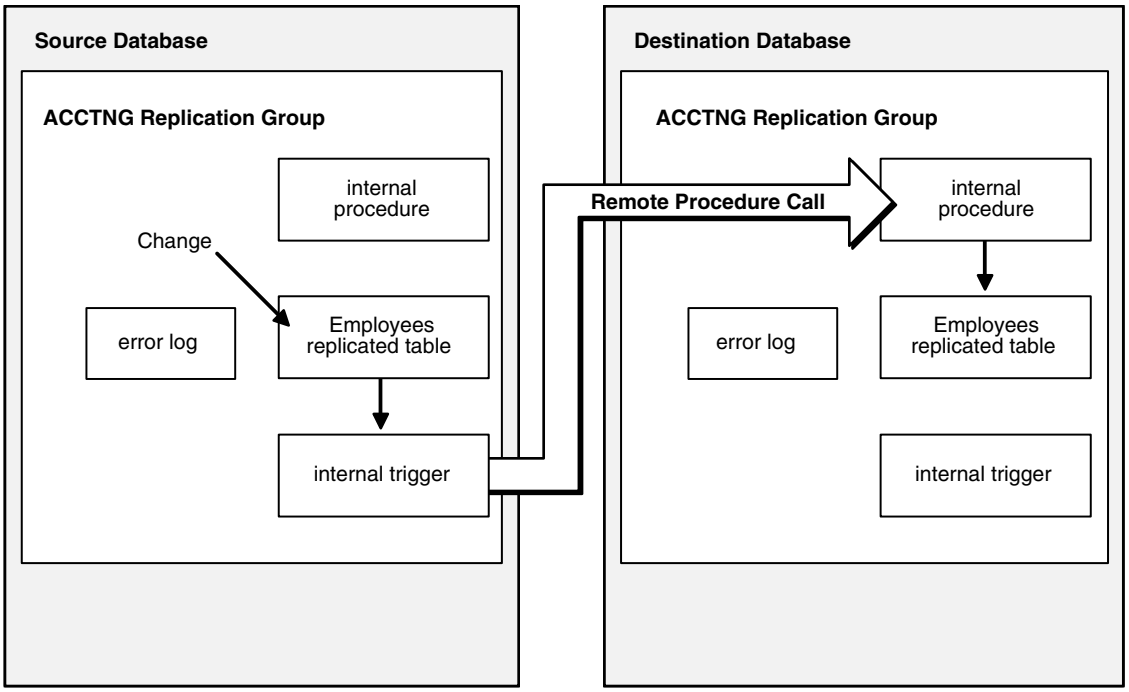
As Figure 2–6 shows, Oracle uses its internal triggers, deferred transactions, deferred transaction queues, and job queues to propagate data-level changes asynchronously among master sites in a replication environment, as well as from an updatable materialized view to its master table.

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



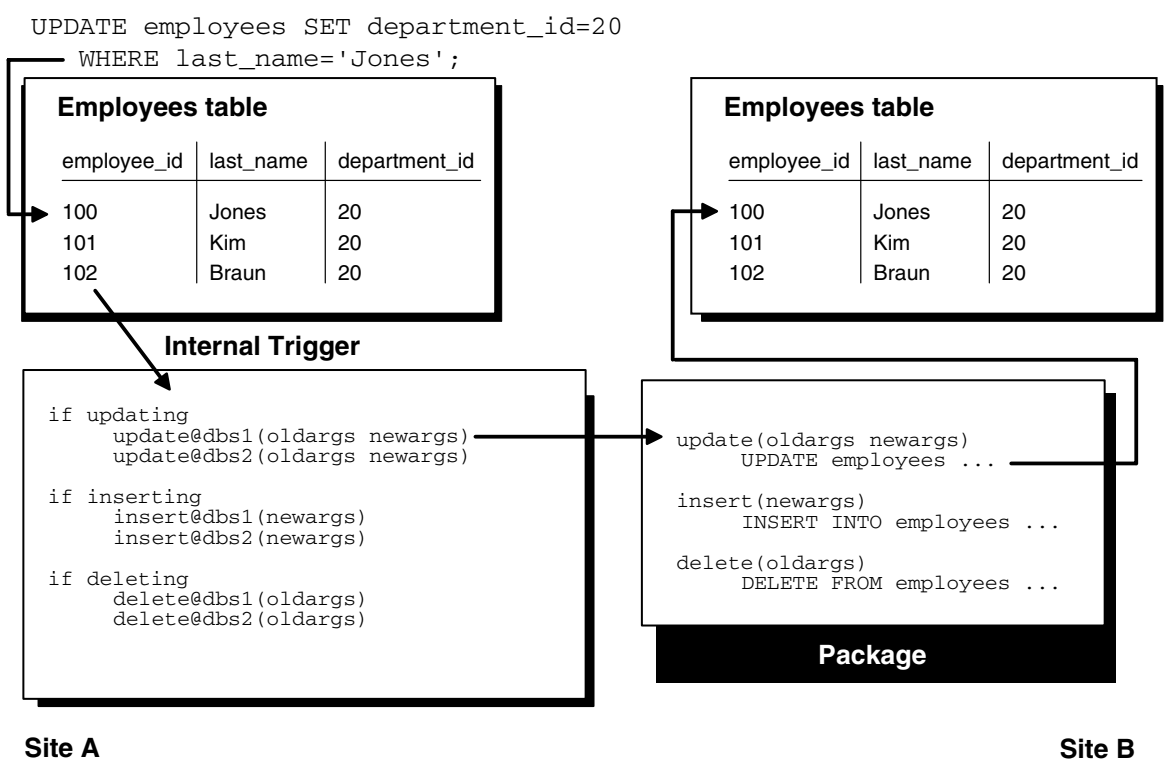
As Figure 2–7 shows, Oracle uses the same internal triggers to generate remote procedure calls (RPCs) that asynchronously replicate data-level changes to other replication sites to support synchronous, row-level data replication. However, Oracle does not defer the execution of such RPCs. Instead, data 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.

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 replication 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 Also:** *Oracle Database Administrator's Guide* for more information about distributed transactions

Figure 2–8 Propagating Changes Using Synchronous Row-Level Replication



Restrictions Because of the locking mechanism used by synchronous replication, deadlocks can occur when the same row is updated at two different sites at the same time. 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, then applications should never experience replication conflicts. However, if even one site is sending changes asynchronously to another site, then applications can experience conflicts at any site in the replication environment.

If the change is being propagated synchronously, then an error is raised and a rollback is required. If the change is propagated asynchronously, then Oracle automatically detects the conflicts and either logs the conflict in the error queue or, if you designate an appropriate resolution method, resolves the conflict.

See Also: Chapter 5, "Conflict Resolution Concepts and Architecture"

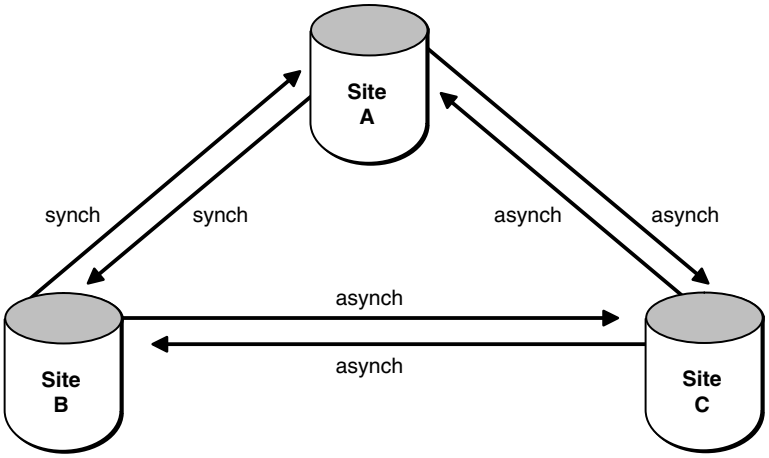
Understanding Mixed-Mode Multimaster Systems

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, then site A sends changes to site B synchronously and site B sends changes to site A synchronously. There is no need to be concerned about 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.

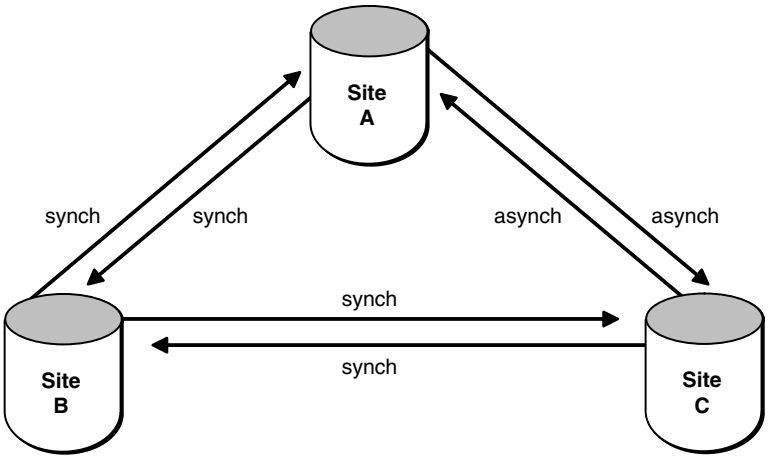
Figure 2–9 Selecting a Propagation Mode



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, then you can use the following methods to propagate the deferred transactions:

- **Scheduled job:** In most cases, use a scheduled job to automatically propagate the deferred transactions at a set interval.
- **Manual propagation:** You can also manually propagate the changes by executing a stored procedure or using the Advanced Replication interface in Oracle Enterprise Manager. You might 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. Advanced Replication provides several mechanisms to help increase the performance of your replication 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 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 processes used to push the queue remain dedicated to the push until it is complete.

To configure a pool of parallel 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 processes. Therefore, you do not need to adjust the size of any server's pool of parallel processes to account for replication. Typically, serial propagation is used only for backward compatibility.
- When you configure one or more scheduled links to use parallel propagation, you must consider the number of parallel 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 processes used to push transactions to its destination. Therefore, you should increase the number of parallel 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 6-4
- *Oracle Database Reference*

Implementing Parallel Propagation For most users, setting the parallel propagation parameter to a value of 1 provides sufficient performance. A setting of 1 enables the optimized data transfer method discussed in the previous section instead of serial propagation. However, some users might 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, then you have implemented parallel propagation, and you do not need to complete the remaining steps in this procedure.

   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 processes.

3. If you want to try to achieve greater propagation throughput than with a value of 1, then set your parallel propagation value to 2.

4. Test your database environment and carefully measure the propagation throughput.

   In many cases, you might 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 processes 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, then 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 2-35 to learn about techniques to reduce transaction dependencies

   If your performance did improve with a value of 2, 4, or 8, then it suggests that your transactions have a low degree of interdependence. You can 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 processes.

5. Set parallel propagation to the value that offers the best performance in your environment based on your testing.

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

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."
- Consider using row-level dependency tracking.

  > **See Also:** "Use of Row-Level Dependency Tracking to Improve Parallelism" on page 2-39

Minimum Communication

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. By default, Oracle minimizes the amount of data that must be communicated to detect conflicts for each changed row in the table. Specifically, Oracle propagates:

- The primary key value 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.
- Ensure that your replication environment uses minimum communication by ensuring that the `min_communication` parameter is set to the default value of `TRUE` when you run the procedures `CREATE_MVIEW_REPOBJECT` and `GENERATE_REPLICATION_SUPPORT` in the `DBMS_REPCAT` package.

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.

When you are pushing deferred transactions, you set the `delay_seconds` parameter in the `SCHEDULE_PUSH` procedure or the `PUSH` function. When you are purging deferred transactions, you set the `delay_seconds` parameter in the `SCHEDULE_PURGE` procedure or the `PURGE` function. These procedures and functions are in the `DBMS_DEFER_SYS` package.

The `delay_seconds` parameter 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, then 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, then 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, then 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, then 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.

Why not 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. Keep the length of the `delay_seconds` parameter in check for the following reasons:

- **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 might keep the parallel process unavailable for other operations. To use parallel propagation, you

set the parallelism parameter to 1 or higher in the SCHEDULE_PUSH procedure or the PUSH function.

- **Serial Propagation**: If you are using serial propagation (not parallel propagation), then the delay_seconds value causes the open session to "sleep" for the entire length of the delay, providing none of the benefits earlier described. To use serial propagation, you set the parallelism parameter to 0 (zero) in the SCHEDULE_PUSH procedure or the PUSH function.
- **Precise Purge**: If you specify the purge_method_precise method when using the DBMS_DEFER_SYS.PURGE procedure and you have defined a large delay_seconds value, then you might experience performance degradation when performing the specified purge. Using purge_method_precise is more expensive than the alternative (purge_method_quick), but it ensures that the deferred transactions and procedure calls are purged after they have been successfully pushed.

As a general rule of thumb, there are few viewable benefits of setting the delay_seconds parameter to a value greater than 20 minutes (which is 1200 seconds for the parameter setting).

Additionally, if you are using serial propagation by setting the parallelism parameter to 0, then you probably do not want to set a large delay_seconds value. Unlike parallel propagation, serial propagation only checks the queue after the duration of the delay_seconds value has elapsed. If you use serial propagation and set delay_seconds to 20 minutes, then the scheduled job sleeps for the entire 20 minutes, and any deferred transactions that enter the deferred transaction queue during that time are not pushed until 20 minutes have elapsed. Therefore, if you are using serial propagation, then consider setting delay_seconds to a value of 60 seconds or lower.

If you set a value of 20 minutes for parallel propagation, then the parallel push checks once a minute. If you can afford this resource lock, then the relatively high delay_seconds value of 20 minutes is probably most efficient in your environment. If, however, you cannot afford this resource lock, then consider setting the delay_seconds value to 10 or 20 seconds. Although you must 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 the default value of 0 seconds).

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. Oracle protects transactions in the following ways:

- Multiple procedure calls submitted within a single local transaction are executed within a transaction remotely.
- If the network or remote database fails during propagation, then the transaction is rolled back at the remote site and the transaction remains in the local queue at the originating site until the remote database becomes accessible again and the transaction 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. Even after the transaction is successfully propagated to all destination sites, it remains in the queue until the purge job removes it.

In the case of parallel propagation, replication uses a special-purpose distributed transaction protocol optimized for propagation. The remote site keeps track of the

transactions that have been propagated successfully and sends this information back to the local site when it is requested. 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 about the transactions that have been propagated successfully so that propagation can continue at the appropriate point.

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 is logged at the remote site as an error transaction.

See Also:

- "Parallel Propagation" on page 2-33
- The Advanced Replication interface online Help for more information about viewing and managing error transactions with the Advanced Replication interface in Oracle Enterprise Manager

Data Propagation Dependency Maintenance

Oracle maintains dependency ordering when propagating replicated transactions to remote sites. For example, consider the following transactions:

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

Transaction B depends on transaction A because transaction B sees the committed update canceling 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 executes a new transaction on the local system, Oracle completes the following process:

1. Oracle records the system change number (SCN) of the most recent transaction that updates data that is seen by the new transaction as the dependent SCN. You can record the SCN either at the data block level or at the row level, as discussed later in this chapter.
2. Oracle ensures that transactions with SCNs less than or equal to the dependent SCN propagate successfully to the remote system.
3. Oracle propagates the waiting, 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 transactions 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 and 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 site 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 2-17

Use of Row-Level Dependency Tracking to Improve Parallelism When you create a table, you can specify the following options for tracking system change numbers (SCN)s:

- `NOROWDEPENDENCIES`, the default, specifies that the SCN is tracked at the data block level.
- `ROWDEPENDENCIES` specifies that the SCN is tracked for each row in the table.

When you use the `NOROWDEPENDENCIES` clause for a table, the data block SCN tracks the most recent update of a row that is stored in the data block. Other rows that were updated earlier can be stored in the same data block, but information about when these rows were updated is lost when a new SCN is applied at the data block level.

When you use the `ROWDEPENDENCIES` clause for a table, multiple SCNs can be stored in a single data block. That is, a separate SCN tracks changes for each row that is stored in the data block. If two rows that are stored in the same data block are changed by different transactions, then each row has an SCN that tracks the change. To track the SCN at the row level, each row in the table uses an additional six bytes of storage space.

Using the `ROWDEPENDENCIES` clause for a table enables parallel propagation to track dependencies and order changes more efficiently when applying the deferred transaction queue. This increased efficiency improves performance and provides greater scalability in replication environments.

You can use the following query to list the tables that are using the `ROWDEPENDENCIES` clause currently:

```
SELECT OWNER, TABLE_NAME FROM DBA_TABLES
  WHERE DEPENDENCIES = 'ENABLED';
```

See Also: "Row-Level Dependency Tracking" on page 6-4 for information about creating a table using the `ROWDEPENDENCIES` clause

Minimize Transaction Dependencies to Improve Parallelism If you did not use the `ROWDEPENDENCIES` clause for some of your replicated tables, then you can improve the performance of parallel propagation for these tables by minimizing transaction dependencies.

In this case, 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, 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.

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 "Send and Compare Old Values" on page 5-27.

Row Identification 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 uniquely identify rows in the table during data replication.

> **Caution:** Do not permit applications to update the primary key or alternate key columns of a table. This ensures that Oracle can identify rows and preserve the integrity of replicated data.

Resolution of Data Conflicts

Oracle provides a mechanism that enables you to define a conflict resolution method that resolves a data conflict when detected. Oracle provides several prebuilt conflict resolution methods:

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

If the prebuilt Oracle conflict resolution methods do not meet the needs of your replication environment, then 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 and Architecture" to learn how conflict resolution works.

> **See Also:** The online Help for the Advanced Replication interface to learn how to implement conflict resolution with Oracle Enterprise Manager, and see the *Oracle Database Advanced Replication Management API Reference* to learn how to implement conflict resolution using the replication management API.

3

Materialized View Concepts and Architecture

This chapter explains the concepts and architecture of Oracle materialized views.

This chapter contains these topics:

- Materialized View Concepts
- Materialized View Architecture

Materialized View Concepts

Oracle uses **materialized views** (also known as snapshots in prior releases) to replicate data to nonmaster sites in a replication environment and to cache expensive queries in a data warehouse environment. This chapter, and this *Oracle Database Advanced Replication* manual in general, discusses materialized views for use in a replication environment.

This section contains these topics:

- What is a Materialized View?
- Why Use Materialized Views?
- Read-Only, Updatable, and Writeable Materialized Views
- Available Materialized Views
- Required Privileges for Materialized View Operations
- Data Subsetting with Materialized Views
- Determining the Fast Refresh Capabilities of a Materialized View
- Multitier Materialized Views
- How Materialized Views Work with Object Types and Collections
- Materialized View Registration at a Master Site or Master Materialized View Site

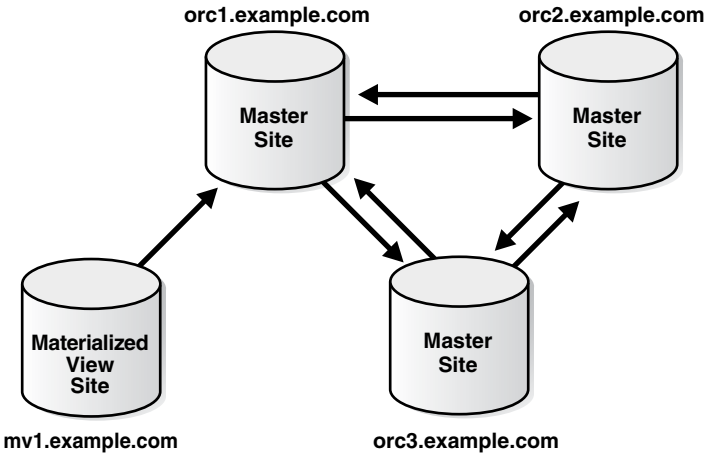
> **See Also:** *Oracle Database Data Warehousing Guide* to learn more about materialized views for data warehousing

What is a Materialized View?

A materialized view is a replica of a target master from a single point in time. The master can be either a master table at a master site or a master materialized view at a materialized view site. Whereas in multimaster replication tables are continuously updated by other master sites, materialized views are updated from one or more masters through individual batch updates, known as a **refreshes**, from a single master

site or master materialized view site, as illustrated in Figure 3–1. The arrows in Figure 3–1 represent database links.

Figure 3–1 Materialized View Connected to a Single Master Site



When a fast refresh is performed on a materialized view, Oracle must examine all of the changes to the master table or master materialized view since the last refresh to see if any apply to the materialized view. Therefore, if any changes where made to the master since the last refresh, then a materialized view refresh takes some time to apply the changes to the materialized view. If, however, no changes at all were made to the master since the last refresh of a materialized view, then the materialized view refresh should be very quick.

Why Use Materialized Views?

You can use materialized views to achieve one or more of the following goals:

- Ease Network Loads
- Create a Mass Deployment Environment
- Enable Data Subsetting
- Enable Disconnected Computing

Ease Network Loads

If one of your goals is to reduce network loads, then you can use materialized views 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. Through the use of multitier materialized views, you can create materialized views based on other materialized views, which enables you to distribute user load to an even greater extent because clients can access materialized view sites instead of master sites. To decrease the amount of data that is replicated, a materialized view can be a subset of a master table or master materialized view.

While multimaster replication also distributes a corporate database among multiple sites, the networking requirements for multimaster replication are greater than those for replicating with materialized views because of the transaction by transaction nature of multimaster replication. Further, the ability of multimaster replication to provide real-time or near real-time replication can result in greater network traffic, and might require a dedicated network link.

Materialized views are updated through an efficient batch process from a single master site or master materialized view site. They have lower network requirements and dependencies than multimaster replication because of the point in time nature of materialized view replication. Whereas multimaster replication requires constant communication over the network, materialized view replication requires only periodic refreshes.

In addition to not requiring a dedicated network connection, replicating data with materialized views 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.

Create a Mass Deployment Environment

Deployment templates enable you to precreate a materialized view environment locally. You can then use deployment templates to quickly and easily deploy materialized view environments to support sales force automation and other mass deployment environments. Parameters enable you to create custom data sets for individual users without changing the deployment template. This technology enables you to roll out a database infrastructure to hundreds or thousands of users.

Enable Data Subsetting

Materialized views enable you to replicate data based on column- and row-level subsetting, while multimaster replication requires replication of the entire table. Data subsetting enables you to replicate information that pertains only to a particular site. For example, if you have a regional sales office, then you might replicate only the data that is needed in that region, thereby cutting down on unnecessary network traffic.

Enable Disconnected Computing

Materialized views do not require a dedicated network connection. Though you have the option of automating the refresh process by scheduling a job, you can manually refresh your materialized view on-demand, which 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 orders, the salesperson simply dials up the network and uses the integrated mechanism to refresh the database, thus transferring the orders to the main office.

Read-Only, Updatable, and Writeable Materialized Views

A materialized view can be either read-only, updatable, or writeable. Users cannot perform data manipulation language (DML) statements on read-only materialized views, but they can perform DML on updatable and writeable materialized views.

Note:

- For read-only, updatable, and writeable materialized views, the defining query of the materialized view must reference all of the primary key columns in the master.
- Materialized views do not support columns that have been encrypted using transparent data encryption.

See Also:

- "Materialized View Replication" on page 1-5 for an introduction to read-only and updatable materialized views
- "Considerations for Replicated Tables" on page 6-1

Read-Only Materialized Views

You can make a materialized view read-only during creation by omitting the `FOR UPDATE` clause or disabling the equivalent option in the Advanced Replication interface in Oracle Enterprise Manager. Read-only materialized views use many of the same mechanisms as updatable materialized views, except that they do not need to belong to a materialized view group.

In addition, using read-only materialized views eliminates the possibility of a materialized view introducing data conflicts at the master site or master materialized view site, although this convenience means that updates cannot be made at the remote materialized view site. The following is an example of a read-only materialized view:

```
CREATE MATERIALIZED VIEW hr.employees AS
  SELECT * FROM hr.employees@orc1.example.com;
```

Updatable Materialized Views

You can make a materialized view updatable during creation by including the `FOR UPDATE` clause or enabling the equivalent option in the Advanced Replication interface in Oracle Enterprise Manager. For changes made to an updatable materialized view to be pushed back to the master during refresh, the updatable materialized view must belong to a materialized view group.

Updatable materialized views enable you to decrease the load on master sites because users can make changes to the data at the materialized view site. The following is an example of an updatable materialized view:

```
CREATE MATERIALIZED VIEW hr.departments FOR UPDATE AS
  SELECT * FROM hr.departments@orc1.example.com;
```

The following statement creates a materialized view group:

```
BEGIN
   DBMS_REPCAT.CREATE_MVIEW_REPGROUP (
      gname => 'hr_repg',
      master => 'orc1.example.com',
      propagation_mode => 'ASYNCHRONOUS');
END;
/
```

The following statement adds the `hr.departments` materialized view to the materialized view group, making the materialized view updatable:

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

You can also use the Advanced Replication interface in Oracle Enterprise Manager to create a materialized view group and add a materialized view to it.

In a single master site environment that has updatable materialized views, quiesce is not required when you perform administration operations on the master site if you:

- Propagate all of the deferred transactions at the databases containing updatable materialized views before you perform the administration operations to the master group.
- Do not allow any database manipulation language (DML) changes on the updatable materialized views until you have finished the administration operation on the master site and regenerated replication support for the materialized view.

If you do not perform these actions, then quiesce the master group before you perform the administration operations on it.

Note:

- Do not use column aliases when you are creating an updatable materialized view. Column aliases cause an error when you attempt to add the materialized view to a materialized view group using the `CREATE_MVIEW_REPOBJECT` procedure.
- An updatable materialized view based on a master table or master materialized view that has defined column default values does not automatically use the master's default values.
- A `DELETE CASCADE` constraint used with an updatable materialized view must be deferrable.

See Also:

- "Materialized View Groups" on page 3-41 for more information
- *Oracle Database SQL Language Reference* for more information about column aliases

Writeable Materialized Views

A writeable materialized view is one that is created using the `FOR UPDATE` clause but is not part of a materialized view group. Users can perform DML operations on a writeable materialized view, but if you refresh the materialized view, then these changes are not pushed back to the master and the changes are lost in the materialized view itself. Writeable materialized views are typically allowed wherever fast-refreshable read-only materialized views are allowed.

Note: Most of the documentation about materialized views only refers to read-only and updatable materialized views because writeable materialized views are rarely used.

Available Materialized Views

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

The following sections contain examples of creating different types of materialized views:

- Primary Key Materialized Views
- Object Materialized Views
- ROWID Materialized Views
- Complex Materialized Views

Whenever you create a materialized view, regardless of its type, always specify the schema name of the table owner in the query for the materialized view. For example, consider the following CREATE MATERIALIZED VIEW statement:

```
CREATE MATERIALIZED VIEW hr.employees
  AS SELECT * FROM hr.employees@orc1.example.com;
```

Here, the schema hr is specified in the query.

> **Note:** You cannot execute a distributed transaction on the master table of a refresh-on-commit materialized view. Refresh-on-commit materialized views are those created using the ON COMMIT REFRESH clause in the CREATE MATERIALIZED VIEW statement. You can execute a distributed transaction on the master table of a refresh-on-demand materialized view.

Primary Key Materialized Views

Primary key materialized views are the default type of materialized view. They are updatable if the materialized view was created as part of a materialized view group and FOR UPDATE was specified when defining the materialized view. An updatable materialized view must belong to a materialized view group that has the same name as the replication group at its master site or master materialized view site. In addition, an updatable materialized view must reside in a different database than the master replication group.

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 following is an example of a SQL statement for creating an updatable, primary key materialized view:

```
CREATE MATERIALIZED VIEW oe.customers FOR UPDATE AS
  SELECT * FROM oe.customers@orc1.example.com;
```

Primary key materialized views can contain a subquery so that you can create a subset of rows at the remote materialized view site. A subquery is a query imbedded within the primary query, so that you have more than one SELECT statement in the CREATE MATERIALIZED VIEW statement. This subquery can be as simple as a basic WHERE clause or as complex as a multilevel WHERE EXISTS clause. Primary key materialized views that contain a selected class of subqueries can still be incrementally (or fast) refreshed, if each master referenced has a materialized view log. A fast refresh uses materialized view logs to update only the rows that have changed since the last refresh.

The following materialized view is created with a WHERE clause containing a subquery:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST AS
 SELECT * FROM oe.orders@orc1.example.com o
 WHERE EXISTS
```

```
(SELECT * FROM oe.customers@orc1.example.com c
 WHERE o.customer_id = c.customer_id AND c.credit_limit > 10000);
```

This type of materialized view is called a subquery materialized view.

Note: To create this `oe.orders` materialized view, `credit_limit` must be logged in the master's materialized view log. See "Logging Columns in the Materialized View Log" on page 6-13 for more information.

See Also:

- "Materialized View Groups" on page 3-41 for more information about materialized view groups
- "Materialized Views with Subqueries" on page 3-13 for more information about materialized views with subqueries
- "Refresh Types" on page 3-45 for more information about fast refresh
- "Materialized View Log" on page 3-38 for more information about materialized view logs
- *Oracle Database SQL Language Reference* for more information about subqueries

Object Materialized Views

If a materialized view is based on an object table and is created using the `OF` *type* clause, then the materialized view is called an **object materialized view**. An object materialized view is structured in the same way as an object table. That is, an object materialized view is composed of row objects, and each row object is identified by an object identifier (OID) column.

See Also: "Materialized Views Based on Object Tables" on page 3-29

ROWID Materialized Views

Oracle supports `ROWID` materialized views in addition to the default primary key materialized views. A `ROWID` materialized view is based on the physical row identifiers (rowids) of the rows in a master. `ROWID` materialized views can be used for materialized views based on master tables that do not have a primary key, or for materialized views that do not include all primary key columns of the master tables.

The following is an example of a `CREATE MATERIALIZED VIEW` statement that creates a `ROWID` materialized view:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH WITH ROWID AS
 SELECT * FROM oe.orders@orc1.example.com;
```

See Also:

- "Materialized View Log" on page 3-38 for more information about the differences between a ROWID and primary key materialized view
- *Oracle Database SQL Language Reference* for more information about the WITH ROWID clause in the CREATE MATERIALIZED VIEW statement

Complex Materialized Views

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

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

- A CONNECT BY clause

  For example, the following statement creates a complex materialized view:

```
CREATE MATERIALIZED VIEW hr.emp_hierarchy AS
  SELECT LPAD(' ', 4*(LEVEL-1))||email USERNAME
    FROM hr.employees@orc1.example.com START WITH manager_id IS NULL
    CONNECT BY PRIOR employee_id = manager_id;
```

- An INTERSECT, MINUS, or UNION ALL set operation

  For example, the following statement creates a complex materialized view because it has a UNION ALL set operation:

```
CREATE MATERIALIZED VIEW hr.mview_employees AS
  SELECT employees.employee_id, employees.email
  FROM hr.employees@orc1.example.com
UNION ALL
  SELECT new_employees.employee_id, new_employees.email
  FROM hr.new_employees@orc1.example.com;
```

- The DISTINCT or UNIQUE keyword

  For example, the following statement creates a complex materialized view:

```
CREATE MATERIALIZED VIEW hr.employee_depts AS
  SELECT DISTINCT department_id FROM hr.employees@orc1.example.com
  ORDER BY department_id;
```

- In some cases, an aggregate function, although it is possible to have an aggregate function in the defining query and still have a simple materialized view

  For example, the following statement creates a complex materialized view:

```
CREATE MATERIALIZED VIEW hr.average_sal AS
  SELECT AVG(salary) "Average" FROM hr.employees@orc1.example.com;
```

- In some cases, joins other than those in a subquery, although it is possible to have joins in the defining query and still have a simple materialized view

  For example, the following statement creates a complex materialized view:

```
CREATE MATERIALIZED VIEW hr.emp_join_dep AS
  SELECT last_name
```

```
FROM hr.employees@orc1.example.com e, hr.departments@orc1.example.com d
WHERE e.department_id = d.department_id;
```

- In some cases, a UNION operation

  Specifically, a materialized view with a UNION operation is complex if any one of these conditions is true:

  – Any query within the UNION is complex. The previous bullet items specify when a query makes a materialized view complex.

  – The outermost SELECT list columns do not match for the queries in the UNION. In the following example, the first query only has order_total in the outermost SELECT list while the second query has customer_id in the outermost SELECT list. Therefore, the materialized view is complex.

```
CREATE MATERIALIZED VIEW oe.orders AS
  SELECT order_total
  FROM oe.orders@orc1.example.com o
  WHERE EXISTS
    (SELECT cust_first_name, cust_last_name
       FROM oe.customers@orc1.example.com c
       WHERE o.customer_id = c.customer_id
       AND c.credit_limit > 50)
UNION
  SELECT customer_id
  FROM oe.orders@orc1.example.com o
  WHERE EXISTS
    (SELECT cust_first_name, cust_last_name
       FROM oe.customers@orc1.example.com c
       WHERE o.customer_id = c.customer_id
       AND c.account_mgr_id = 30);
```

  The innermost SELECT list has no bearing on whether a materialized view is complex. In the previous example, the innermost SELECT list is cust_first_name and cust_last_name for both queries in the UNION.

- Clauses that do not comply with the requirements detailed in "Restrictions for Materialized Views with Subqueries" on page 3-19

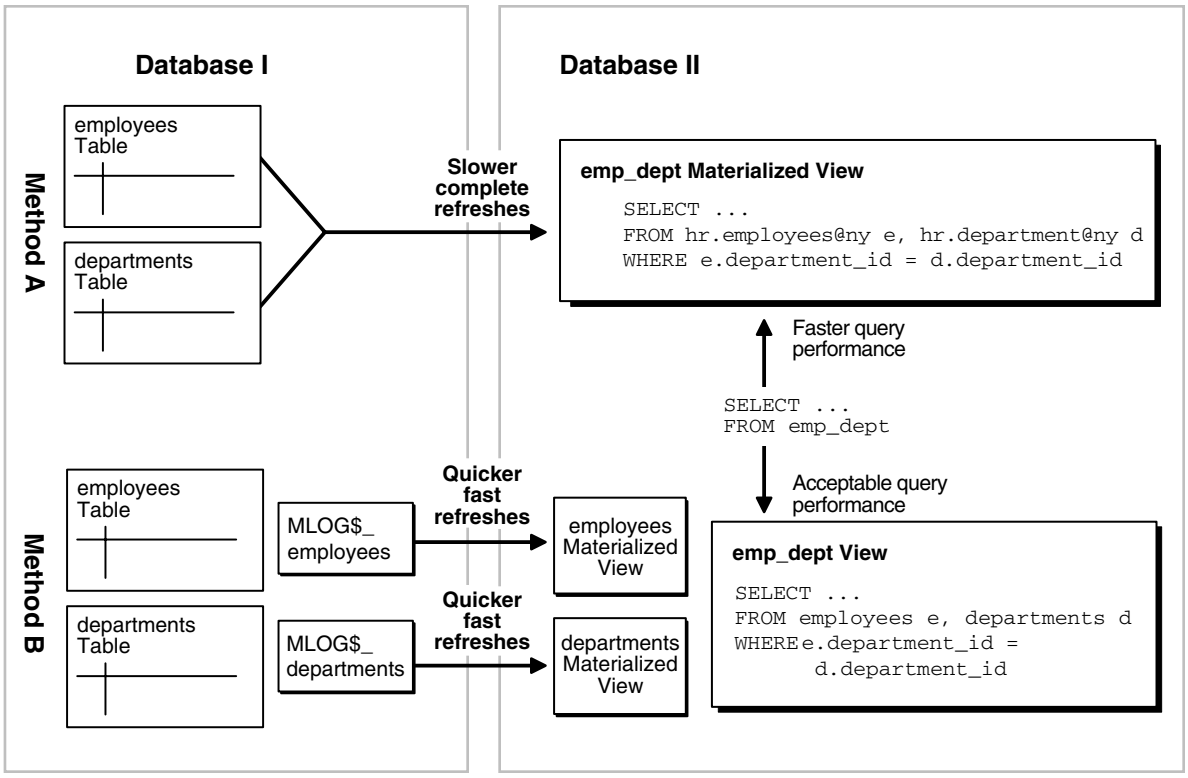
Note: If possible, you should avoid using complex materialized views because they cannot be fast refreshed, which might degrade network performance (see "Refresh Process" on page 3-45 for information).

See Also:

- *Oracle Database Data Warehousing Guide* for information about materialized views with aggregate functions and joins
- *Oracle Database SQL Language Reference* for more information about the CONNECT BY clause, set operations, the DISTINCT keyword, and aggregate functions

A Comparison of Simple and Complex Materialized Views For certain applications, you might want to consider using a complex materialized view. Figure 3–2 and the following text discuss some issues that you should consider.

Figure 3–2 Comparison of Simple and Complex Materialized Views



- **Complex Materialized View**: Method A in Figure 3–2 shows a complex materialized view. The materialized view in Database II exhibits efficient query performance because the join operation was completed during the materialized view's refresh. However, complete refreshes must be performed because the materialized view is complex, and these refreshes will probably be slower than fast refreshes.
- **Simple Materialized Views with a Joined View**: Method B in Figure 3–2 shows two simple materialized views in Database II, as well as a view that performs the join in the materialized view's database. Query performance against the view would not be as good as the query performance against the complex materialized view in Method A. However, the simple materialized views can be refreshed more efficiently using fast refresh and materialized view logs.

In summary, to decide which method to use:

- If you refresh rarely and want faster query performance, then use Method A (complex materialized view).
- If you refresh regularly and can sacrifice query performance, then use Method B (simple materialized view).

Required Privileges for Materialized View Operations

Three distinct types of users perform operations on materialized views:

- **Creator:** the user who creates the materialized view.
- **Refresher:** the user who refreshes the materialized view.
- **Owner:** the user who owns the materialized view. The materialized view resides in this user's schema.

One user can perform all of these operations on a particular materialized view. However, in some replication environments, different users perform these operations on a particular materialized view. The privileges required to perform these operations depend on whether the same user performs them or different users perform them. The following sections explain the privileges requirements in detail.

If the owner of a materialized view at the materialized view site has a private database link to the master site or master materialized view site, then the database link connects to the owner of the master at the master site or master materialized view site. Otherwise, the normal rules for connections through database links apply.

Note: The following sections do not cover the requirements necessary to create materialized views with query rewrite enabled. See the *Oracle Database SQL Language Reference* for information.

See Also: The following sections discuss database links. See the *Oracle Database Administrator's Guide* for more information about using database links.

Creator Is Owner

If the creator of a materialized view also owns the materialized view, then this user must have the following privileges to create a materialized view, granted either explicitly or through a role:

- `CREATE MATERIALIZED VIEW` or `CREATE ANY MATERIALIZED VIEW`.
- `CREATE TABLE` or `CREATE ANY TABLE`.
- `SELECT` object privilege on the master and the master's materialized view log or `SELECT ANY TABLE` system privilege. If the master site or master materialized view site is remote, then the `SELECT` object privilege must be granted to the user at the master site or master materialized view site to which the user at the materialized view site connects through a database link.

Creator Is Not Owner

If the creator of a materialized view is not the owner, certain privileges must be granted to the creator and to the owner to create a materialized view. The creator's privileges can be granted explicitly or through a role, but the owner's privileges must be granted explicitly. That is, the privileges granted to the owner cannot be granted through a role.

Table 3–1 shows the required privileges when the creator of the materialized view is not the owner.

Table 3–1 Required Privileges for Creating Materialized Views (Creator != Owner)

Creator	Owner
CREATE ANY MATERIALIZED VIEW	CREATE TABLE or CREATE ANY TABLE SELECT object privilege on the master and the master's materialized view log or SELECT ANY TABLE system privilege. If the master site or master materialized view site is remote, then the SELECT object privilege must be granted to the user at the master site or master materialized view site to which the user at the materialized view site connects through a database link. Note: These privileges for the owner must be granted to the user explicitly, not through a role.

Refresher Is Owner

If the refresher of a materialized view also owns the materialized view, this user must have SELECT object privilege on the master and the master's materialized view log or SELECT ANY TABLE system privilege. If the master site or master materialized view site is remote, then the SELECT object privilege must be granted to the user at the master site or master materialized view site to which the user at the materialized view site connects through a database link. This privilege can be granted either explicitly or through a role.

Refresher Is Not Owner

If the refresher of a materialized view is not the owner, certain privileges must be granted to the refresher and to the owner. These privileges can be granted either explicitly or through a role.

Table 3–2 shows the required privileges when the refresher of the materialized view is not the owner.

Table 3–2 Required Privileges for Refreshing Materialized Views (Refresher != Owner)

Refresher	Owner
ALTER ANY MATERIALIZED VIEW	If the master site or master materialized view site is local, then SELECT object privilege on the master and master's materialized view log or SELECT ANY TABLE system privilege. If the master site or master materialized view site is remote, then the SELECT object privilege must be granted to the user at the master site or master materialized view site to which the user at the materialized view site connects through a database link.

Data Subsetting with Materialized Views

In certain situations, you might want your materialized view to reflect a subset of the data in the master table or master materialized view. Row subsetting enables you to include only the rows that are needed from the master in the materialized views by using a WHERE clause. Column subsetting enables you to include only the columns that are needed from the master in the materialized views. You do this by specifying certain select columns in the SELECT statement during materialized view creation. If you use deployment templates to build your materialized views, then you can define column subsets on updatable materialized views.

Some reasons to use data subsetting are to:

- **Reduce Network Traffic**: In a column-subsetted materialized view, only changes that satisfy the WHERE clause of the materialized view's defining query are propagated to the materialized view site, thereby reducing the amount of data transferred and reducing network traffic.
- **Secure Sensitive Data**: Users can only view data that satisfies the defining query for the materialized view.
- **Reduce Resource Requirements**: If the materialized view is located on a laptop, then hard disks are generally significantly smaller than the hard disks on a corporate server. Subsetted materialized views might require significantly less storage space.
- **Improve Refresh Times**: Because less data is propagated to the materialized view site, the refresh process is faster, which is essential for those who need to refresh materialized views using a dial up network connection from a laptop.

For example, the following statement creates a materialized view based on the oe.orders@orc1.example.com master table and includes only the rows for the sales representative with a sales_rep_id number of 173:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST AS
 SELECT * FROM oe.orders@orc1.example.com
 WHERE sales_rep_id = 173;
```

Rows of the orders table with a sales_rep_id number other than 173 are excluded from this materialized view.

> **Note:** The following sections discuss row subsetting through the use of subqueries. For more information about column subsetting, see "Column Subsetting with Deployment Templates" on page 4-15.

Materialized Views with Subqueries

The previous example works well for individual materialized views that do not have any referential constraints to other materialized views. But, if you want to replicate data based on the information in more than one table, then maintaining and defining these materialized views can be difficult. The following sections provide examples of situations where a subquery is useful.

Many to One Subqueries Consider a scenario where you have the customers table and orders table in the oe schema, and you want to create a materialized view of the orders table based on data in both the orders table and the customers table. For example, suppose a salesperson wants to see all of the orders for the customers with a credit limit greater than $10,000. In this case, the CREATE MATERIALIZED VIEW statement that creates the orders materialized view has a subquery with a many to one relationship, because there can be many orders for each customer.

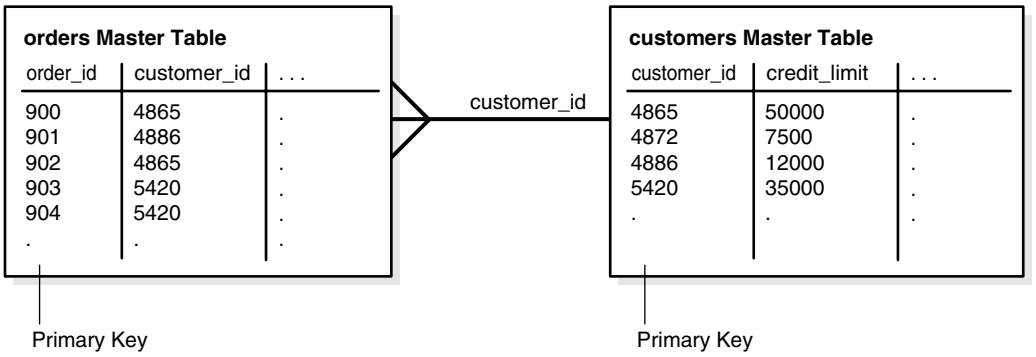
Look at the relationships in Figure 3–3, and notice that the customers and orders tables are related through the customer_id column. The following statement satisfies the original goal of the salesperson. That is, the following statement creates a materialized view that contains orders for customers whose credit limit is greater than $10,000:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST FOR UPDATE AS
  SELECT * FROM oe.orders@orc1.example.com o
  WHERE EXISTS
```

```
(SELECT * FROM oe.customers@orc1.example.com c
 WHERE o.customer_id = c.customer_id AND c.credit_limit > 10000);
```

Note: To create this `oe.orders` materialized view, `credit_limit` must be logged in the master's materialized view log. See "Logging Columns in the Materialized View Log" on page 6-13 for more information.

Figure 3–3 Row Subsetting with Many to One Subqueries



As you can see, the materialized view created by this statement is fast refreshable and updatable. If new customers are identified that have a credit limit greater than $10,000, then the new data will be propagated to the materialized view site during the subsequent refresh process. Similarly, if a customer's credit limit drops to less than $10,000, then the customer's data will be removed from the materialized view during the subsequent refresh process.

One to Many Subqueries Consider a scenario where you have the `customers` table and `orders` table in the `oe` schema, and you want to create a materialized view of the `customers` table based on data in both the `customers` table and the `orders` table. For example, suppose a salesperson wants to see all of the customers who have an order with an order total greater than $20,000, then the most efficient method is to create a materialized view with a one to many subquery in the defining query of a materialized view.

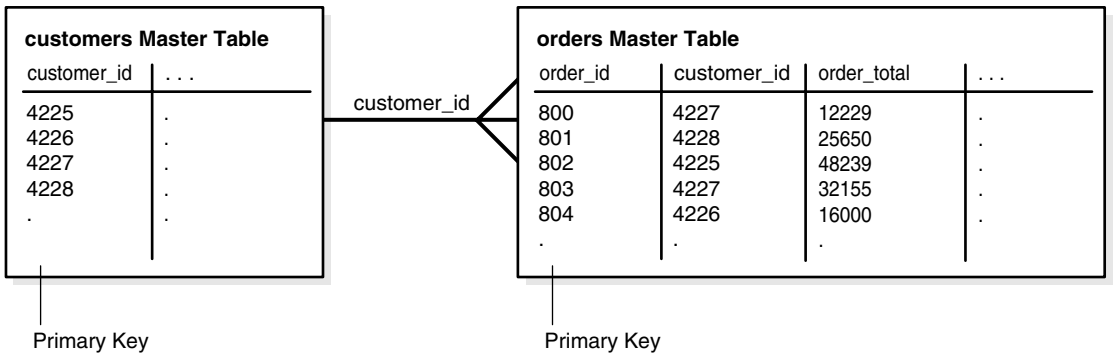
Here, the defining query in the `CREATE MATERIALIZED VIEW` statement on the `customers` table has a subquery with a one to many relationship. That is, one customer can have many orders.

Look at the relationships in Figure 3–4, and notice that the `orders` table and `customers` table are related through the `customer_id` column. The following statement satisfies the original goal of the salesperson. That is, this statement creates a materialized view that contains customers who have an order with an order total greater than $20,000:

```
CREATE MATERIALIZED VIEW oe.customers REFRESH FAST FOR UPDATE AS
  SELECT * FROM oe.customers@orc1.example.com c
  WHERE EXISTS
    (SELECT * FROM oe.orders@orc1.example.com o
     WHERE c.customer_id = o.customer_id AND o.order_total > 20000);
```

Note: To create this `oe.customers` materialized view, `customer_id` and `order_total` must be logged in the materialized view log for the `orders` table. See "Logging Columns in the Materialized View Log" on page 6-13 for more information.

Figure 3–4 Row Subsetting with One to Many Subqueries



The materialized view created by this statement is fast refreshable and updatable. If new customers are identified that have an order total greater than $20,000, then the new data will be propagated to the materialized view site during the subsequent refresh process. Similarly, if a customer cancels an order with an order total greater than $20,000 and has no other order totals greater than $20,000, then the customer's data will be removed from the materialized view during the subsequent refresh process.

Many to Many Subqueries Consider a scenario where you have the `order_items` table and `inventories` table in the `oe` schema, and you want to create a materialized view of the `inventories` table based on data in both the `inventories` table and the `order_items` table. For example, suppose a salesperson wants to see all of the inventories with a quantity on hand greater than 0 (zero) for each product whose `product_id` is in the `order_items` table. In other words, the salesperson wants to see the inventories that are greater than zero for all of the products that customers have ordered. Here, an inventory is a certain quantity of a product at a particular warehouse. So, a certain product can be in many order items and in many inventories.

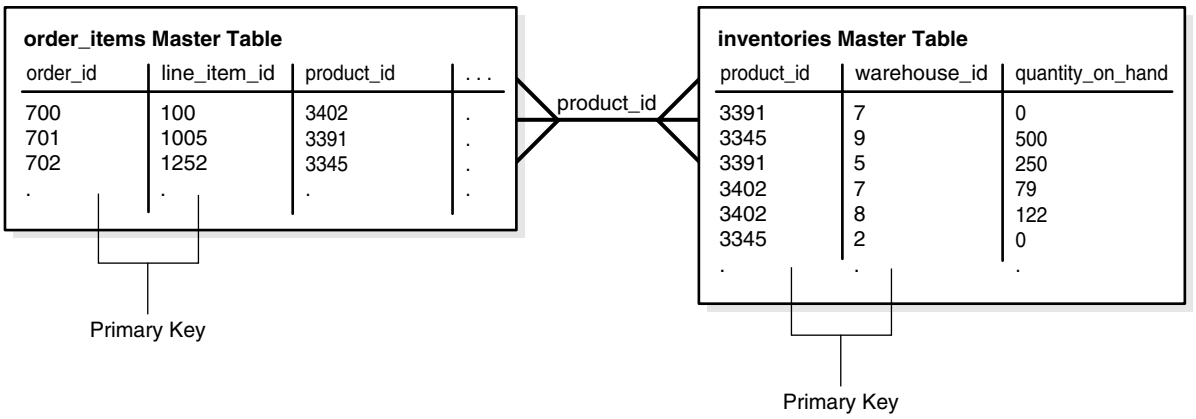
To accomplish the salesperson's goal, you can create a materialized view with a subquery on the many to many relationship between the `order_items` table and the `inventories` table.

When you create the `inventories` materialized view, you want to retrieve the inventories with the quantity on hand greater than zero for the products that appear in the `order_items` table. Look at the relationships in Figure 3–5, and note that the `inventories` table and `order_items` table are related through the `product_id` column. The following statement creates the materialized view:

```
CREATE MATERIALIZED VIEW oe.inventories REFRESH FAST FOR UPDATE AS
  SELECT * FROM oe.inventories@orc1.example.com i
  WHERE i.quantity_on_hand > 0 AND EXISTS
    (SELECT * FROM oe.order_items@orc1.example.com o
     WHERE i.product_id = o.product_id);
```

Note: To create this `oe.inventories` materialized view, the `product_id` column in the `order_items` table must be logged in the master's materialized view log. See "Logging Columns in the Materialized View Log" on page 6-13 for more information.

Figure 3–5 Row Subsetting with Many to Many Subqueries



The materialized view created by this statement is fast refreshable and updatable. If new inventories that are greater than zero are identified for products in the `order_items` table, then the new data will be propagated to the materialized view site during the subsequent refresh process. Similarly, if a customer cancels an order for a product and there are no other orders for the product in the `order_items` table, then the inventories for the product will be removed from the materialized view during the subsequent refresh process.

Materialized Views with Subqueries and Unions In situations where you want a single materialized view to contain data that matches the complete results of two or more different queries, you can use the `UNION` operator. When you use the `UNION` operator to create a materialized view, you have two `SELECT` statements around each `UNION` operator, one is above it and one is below it. The resulting materialized view contains rows selected by either query.

You can use the `UNION` operator as a way to create fast refreshable materialized views that satisfy "or" conditions without using the `OR` expression in the `WHERE` clause of a subquery. Under some conditions, using an `OR` expression in the `WHERE` clause of a subquery causes the resulting materialized view to be complex, and therefore not fast refreshable.

> **See Also:** "Restrictions for Materialized Views with Subqueries" on page 3-19 for more information about the `OR` expressions in subqueries

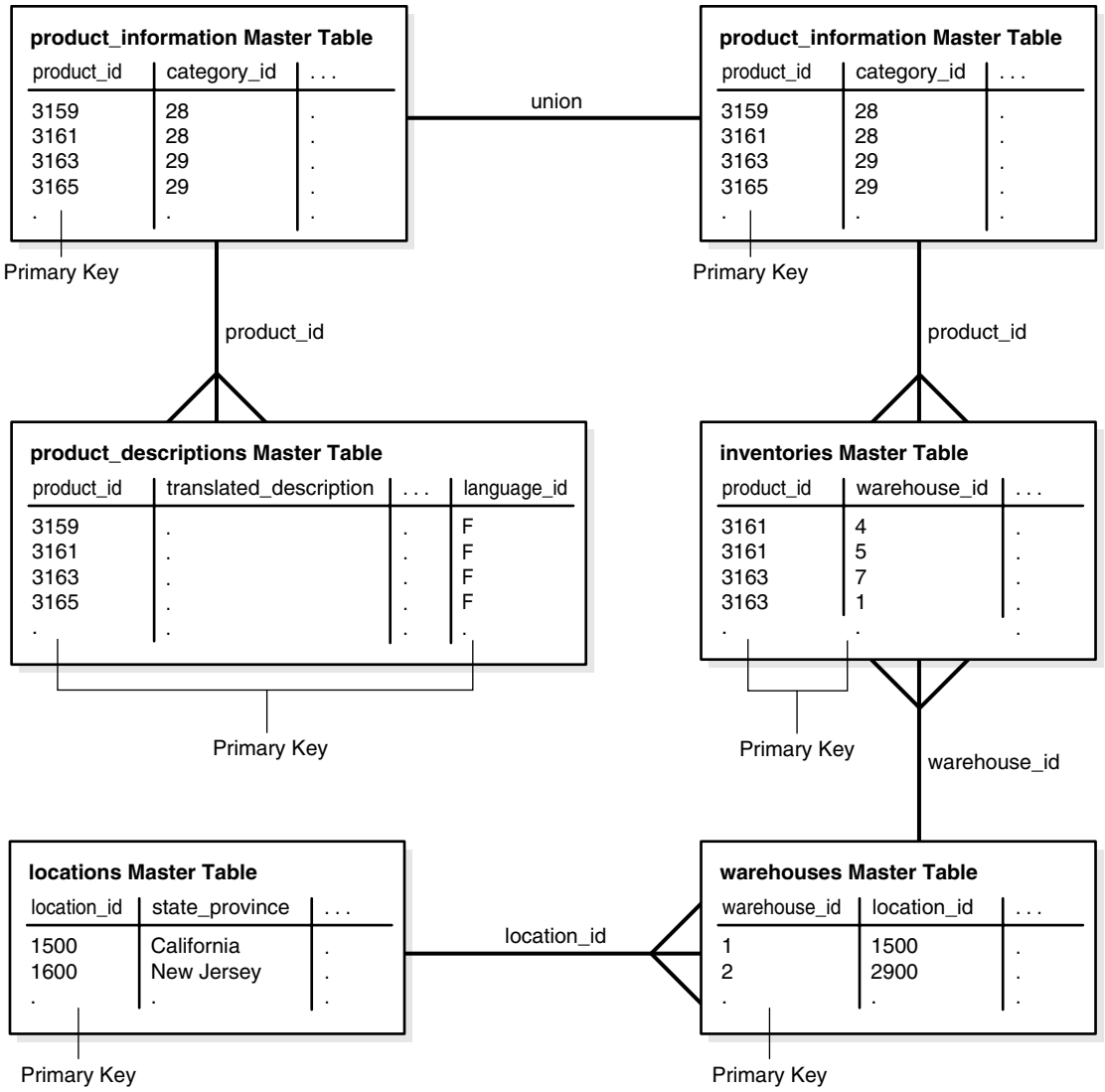
For example, suppose a salesperson wants the product information for the products in a particular `category_id` that are *either* in a warehouse in California *or* contain the word "Rouge" in their translated product descriptions (for the French translation). The following statement uses the `UNION` operator and subqueries to capture this data in a materialized view for products in `category_id` 29:

```
CREATE MATERIALIZED VIEW oe.product_information REFRESH FAST FOR UPDATE AS
 SELECT * FROM oe.product_information@orc1.example.com pi
 WHERE pi.category_id = 29 AND EXISTS
  (SELECT * FROM oe.product_descriptions@orc1.example.com pd
  WHERE pi.product_id = pd.product_id AND
        pd.translated_description LIKE '%Rouge%')
UNION
 SELECT * FROM oe.product_information@orc1.example.com pi
 WHERE pi.category_id = 29 AND EXISTS
  (SELECT * FROM oe.inventories@orc1.example.com i
  WHERE pi.product_id = i.product_id AND EXISTS
    (SELECT * FROM oe.warehouses@orc1.example.com w
    WHERE i.warehouse_id = w.warehouse_id AND EXISTS
      (SELECT * FROM hr.locations@orc1.example.com l
       WHERE w.location_id = l.location_id
       AND l.state_province = 'California')));
```

Note: To create the `oe.product_information` materialized view, `translated_description` in the `oe.product_descriptions` table, the `state_province` in the `hr.locations` table, and the `location_id` column in the `oe.warehouses` table must be logged in each master's materialized view log. See "Logging Columns in the Materialized View Log" on page 6-13 for more information.

Figure 3–6 shows the relationships of the master tables involved in this statement.

Figure 3–6 Row Subsetting with Subqueries and Unions



In addition to the `UNION` operation, this statement contains the following subqueries:

- A subquery referencing the `product_information` table and the `product_descriptions` table. This subquery is one to many because one product can have multiple product descriptions (for different languages).
- A subquery referencing the `product_information` table and the `inventories` table. This subquery is one to many because a product can be in many inventories.
- A subquery referencing the `inventories` table and the `warehouses` table. This subquery is many to one because many inventories can be stored in one warehouse.
- A subquery referencing the `warehouses` table and the `locations` table. This subquery is many to one because many warehouses can be in one location.

The materialized view created by this statement is fast refreshable and updatable. If a new product is added that is stored in a warehouse in California or that has the string "Rouge" in the translated product description, then the new data will be propagated to the `product_information` materialized view during the subsequent refresh process.

Restrictions for Materialized Views with Subqueries

The defining query of a materialized view with a subquery is subject to several restrictions to preserve the materialized view's fast refresh capability.

The following are restrictions for fast refresh materialized views with subqueries:

- Materialized views must be primary key materialized views.
- The master's materialized view log must include certain columns referenced in the subquery. For information about which columns must be included, see "Logging Columns in the Materialized View Log" on page 6-13.
- If the subquery is many to many or one to many, join columns that are not part of a primary key must be included in the materialized view log of the master. This restriction does not apply to many to one subqueries.
- The subquery must be a positive subquery. For example, you can use the `EXISTS` condition, but not the `NOT EXISTS` condition.
- The subquery must use `EXISTS` to connect each nested level (`IN` is not allowed).
- Each table can be in only one `EXISTS` expression.
- The join expression must use exact match or equality comparisons (that is, equi-joins).
- Each table can be joined only once within the subquery.
- A primary key must exist for each table at each nested level.
- Each nested level can only reference the table in the level above it.
- Subqueries can include `AND` conditions, but each `OR` condition can only reference columns contained within one row. Multiple `OR` conditions within a subquery can be connected with an `AND` condition.
- All tables referenced in a subquery must reside in the same master site or master materialized view site.

Note: If the `CREATE MATERIALIZED VIEW` statement includes an `ON PREBUILT TABLE` clause and a subquery, then the subquery is treated as many to many. Therefore, in this case, the join columns must be recorded in the materialized view log. See the *Oracle Database SQL Language Reference* for more information about the `ON PREBUILT TABLE` clause in the `CREATE MATERIALIZED VIEW` statement.

See Also: "Primary Key Materialized Views" on page 3-6 for more information about primary key materialized views

Restrictions for Materialized Views with Unions Containing Subqueries

The following are restrictions for fast refresh materialized views with unions containing subqueries:

- All of the restrictions described in the previous section, "Restrictions for Materialized Views with Subqueries" on page 3-19, apply to the subqueries in each union block.
- All join columns must be included in the materialized view log of the master, even if the subquery is many to one.

- All of the restrictions described in the previous section, "Complex Materialized Views" on page 3-8, for clauses with UNIONS.

Examples of Materialized Views with Unions Containing Subqueries The following statement creates the oe.orders materialized view. This materialized view is fast refreshable because the subquery in each union block satisfies the restrictions for subqueries described in "Restrictions for Materialized Views with Subqueries" on page 3-19.

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST AS
  SELECT * FROM oe.orders@orc1.example.com o
  WHERE EXISTS
    (SELECT * FROM oe.customers@orc1.example.com c
     WHERE o.customer_id = c.customer_id
     AND c.credit_limit > 50)
UNION
  SELECT *
  FROM oe.orders@orc1.example.com o
  WHERE EXISTS
    (SELECT * FROM oe.customers@orc1.example.com c
     WHERE o.customer_id = c.customer_id
     AND c.account_mgr_id = 30);
```

Notice that one of the restrictions for subqueries states that each table can be in only one EXISTS expression. Here, the customers table appears in two EXISTS expressions, but the EXISTS expressions are in separate UNION blocks. Because the restrictions described in "Restrictions for Materialized Views with Subqueries" on page 3-19 only apply to each UNION block, not to the entire CREATE MATERIALIZED VIEW statement, the materialized view is fast refreshable.

In contrast, the materialized view created with the following statement cannot be fast refreshed because the orders table is referenced in two different EXISTS expressions within the same UNION block:

```
CREATE MATERIALIZED VIEW oe.orders AS
  SELECT * FROM oe.orders@orc1.example.com o
  WHERE EXISTS
    (SELECT * FROM oe.customers@orc1.example.com c
     WHERE o.customer_id = c.customer_id  -- first reference to orders table
     AND c.credit_limit > 50
     AND EXISTS
        (SELECT * FROM oe.orders@orc1.example.com o
         WHERE order_total > 5000
         AND o.customer_id = c.customer_id)) -- second reference to orders table
UNION
  SELECT *
  FROM oe.orders@orc1.example.com o
  WHERE EXISTS
    (SELECT * FROM oe.customers@orc1.example.com c
     WHERE o.customer_id = c.customer_id
     AND c.account_mgr_id = 30);
```

Determining the Fast Refresh Capabilities of a Materialized View

To determine whether a materialized view's subquery satisfies the restrictions detailed in the previous section, create the materialized view with fast refresh. Oracle returns errors if the materialized view violates any restrictions for subquery materialized views. If you specify force refresh, then you might not receive any errors because, when a force refresh is requested, Oracle automatically performs a complete refresh if it cannot perform a fast refresh.

You can also use the EXPLAIN_MVIEW procedure in the DBMS_MVIEW package to determine the following information about an existing materialized view or a proposed materialized view that does not yet exist:

- The capabilities of a materialized view
- Whether each capability is possible
- If a capability is not possible, why it is not possible

This information can be stored in a varray or in the MV_CAPABILITIES_TABLE. If you want to store the information in the table, then, before you run the EXPLAIN_MVIEW procedure, you must build this table by running the utlxmv.sql script in the *Oracle_home*/rdbms/admin directory.

For example, to determine the capabilities of the oe.orders materialized view, enter:

```
EXECUTE DBMS_MVIEW.EXPLAIN_MVIEW ('oe.orders');
```

Or, if the materialized view does not yet exist, then you can supply the query that you want to use to create it:

```
BEGIN
  DBMS_MVIEW.EXPLAIN_MVIEW ('SELECT * FROM oe.orders@orc1.example.com o
    WHERE EXISTS (SELECT * FROM oe.customers@orc1.example.com c
    WHERE o.customer_id = c.customer_id AND c.credit_limit > 500)');
END;
/
```

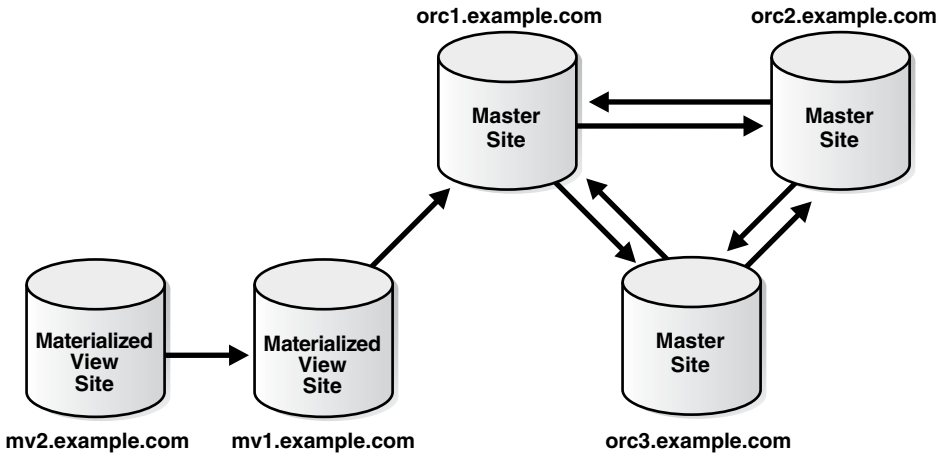
Query the MV_CAPABILITIES_TABLE to see the results.

> **See Also:** *Oracle Database Data Warehousing Guide* for more information about the EXPLAIN_MVIEW procedure

Multitier Materialized Views

The ability to create materialized views that are based on other materialized views enables you to create **multitier materialized views**. Materialized views that are based on other materialized views can be read-only or updatable. The arrows in Figure 3–7 represent database links.

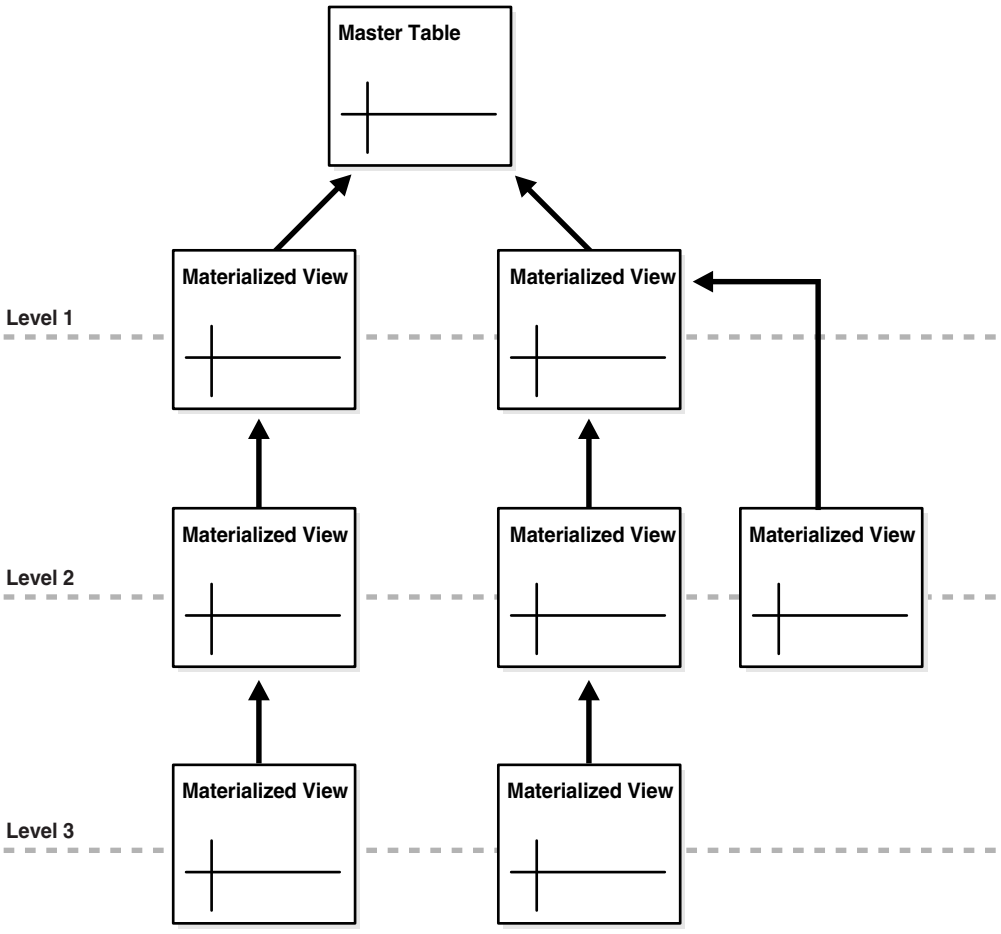
Figure 3–7 Multitier Materialized Views



When you are using multitier materialized views, the materialized view based on a master table is called a level 1 materialized view. Then, a materialized view based on

the level 1 materialized view is called a level 2 materialized view. Next is level 3 and so on. Figure 3–8 shows these levels.

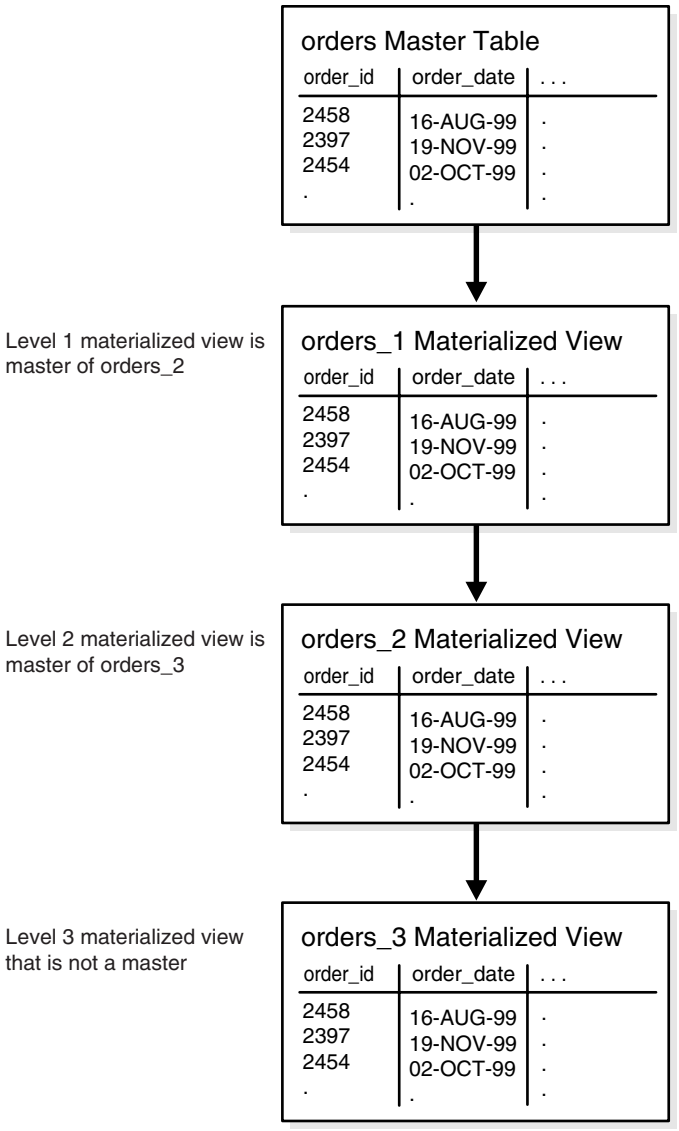
Figure 3–8 Levels of Materialized Views



A materialized view that is acting as the master for another materialized view is called a **master materialized view**. A materialized view at any level can be a master materialized view, and, as you can see in Figure 3–8, a master materialized view can have more than one materialized view based on it. In Figure 3–8, two level 2 materialized views are based on one level 1 materialized view.

Figure 3–9 illustrates an example that shows a master materialized view at level 1 (`orders_1`) and level 2 (`orders_2`).

Figure 3–9 Master Materialized Views



The master for the level 1 materialized view `orders_1` is the master table `orders` at the master site, but, starting with level 2, each materialized view has a master materialized view at the level above it. For example, the master for the level 2 materialized view `orders_2` is the level 1 materialized view `orders_1`.

A master materialized view functions the same way a master table does at a master site. That is, changes pushed from a level 2 materialized view to a level 1 materialized view are handled in exactly the same way that changes pushed from a level 1 materialized view to a master table are handled.

A receiver must be registered at a master materialized view site. The receiver is responsible for receiving and applying the deferred transactions from the propagator at multitier materialized view sites that are based on the master materialized view.

See Also: "Receiver" on page 2-14

Multitier materialized views offer greater flexibility in the design of a replication environment. Some materialized view sites might not need to replicate all of the data in master tables, and, in fact, these sites might not have the storage capacity for all of the data. In addition, replicating less data means that there is less activity on the network.

Multitier materialized views are ideal for organizations that are structured on three or more levels or constrained by limited network resources. For example, consider a company with international, national, and local offices. This company has many computers at both the national and local level that replicate data. Here, the replication environment can be configured with the master site at the international headquarters and with materialized views at the national level. These materialized views at the national level only replicate the subset of data from the master tables that apply to their respective countries. Now, using multitier materialized views, another level of materialized views at the local level can be based on the materialized views at the national level. The materialized views at the local level contain the subset of data from the level 1 materialized views that apply to their local customers.

Scenario for Using Multitier Materialized Views

Consider a multinational company that maintains all employee information at headquarters, which is in the in the United States. The company uses the tables in the `hr` schema to maintain the employee information. This company has one main office in 14 countries and many regional offices for cities in these countries.

For example, the company has one main office for all of the United Kingdom, but it also has an office in the city of London. The United Kingdom office maintains employee information for all of the employees in the United Kingdom, while the London office only maintains employee information for the employees at the London office. In this scenario, the `hr.employees` master table is at headquarters in the United States and each regional office has a an `hr.employees` materialized view that only contains the necessary employee information.

The following statement creates the `hr.employees` materialized view for the United Kingdom office. The statement queries the master table in the database at headquarters, which is `orc1.example.com`. Notice that the statement uses subqueries so that the materialized view only contains employees whose `country_id` is `UK`.

```
CREATE MATERIALIZED VIEW hr.employees REFRESH FAST FOR UPDATE AS
  SELECT * FROM hr.employees@orc1.example.com e
    WHERE EXISTS
      (SELECT * FROM hr.departments@orc1.example.com d
       WHERE e.department_id = d.department_id
       AND EXISTS
         (SELECT * FROM hr.locations@orc1.example.com l
          WHERE l.country_id = 'UK'
          AND d.location_id = l.location_id));
```

Note: To create this `hr.employees` materialized view, the following columns must be logged:

- The `department_id` column must be logged in the materialized view log for the `hr.employees` master table at `orc1.example.com`.
- The `country_id` must be logged in the materialized view log for the `hr.locations` master table at `orc1.example.com`.

See "Logging Columns in the Materialized View Log" on page 6-13 for more information.

The following statement creates the `hr.employees` materialized view for the London office based on the level 1 materialized view at the United Kingdom office. The statement queries the materialized view in the database at the United Kingdom office, which is `reg_uk.example.com`. Notice that the statement uses subqueries so that the materialized view only contains employees whose `city` is `London`.

```
CREATE MATERIALIZED VIEW hr.employees REFRESH FAST FOR UPDATE AS
  SELECT * FROM hr.employees@reg_uk.example.com e
    WHERE EXISTS
      (SELECT * FROM hr.departments@reg_uk.example.com d
       WHERE e.department_id = d.department_id
       AND EXISTS
         (SELECT * FROM hr.locations@reg_uk.example.com l
          WHERE l.city = 'London'
          AND d.location_id = l.location_id));
```

Note: To create this `hr.employees` materialized view, the following columns must be logged:

- The `department_id` column must be logged in the materialized view log for the `hr.employees` master materialized view at `reg_uk.example.com`.
- The `country_id` must be logged in the materialized view log for the `hr.locations` master materialized view at `reg_uk.example.com`.

See "Logging Columns in the Materialized View Log" on page 6-13 for more information.

Restrictions for Using Multitier Materialized Views

Both master materialized views and materialized views based on materialized views must be primary key materialized views.

Additional Restrictions for Master Materialized Views The following types of materialized views cannot be masters for updatable materialized views:

- `ROWID` materialized views
- Complex materialized views
- Read-only materialized views

However, these types of materialized views can be masters for read-only materialized views.

Additional Restrictions for Updatable Materialized Views Based on Materialized Views

Updatable materialized views based on materialized views must:

- Belong to a materialized view group that has the same name as the materialized view group at its master materialized view site.
- Reside in a different database than the materialized view group at its master materialized view site.
- Be based on another updatable materialized view or other updatable materialized views, not on a read-only materialized view.
- Be based on a materialized view in a materialized view group that is owned by `PUBLIC` at the master materialized view site.

How Materialized Views Work with Object Types and Collections

Oracle **object types** are user-defined data types that make it possible to model complex real-world entities such as customers and orders as single entities, called **objects**, in the database. You create object types using the `CREATE TYPE ... AS OBJECT` statement. You can replicate object types and objects between master sites and materialized view sites in a replication environment.

An Oracle object that occupies a single column in a table is called a **column object**. Typically, tables that contain column objects also contain other columns, which can be built-in data types, such as `VARCHAR2` and `NUMBER`. An **object table** is a special kind of table in which each row represents an object. Each row in an object table is a **row object**.

You can also replicate **collections**. Collections are user-defined data types that are based on `VARRAY` and nested table data types. You create varrays with the `CREATE TYPE ... AS VARRAY` statement, and you create nested tables with the `CREATE TYPE ... AS TABLE` statement.

Note:

- You cannot create refresh-on-commit materialized views based on a master with user-defined types or Oracle-supplied types. Refresh-on-commit materialized views are those created using the `ON COMMIT REFRESH` clause in the `CREATE MATERIALIZED VIEW` statement.
- Advanced Replication does not support type inheritance, and Advanced Replication does not support types created with the `NOT FINAL` clause.

See Also:

- *Oracle Database Object-Relational Developer's Guide* for detailed information about user-defined types, Oracle objects, and collections. This section assumes a basic understanding of the information in that book.
- *Oracle Database SQL Language Reference* for more information about user-defined types and Oracle-supplied types

Type Agreement at Replication Sites

User-defined types include all types created using the CREATE TYPE statement, including object, nested table, VARRAY, and indextype. To replicate schema objects based on user-defined types, the user-defined types themselves must exist, and must be exactly the same, at all replication sites. In addition, Oracle recommends that you add a user-defined type to the replication group in which it is used, but doing so is not required.

When replicating user-defined types and the schema objects on which they are based, the following conditions apply:

- The user-defined types replicated at the master site and materialized view site must be created at the materialized view site before you create any materialized views that depend on these types.
- All of the masters on which a materialized view is based must be at the same master site to create a materialized view with user-defined types.
- A user-defined type must be exactly the same at all replication sites:
  - All replication sites must have the same object identifier (OID), schema owner, and type name for each replicated user-defined type.
  - If the user-defined type is an object type, then all replication sites must agree on the order and data type of the attributes in the object type. You establish the order and data types of the attributes when you create the object type. For example, consider the following object type:

```
CREATE TYPE cust_address_typ AS OBJECT
     (street_address     VARCHAR2(40),
      postal_code        VARCHAR2(10),
      city               VARCHAR2(30),
      state_province     VARCHAR2(10),
      country_id         CHAR(2));
/
```

    At all replication sites, street_address must be the first attribute for this type and must be VARCHAR2(40), postal_code must be the second attribute and must be VARCHAR2(10), city must be the third attribute and must be VARCHAR2(30), and so on.

  - All replication sites must agree on the hashcode of the user-defined type. Oracle examines a user-defined type and assigns the hashcode. This examination includes the type attributes, order of attributes, and type name. When all of these items are the same for two or more types, the types have the same hashcode. You can view the hashcode for a type by querying the DBA_TYPE_VERSIONS data dictionary view.

To ensure that a user-defined type is exactly the same at all replication sites, you must create the user-defined type at the materialized view site in one of the following ways:

- Use the Replication Management API
- Use a CREATE TYPE Statement

Use the Replication Management API Oracle recommends that you use the replication management API to create, modify, or drop any replicated object at a materialized view site, including user-defined types. If you do not use the replication management API for these actions, then replication errors might result.

Specifically, to create a user-defined type that is exactly the same at the master site and the materialized view site, use the CREATE_MVIEW_REPOBJECT procedure in the

DBMS_REPCAT package. This procedure creates the type and adds it to a materialized view group. To drop a user-defined type from the materialized view site, use the DROP_MVIEW_REPOBJECT procedure in the DBMS_REPCAT package.

> **See Also:** *Oracle Database Advanced Replication Management API Reference*

Use a CREATE TYPE Statement You can use a CREATE TYPE statement at the materialized view site to create the type. It might be necessary to do this if you want to create a read-only materialized view that uses the type, and you do not want to add the read-only materialized view to a materialized view group.

If you choose this option, then you must ensure the following:

- The type is in the same schema at both the materialized view site and the master site.
- The type has exactly the same attributes in exactly the same order at both the materialized view site and the master site.
- The type has exactly the same data type for each attribute at both the materialized view site and the master site.
- The type has the same object identifier at both the materialized view site and the master site.

You can find the object identifier for a type by querying the DBA_TYPES data dictionary view. For example, to find the object identifier (OID) for the cust_address_typ, enter the following query:

```
SELECT TYPE_OID FROM DBA_TYPES WHERE TYPE_NAME = 'CUST_ADDRESS_TYP';

TYPE_OID
--------------------------------
6F9BC33653681B7CE03400400B40A607
```

Now that you know the OID for the type at the master site, complete the following steps to create the type at the materialized view site:

1. Log in to the materialized view site as the user who owns the type at the master site. If this user does not exist at the materialized view site, then create the user.
2. Issue the CREATE TYPE statement and specify the OID:

```
CREATE TYPE oe.cust_address_typ OID '6F9BC33653681B7CE03400400B40A607'
     AS OBJECT (
     street_address     VARCHAR2(40),
     postal_code        VARCHAR2(10),
     city               VARCHAR2(30),
     state_province     VARCHAR2(10),
     country_id         CHAR(2));
/
```

The type is now ready for use at the materialized view site.

Column Subsetting of Masters with Column Objects

A read-only materialized view can replicate specific attributes of a column object without replicating other attributes. For example, using the cust_address_typ user-defined data type described in the previous section, suppose a customers_sub master table is created at master site orc1.example.com:

```
CREATE TABLE oe.customers_sub (
      customer_id        NUMBER(6)  PRIMARY KEY,
      cust_first_name    VARCHAR2(20),
      cust_last_name     VARCHAR2(20),
      cust_address       oe.cust_address_typ);
```

You can create the following read-only materialized view at a remote materialized view site:

```
CREATE MATERIALIZED VIEW oe.customers_mv1 AS
   SELECT customer_id, cust_last_name, c.cust_address.postal_code
   FROM oe.customers_sub@orc1.example.com c;
```

Notice that the `postal_code` attribute is specified in the `cust_address` column object.

An updatable materialized view must replicate the entire column object. It cannot replicate some attributes of a column object but not others. The following statement is valid because it specifies the entire `cust_address` column object:

```
CREATE MATERIALIZED VIEW oe.customers_mv1 FOR UPDATE AS
   SELECT customer_id, cust_last_name, cust_address
   FROM oe.customers_sub@orc1.example.com;
```

> **See Also:** "Column Subsetting with Deployment Templates" on page 4-15 for more information about column subsetting with deployment templates. Column subsetting is supported only through the use of deployment templates.

Materialized Views Based on Object Tables

If a materialized view is based on an object table and is created using the `OF` *type* clause, then the materialized view is called an **object materialized view**. An object materialized view is structured in the same way as an object table. That is, an object materialized view is composed of row objects. If a materialized view that is based on an object table is created without using the `OF` *type* clause, then the materialized view is read-only and is not an object materialized view. That is, such a materialized view has regular rows, not row objects.

To create a materialized view based on an object table, the types on which the materialized view depends must exist at the materialized view site, and each type must have the same object identifier as it does at the master site.

Creation of Object Materialized Views Using the OF *type* Clause After the required types are created at the materialized view site, you can create an object materialized view by specifying the `OF` *type* clause.

For example, suppose the following SQL statements create the `oe.categories_tab` object table at the `orc1.example.com` master site:

```
CREATE TYPE oe.category_typ AS OBJECT
   (category_name           VARCHAR2(50),
    category_description    VARCHAR2(1000),
    category_id             NUMBER(2));
/

CREATE TABLE oe.categories_tab OF oe.category_typ
    (category_id    PRIMARY KEY);
```

If you want to create materialized views that can be fast refreshed based on the oe.categories_tab master table, then create a materialized view log for this table:

```
CREATE MATERIALIZED VIEW LOG ON oe.categories_tab WITH OBJECT ID;
```

The WITH OBJECT ID clause is required when you create a materialized view log on an object table.

After you create the oe.category_typ type at the materialized view site with the same object identifier as the same type at the master site, you can create an object materialized view based on the oe.categories_tab object table using the OF *type* clause, as in the following SQL statement:

```
CREATE MATERIALIZED VIEW oe.categories_objmv OF oe.category_typ
   REFRESH FAST FOR UPDATE
   AS SELECT * FROM oe.categories_tab@orc1.example.com;
```

Here, *type* is oe.category_typ.

> **Note:** The types must be exactly the same at the materialized view site and master site. See "Type Agreement at Replication Sites" on page 3-27 for more information.

Materialized Views Based on Object Tables Created Without Using the OF *type* Clause If you create a materialized view based on an object table without using the OF *type* clause, then the materialized view is read-only, and it loses the object properties of the object table on which it is based. That is, the resulting read-only materialized view contains one or more of the columns of the master, but each row functions as a row in a relational table. The rows are not row objects.

For example, you can create a materialized view base on the categories_tab master by using the following SQL statement:

```
CREATE MATERIALIZED VIEW oe.categories_relmv
   AS SELECT * FROM oe.categories_tab@orc1.example.com;
```

In this case, the categories_relmv materialized view must be read-only, and the rows in this materialized view function in the same way as rows in a relational table.

OID Preservation in Object Materialized Views An object materialized view inherits the object identifier (OID) specifications of its master. If the master has a primary key-based OID, then the OIDs of row objects in the materialized view are primary key-based. If the master has a system generated OID, then the OIDs of row objects in the materialized view are system generated. Also, the OID of each row in the object materialized view matches the OID of the same row in the master, and the OIDs are preserved during refresh of the materialized view. Consequently, REFs to the rows in the object table remain valid at the materialized view site.

Materialized Views with Collection Columns

Collection columns are columns based on varray and nested table data types. Oracle supports the creation of materialized views with collection columns.

If the collection column is a nested table, then you can optionally specify the *nested_table_storage_clause* during materialized view creation. The *nested_table_storage_clause* lets you specify the name of the storage table for the nested table in the materialized view. For example, suppose you create the master table people_reltab at the master site orc1.example.com that contains the nested table phones_ntab:

```
CREATE TYPE oe.phone_typ AS OBJECT (
   location    VARCHAR2(15),
   num         VARCHAR2(14));
/

CREATE TYPE oe.phone_ntabtyp AS TABLE OF oe.phone_typ;
/

CREATE TABLE oe.people_reltab (
   id              NUMBER(4) CONSTRAINT pk_people_reltab PRIMARY KEY,
   first_name      VARCHAR2(20),
   last_name       VARCHAR2(20),
   phones_ntab     oe.phone_ntabtyp)
   NESTED TABLE phones_ntab STORE AS phone_store_ntab
   ((PRIMARY KEY (NESTED_TABLE_ID, location)));
```

Notice the `PRIMARY KEY` specification in the last line of the preceding SQL statement. You must specify a primary key for the storage table if you plan to create materialized views based on its parent table. In this case, the storage table is `phone_store_ntab` and the parent table is `people_reltab`.

If you want to create materialized views that can be fast refreshed, then create a materialized view log on both the parent table and the storage table, specifying the nested table column as a filter column for the parent table's materialized view log:

```
CREATE MATERIALIZED VIEW LOG ON oe.people_reltab;

ALTER MATERIALIZED VIEW LOG ON oe.people_reltab ADD(phones_ntab);

CREATE MATERIALIZED VIEW LOG ON oe.phone_store_ntab WITH PRIMARY KEY;
```

At the materialized view site, create the required types, ensuring that the object identifier for each type is the same as the object identifier at the master site. Then, you can create a materialized view based on `people_reltab` and specify its storage table using the following statement:

```
CREATE MATERIALIZED VIEW oe.people_reltab_mv
   NESTED TABLE phones_ntab STORE AS phone_store_ntab_mv
   REFRESH FAST AS SELECT * FROM oe.people_reltab@orc1.example.com;
```

In this case, the *nested_table_storage_clause* is the line that begins with "`NESTED TABLE`" in the previous example, and it specifies that the storage table's name is `phone_store_ntab_mv`. The *nested_table_storage_clause* is optional. If you do not specify this clause, Oracle automatically names the storage table. To view the name of a storage table, query the `DBA_NESTED_TABLES` data dictionary table.

The storage table:

- Is a separate, secondary materialized view
- Is refreshed automatically when you refresh the materialized view containing the nested table
- Is dropped automatically when you drop the materialized view containing the nested table
- Inherits the primary key constraint of the master's storage table

Because the storage table inherits the primary key constraint of the master's storage table, do not specify `PRIMARY KEY` in the `STORE AS` clause.

The following actions are not allowed directly on the storage table of a nested table in a materialized view:

- Refreshing the storage table
- Adding the storage table to a replication group
- Altering the storage table
- Dropping the storage table
- Generating replication support on the storage table

These actions can occur indirectly when they are performed on the materialized view that contains the nested table. In addition, you cannot replicate a subset of the columns in a storage table.

> **See Also:** *Oracle Database SQL Language Reference* for more information about the *nested_table_storage_clause*, which is fully documented in the `CREATE TABLE` statement

Restrictions for Materialized Views with Collection Columns The following restrictions apply to materialized views with collection columns:

- Row subsetting of collection columns is not allowed. However, you can use row subsetting on the parent table of a nested table and doing so can result in a subset of the nested tables in the materialized view.
- Column subsetting of collection columns is not allowed.
- A nested table's storage table must have a primary key.
- For the parent table of a nested table to be fast refreshed, both the parent table and the nested table's storage table must have a materialized view log.

Materialized Views with REF Columns

You can create materialized views with `REF` columns. A `REF` is an Oracle built-in data type that is a logical "pointer" to a row object in an object table. A scoped `REF` is a `REF` that can contain references only to a specified object table, while an unscoped `REF` can contain references to any object table in the database that is based on the corresponding object type. A scoped `REF` requires less storage space and provides more efficient access than an unscoped `REF`.

As described in the following section, you can rescope a `REF` column to a local materialized view or table at the materialized view site during creation of the materialized view. If you do not rescope the `REF` column, then they continue to point to the remote master. Unscoped `REF` columns always continue to point to the master. When a `REF` column at a materialized view site points to a remote master, the `REF`s are considered dangling. In SQL, dereferencing a dangling `REF` returns a `NULL`. Also, PL/SQL only supports dereferencing `REF`s by using the `UTL_OBJECT` package and raises an exception for dangling `REF`s.

Scoped REF Columns If you are creating a materialized view based on a master that has a scoped `REF` column, then you can rescope the `REF` to a different object table or object materialized view at the materialized view site. Typically, you would rescope the `REF` column to the local object materialized view instead of the original remote object table. To rescope a materialized view, you can either use the `SCOPE FOR` clause in the `CREATE MATERIALIZED VIEW` statement, or you can use the `ALTER MATERIALIZED VIEW` statement after creating the materialized view. If you do not rescope the `REF` column, then the materialized view retains the `REF` scope of the master.

For example, suppose you create the customers_with_ref master table at the orc1.example.com master site using the following statements:

```
-- Create the user-defined data type cust_address_typ.
CREATE TYPE oe.cust_address_typ AS OBJECT
   (street_address     VARCHAR2(40),
    postal_code        VARCHAR2(10),
    city               VARCHAR2(30),
    state_province     VARCHAR2(10),
    country_id         CHAR(2));
/

-- Create the object table cust_address_objtab.
CREATE TABLE oe.cust_address_objtab OF oe.cust_address_typ;

-- Create table with REF to cust_address_typ.
CREATE TABLE oe.customers_with_ref (
     customer_id        NUMBER(6) PRIMARY KEY,
     cust_first_name    VARCHAR2(20),
     cust_last_name     VARCHAR2(20),
     cust_address       REF oe.cust_address_typ
                          SCOPE IS oe.cust_address_objtab);
```

Assuming the cust_address_typ exists at the materialized view site with the same object identifier as the type at the master site, you can create a cust_address_objtab_mv object materialized view using the following statement:

```
CREATE MATERIALIZED VIEW oe.cust_address_objtab_mv OF oe.cust_address_typ AS
   SELECT * FROM oe.cust_address_objtab@orc1.example.com;
```

Now, you can create a materialized view of the customers_with_ref master table and rescope the REF to the cust_address_objtab_mv materialized view using the following statement:

```
CREATE MATERIALIZED VIEW oe.customers_with_ref_mv
   (SCOPE FOR (cust_address) IS oe.cust_address_objtab_mv)
   AS SELECT * FROM oe.customers_with_ref@orc1.example.com;
```

If you want to use the SCOPE FOR clause when you create a materialized view, then remember to create the materialized view or table specified in the SCOPE FOR clause first. Otherwise, you cannot specify the SCOPE FOR clause during materialized view creation. For example, if you had created the customers_with_ref_mv materialized view before you created the cust_address_objtab_mv materialized view, then you could not use the SCOPE FOR clause when you created the customers_with_ref_mv materialized view. In this case, the REFs are considered dangling because they point back to the object table at the remote master site.

However, even if you do not use the SCOPE FOR clause when you are creating a materialized view, you can alter the materialized view to specify a SCOPE FOR clause. For example, you can alter the customers_with_ref_mv materialized view with the following statement:

```
ALTER MATERIALIZED VIEW oe.customers_with_ref_mv
   MODIFY SCOPE FOR (cust_address) IS oe.cust_address_objtab_mv;
```

Unscoped REF Columns If you create a materialized view based on a remote master with an unscoped REF column, then the REF column is created in the materialized view, but the REFs are considered dangling because they point to a remote database.

Logging REF Columns in the Materialized View Log If necessary, you can log REF columns in the materialized view log.

> **See Also:** "Logging Columns in the Materialized View Log" on page 6-13 for more information

REFs Created Using the WITH ROWID Clause If the `WITH ROWID` clause is specified for a `REF` column, then Oracle maintains the rowid of the object referenced in the `REF`. Oracle can find the object referenced directly using the rowid contained in the `REF`, without the need to fetch the rowid from the OID index. Therefore, you use the `WITH ROWID` clause to specify a rowid hint. The `WITH ROWID` clause is not supported for scoped `REF`s.

Replicating a `REF` created using the `WITH ROWID` clause results in an incorrect rowid hint at each replication site except the site where the `REF` was first created or modified. The `ROWID` information in the `REF` is meaningless at the other sites, and Oracle does not correct the rowid hint automatically. Invalid rowid hints can cause performance problems. In this case, you can use the `VALIDATE STRUCTURE` option of the `ANALYZE TABLE` statement to determine which rowid hints at each replication site are incorrect.

> **See Also:** *Oracle Database SQL Language Reference* for more information about the `ANALYZE TABLE` statement

Materialized View Registration at a Master Site or Master Materialized View Site

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

Viewing Information about Registered Materialized Views

A level 1 materialized view or materialized view group is registered at its master site. A level 2 or higher multitier materialized view or materialized view group is registered at its master materialized view site, not at the master site. You can query the `DBA_REGISTERED_MVIEWS` data dictionary view at a master site or master materialized view site to list the following information about a remote materialized view:

- The owner, name, and database that contains the materialized view
- The materialized view's defining query
- Other materialized view characteristics, such as its refresh method

You can also query the `DBA_MVIEW_REFRESH_TIMES` view at a master site or master materialized view site to obtain the last refresh times for each materialized view. Administrators can use this information to monitor materialized view activity and coordinate changes to materialized view sites if a master table or master materialized view must be dropped, altered, or relocated.

Internal Mechanisms

Oracle automatically registers a materialized view at its master site or master materialized view site when you create the materialized view, and unregisters the materialized view when you drop it. The same applies to materialized view groups.

When you drop a master materialized view, Oracle does not automatically drop the materialized views based on it. You must drop these materialized views manually. If

you do not drop such a materialized view and the materialized view tries to refresh to a master materialized view that has been dropped, Oracle returns an error.

For example, suppose a materialized view named `orders_lev1` is based on the `oe.orders` master table, and a materialized view named `orders_lev2` is based on `orders_lev1`. If you drop `orders_lev1`, `orders_lev2` remains intact. However, if you try to refresh `orders_lev2`, Oracle returns an error because `orders_lev1` no longer exists.

Caution: Oracle cannot guarantee the registration or unregistration of a materialized view at its master site or master materialized view site during the creation or drop of the materialized view, respectively. If Oracle cannot successfully register a materialized view during creation, then you must complete the registration manually using the `REGISTER_MVIEW` procedure in the `DBMS_MVIEW` package. If Oracle cannot successfully unregister a materialized view when you drop the materialized view, then the registration information for the materialized view persists in the master site or master materialized view site until it is manually unregistered. It is possible that complex materialized views might not be registered.

Manual Materialized View Registration

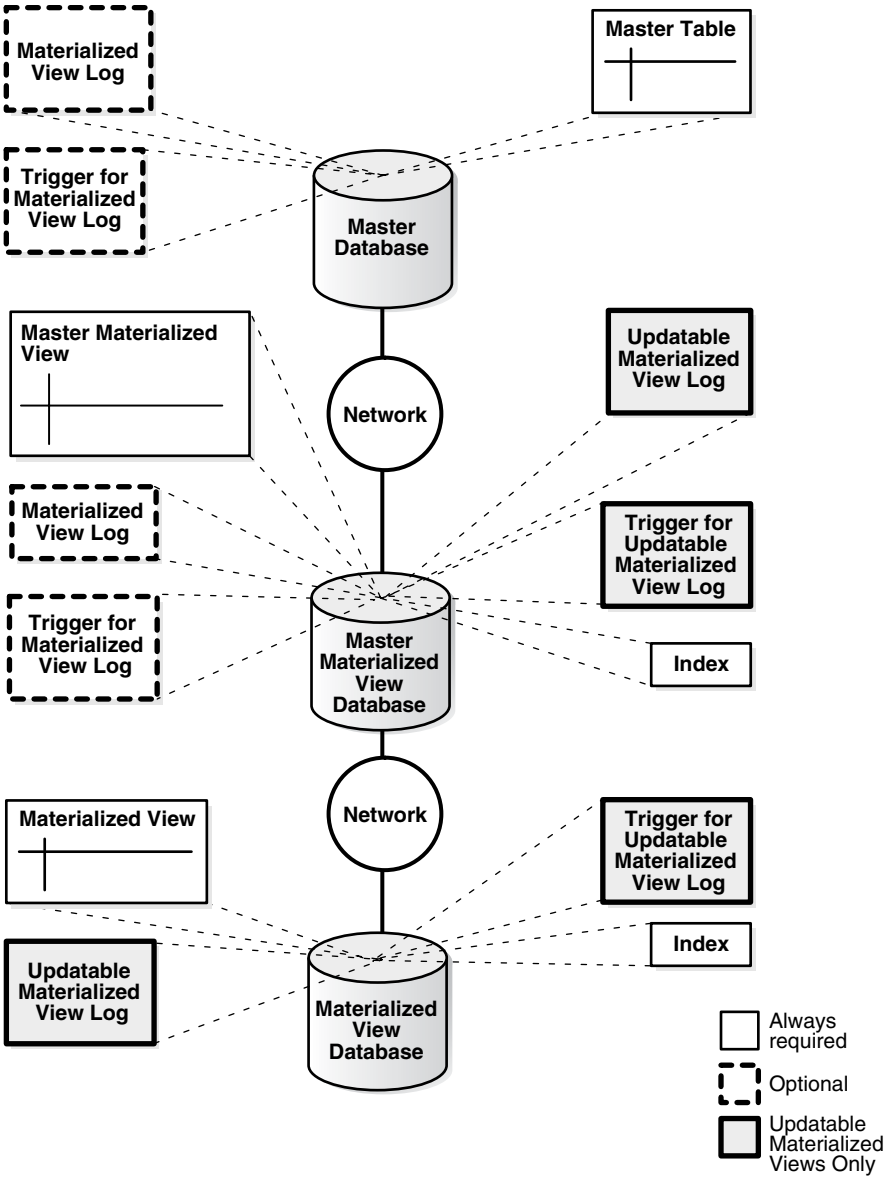
If necessary, you can maintain registration manually. Use the `REGISTER_MVIEW` and `UNREGISTER_MVIEW` procedures of the `DBMS_MVIEW` package at the master site or master materialized view site to add, modify, or remove materialized view registration information.

> **See Also:** The `REGISTER_MVIEW` and `UNREGISTER_MVIEW` procedures are described in the *Oracle Database PL/SQL Packages and Types Reference*

Materialized View Architecture

The objects used in materialized view replication are depicted in Figure 3–10. Some of these objects are optional and are used only as needed to support the created materialized view environment. For example, if you have a read-only materialized view, then you do not have an updatable materialized view log nor an internal trigger at the materialized view site. Also, if you have a complex materialized view that cannot be fast refreshed, then you might not have a materialized view log at the master site.

Figure 3–10 Materialized View Replication Objects



Notice that a master materialized view can have both a materialized view log and an updatable materialized view log. Ensure that you account for the extra space required by these logs when you are planning for your master materialized view site.

This section contains these topics:

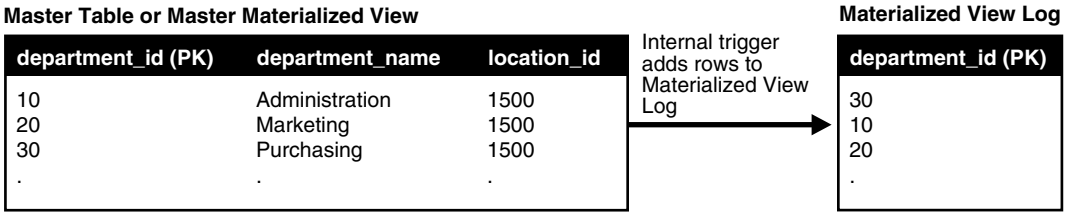
- Master Site and Master Materialized View Site Mechanisms
- Materialized View Site Mechanisms
- Organizational Mechanisms
- Refresh Process

Master Site and Master Materialized View Site Mechanisms

The three mechanisms displayed in Figure 3–11 are required at a master site and at a master materialized view site to support fast refreshing of materialized views.

> **Note:** Master materialized views contain the mechanisms described in "Materialized View Site Mechanisms" on page 3-40 in addition to the mechanisms described in this section.

Figure 3–11 Master Site and Master Materialized View Site Objects



Master Table or Master Materialized View

The master table or master materialized view is the basis for the materialized view. A master table is located at the target master site while a master materialized view is located at a master materialized view site. If the master is a master table, then this table can be involved in both materialized view replication and multimaster replication. Remember that a materialized view points to only one master site or master materialized view site. Changes made to the master table or master materialized view, as recorded by the materialized view log, are propagated to the materialized view during the refresh process.

> **Note:** Fast refreshable materialized views must be based on master tables, master materialized views, or synonyms of master tables or master materialized views. Complete refresh must be used for a materialized view based on a view.

Internal Trigger for the Materialized View Log

When changes are made to the master table or master materialized view using DML, an internal trigger records information about the affected rows in the materialized view log. This information includes the values of the primary key, rowid, or object id, or both, as well as the values of the other columns logged in the materialized view log. This is an internal `AFTER ROW` trigger that is automatically activated when you create a materialized view log for the target master table or master materialized view. It inserts a row into the materialized view log whenever an `INSERT`, `UPDATE`, or `DELETE` statement modifies the table's data. This trigger is always the last trigger to fire.

> **Note:** When the materialized view contains a subquery, you might need to log columns referenced in a subquery. See "Data Subsetting with Materialized Views" on page 3-12 for information about subquery materialized views and "Logging Columns in the Materialized View Log" on page 6-13 for more information about the columns that must be logged.

Materialized View Log

A materialized view log is required on a master if you want to perform a fast refresh on materialized views based on the master. When you create a materialized view log for a master table or master materialized view, Oracle creates an underlying table as the materialized view log. A materialized view log can hold the primary keys, rowids, or object identifiers of rows, or both, that have been updated in the master table or master materialized view. A materialized view log can also contain other columns to support fast refreshes of materialized views with subqueries.

The name of a materialized view log's table is `MLOG$_`*master_name*. The materialized view log is created in the same schema as the target master. One materialized view log can support multiple materialized views on its master table or master materialized view. As described in the previous section, the internal trigger adds change information to the materialized view log whenever a DML transaction has taken place on the target master.

Following are the types of materialized view logs:

- **Primary Key**: The materialized view records changes to the master table or master materialized view based on the primary key of the affected rows.
- **Row ID**: The materialized view records changes to the master table or master materialized view based on the rowid of the affected rows.
- **Object ID**: The materialized view records changes to the master object table or master object materialized view based on the object identifier of the affected row objects.
- **Combination**: The materialized view records changes to the master table or master materialized view based any combination of the three options. It is possible to record changes based on the primary key, the `ROWID`, and the object identifier of the affected rows. Such a materialized view log supports primary key, `ROWID`, and object materialized views, which is helpful for environments that have all three types of materialized views based on a master.

A combination materialized view log works in the same manner as a materialized view log that tracks only one type of value, except that more than one type of value is recorded. For example, a combination materialized view log can track both the primary key and the rowid of the affected row are recorded.

Though the difference between materialized view 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 materialized views and materialized view logs makes reorganizing and truncating your master tables difficult because it prevents your `ROWID` materialized views from being fast refreshed. If you reorganize or truncate your master table, then your rowid materialized view must be `COMPLETE` refreshed because the rowids of the master table have changed.

Note:

- You use the `BEGIN_TABLE_REORGANIZATION` and `END_TABLE_REORGANIZATION` procedures in the `DBMS_MVIEW` package to reorganize a master table. See the *Oracle Database PL/SQL Packages and Types Reference* for more information.
- Online redefinition of tables is another possible way to reorganize master tables, but online redefinition is not allowed on master tables with materialized view logs, master materialized views, and materialized views. Online redefinition is allowed on master tables that do not have materialized view logs. See the *Oracle Database Administrator's Guide* for more information about online redefinition of tables.
- Materialized view logs do not support columns that have been encrypted using transparent data encryption.

Materialized View Logs on Object Tables You can create materialized view logs on object tables. For example, the following SQL statement creates the `categories_typ` user-defined type:

```
CREATE TYPE oe.category_typ AS OBJECT
   (category_name           VARCHAR2(50),
    category_description    VARCHAR2(1000),
    category_id             NUMBER(2));
/
```

When you create an object table based on this type, you can either specify that the object identifier should be system-generated or primary key-based:

```
CREATE TABLE oe.categories_tab_sys OF oe.category_typ
    (category_id    PRIMARY KEY)
    OBJECT ID SYSTEM GENERATED;

CREATE TABLE oe.categories_tab_pkbased OF oe.category_typ
    (category_id    PRIMARY KEY)
    OBJECT ID PRIMARY KEY;
```

When you create a materialized view log on an object table, you must log the object identifier by specifying the `WITH OBJECT ID` clause, but you can also specify that the primary key is logged if the object identifier is primary key-based.

For example, the following statement creates a materialized view log for the `categories_tab_sys` object table and specifies that the object identifier column be logged:

```
CREATE MATERIALIZED VIEW LOG ON oe.categories_tab_sys
   WITH OBJECT ID;
```

The following statement creates a materialized view log for the `categories_tab_pkbased` object table and specifies that the primary key column be logged along with the object identifier column:

```
CREATE MATERIALIZED VIEW LOG ON oe.categories_tab_pkbased
   WITH OBJECT ID, PRIMARY KEY;
```

Restriction on Import of Materialized View Logs to a Different Schema Materialized view logs are exported with the schema name explicitly given in the DDL statements. Therefore, materialized view logs cannot be imported into a schema that is different than the schema from which they were exported. An error is written to the import log file and the items are not imported if you attempt an import using the Data Pump Import utility that specifies the `REMAP_SCHEMA` import parameter to import an export dump file that contains materialized view logs in the specified schema.

Materialized View Site Mechanisms

When a materialized view is created, additional mechanisms are created at the materialized view site to support the materialized view. Specifically, at least one index is created. If you create an updatable materialized view, then an internal trigger and a local log (the updatable materialized view log) are also created at the materialized view site.

Note:

- If the materialized view site is a master materialized view site, then it contains the mechanisms described in the previous section in addition to the mechanisms described in this section. See "Master Site and Master Materialized View Site Mechanisms" on page 3-37.
- The size limit for a materialized view name is 30 bytes. If you try to create a materialized view with a name larger than 30 bytes, Oracle returns an error.

See Also: Figure 3–10, "Materialized View Replication Objects" on page 3-36

Index

At least one index is created at the remote materialized view site for each primary key and `ROWID` materialized view. For a primary key materialized view, the index corresponds to the primary key of the target master table or master materialized view and includes `_PK` in its name. A number is appended if an index with the same name already exists at the materialized view site. For a `ROWID` materialized view, the index is on the `ROWID` column and includes `I_SNAP$_` in its name. Additional indexes can be created by Oracle at the remote materialized view site to support fast refreshing of materialized views with subqueries.

Updatable Materialized View Log

An updatable materialized view log (`USLOG$_`*materialized_view_name*) is used to determine which rows must be overwritten or removed from a materialized view during a fast refresh. A read-only materialized view does not create this log, and Oracle does not use this log during a complete refresh because, in this case, the entire materialized view is replaced.

If there is a conflict between an updatable materialized view and a master, then, during a refresh, the conflict might result in an entry in the updatable materialized view log that is not in the materialized view log at the master site or master materialized view site. In this case, Oracle uses the updatable materialized view log to remove or overwrite the row in the materialized view.

The updatable materialized view log is also used when you fast refresh a writeable materialized view, as illustrated in the following scenario:

1. A user inserts a row into a writeable materialized view that has a remote master. Because the materialized view is writeable and not updatable, the transaction is not stored in the deferred transaction queue at the materialized view site.
2. Oracle logs information about this insert in the updatable materialized view log.
3. The user fast refreshes the materialized view.
4. Oracle uses the information in the updatable materialized view log and deletes the inserted row. A materialized view must be an exact copy of the master when the fast refresh is complete. Therefore, Oracle must delete the inserted row.

Internal Trigger for the Updatable Materialized View Log

Like the internal trigger at the master site or master materialized view site, the internal trigger at the materialized view site records DML changes applied to an updatable materialized view in the `USLOG$_`*materialized_view_name* log. A read-only materialized view does not create this trigger.

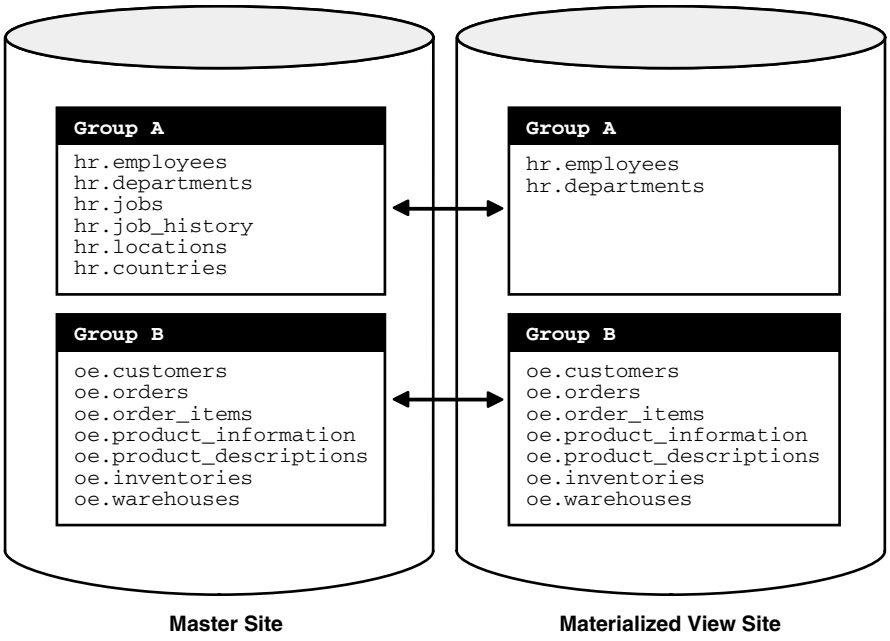
Organizational Mechanisms

In addition to the materialized view mechanisms described in the previous section, several other mechanisms organize the materialized views at the materialized view site. These mechanisms maintain organizational consistency between the materialized view site and its master site or master materialized view site, as well as transactional (read) consistency with the target replication group. These mechanisms are materialized view groups and refresh groups.

Materialized View Groups

A **materialized view group** in a replication system maintains a partial or complete copy of the objects at the target replication group at its master site or master materialized view site. Materialized view groups cannot span the boundaries of the replication group at the master site or master materialized view site. Figure 3–12 displays the correlation between Groups A and B at the master site and Groups A and B at the materialized view site.

Figure 3–12 Materialized View Groups Correspond with Master Groups



Group A at the materialized view site (see Figure 3–12) contains only some of the objects in the corresponding Group A at the master site. Group B at the materialized view site contains all objects in Group B at the master site. Under no circumstances, however, could Group B at the materialized view site contain objects from Group A at the master site. As illustrated in Figure 3–12, a materialized view group has the same name as the master group on which the materialized view group is based. For example, a materialized view group based on a `personnel` master group is also named `personnel`.

In addition to maintaining organizational consistency between materialized view sites and their master sites or master materialized view sites, materialized view groups are required for supporting updatable materialized views. If a materialized view does not belong to a materialized view group, then it must be a read-only or writeable materialized view.

Materialized View Group Owners A materialized view group owner enables you to have multiple materialized view groups based on a single replication group at a master site or master materialized view site. For example, if you need to support multiple users within the same database at a materialized view site, then you might want to create multiple materialized view groups for a target master group. Doing so enables you to define different subqueries for your materialized view definitions in each materialized view group, and allows each user to access only his or her subset of the data.

Defining multiple materialized view groups gives you the ability to control data sets at a group level. For example, if you create different materialized view groups named `hr`, `personnel`, and `manufacturing` for these departments, then you can administer each department as a group, instead of as individual objects. For example, you can drop the objects as a group.

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

After you have defined your materialized view group with the addition of a group owner, you add your materialized view objects to the target materialized view group by defining the same group owner. When using a group owner, remember that each materialized view object must have a unique name. If a single materialized view site has multiple materialized view groups based on the same replication group at the master site or master materialized view site, then a materialized view group's object names cannot have the same name as materialized view objects in another materialized view group. To avoid conflicting names, you can append the group owner name to the end of your object name. For example, if you have group owners `hr` and `ac`, then you might name the `employees` materialized view object as `employees_hr` and `employees_ac`, respectively.

Additionally, all materialized view groups that are based on the same replication group at a single materialized view site must "point" to the same master site or master materialized view site. For example, if the `hr_repg` materialized view group owned by `hr` is based on the associated master group at the `orc1.example.com` master site, then the `hr_repg` materialized view group owned by `personnel` must also be based on the associated master group at `orc1.example.com`, assuming that the `hr` and `personnel` owned groups are at the same materialized view site.

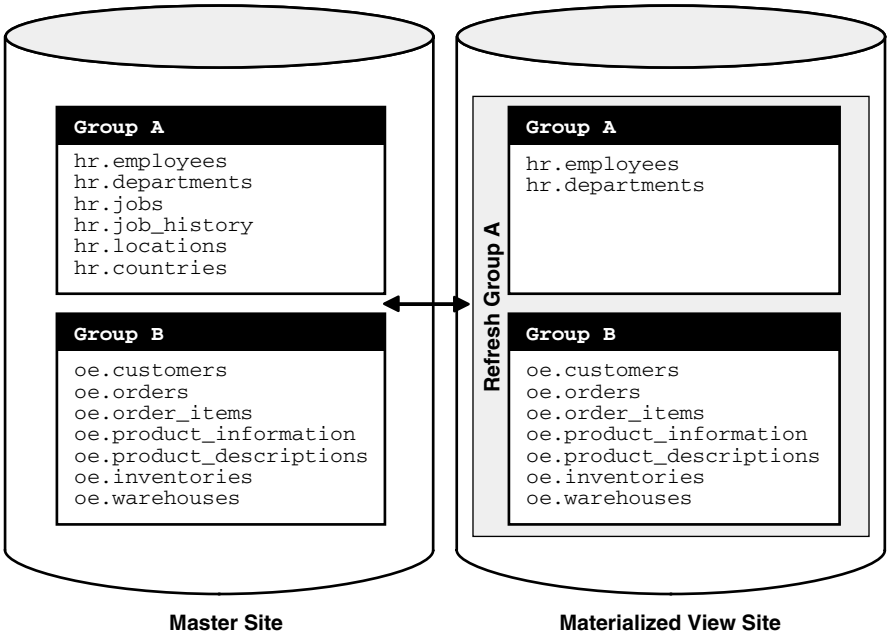
> **See Also:** *Oracle Database Advanced Replication Management API Reference* for more information about defining a group owner using the replication management API

Refresh Groups

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

As illustrated in Figure 3–13, a refresh group can contain materialized views from more than one materialized view group to maintain transactional (read) consistency across replication group boundaries.

Figure 3–13 Refresh Groups Can Contain Objects from Multiple Materialized View Groups



While you might want to define a single refresh group for each materialized view group, it might be more efficient to use one large refresh group that contains objects from multiple materialized view groups. Such a configuration reduces the amount of "overhead" needed to refresh your materialized views. A refresh group can contain up to 400 materialized views.

One configuration that you want to avoid is using multiple refresh groups to refresh the contents of a single materialized view group. Using multiple refresh groups to refresh the contents of a single materialized view group might introduce inconsistencies in the materialized view data, which can cause referential integrity problems at the materialized view site. Only use this type of configuration 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 materialized views in small refresh groups, assuming that the materialized views in the groups are similar. For example, refreshing a refresh group with 100 materialized views is faster than refreshing five refresh groups with 20 materialized views each. Also, large refresh groups enable you to refresh a greater number of materialized views with only one call to the replication management API.

During the refresh of a refresh group, each materialized view in the group is locked at the materialized view site for the amount of time required to refresh all of the materialized views in the refresh group. This locking is required to prevent users from updating the materialized views during the refresh operation, because updates can make the data inconsistent. Therefore, having smaller refresh groups means that the materialized views 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, then 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.

Advanced Replication includes an optimization for null refresh. That is, if there were no changes to the master tables or master materialized views since the last refresh for a particular materialized view, then almost no extra time is required for the materialized view during materialized view group refresh.

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

Table 3–3 Large and Small Refresh Groups

Advantages of Large Refresh Groups	Advantages of Small Refresh Groups
■ Refreshes faster than an equal number of materialized views in multiple refresh groups	■ Materialized views 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 materialized view's data does not necessarily match the current data of its master table or master materialized view at all times. A materialized view is a transactionally (read) consistent reflection of its master as the data existed at a specific point in time (that is, at creation or when a refresh occurs). To keep a materialized view's data relatively current with the data of its master, the materialized view must be refreshed periodically. A **materialized view refresh** is an efficient batch operation that makes a materialized view reflect a more current state of its master table or master materialized view.

A refresh of an updatable materialized view first pushes the deferred transactions at the materialized view site to its master site or master materialized view site. Then, the data at the master site or master materialized view site is pulled down and applied to the materialized view.

A row in a master table can be updated many times between refreshes of a materialized view, but the refresh updates the row in the materialized view only once with the current data. For example, a row in a master table might be updated 10 times since the last refresh of a materialized view, but the result is still only one update of the corresponding row in the materialized view during the next refresh.

Decide how and when to refresh each materialized view to make it more current. For example, materialized views based on masters that applications update often might require frequent refreshes. In contrast, materialized views based on relatively static masters usually require infrequent refreshes. In summary, analyze application characteristics and requirements to determine appropriate materialized view refresh intervals.

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

Refresh Types

Oracle can refresh a materialized view using either a fast, complete, or force refresh.

Complete Refresh To perform a **complete refresh** of a materialized view, the server that manages the materialized view executes the materialized view's defining query, which essentially re-creates the materialized view. To refresh the materialized view, the result set of the query replaces the existing materialized view data. Oracle can perform

a complete refresh for any materialized view. 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.

If you perform a complete refresh of a master materialized view, then the next refresh performed on any materialized views based on this master materialized view must be a complete refresh. If a fast refresh is attempted for such a materialized view after its master materialized view has performed a complete refresh, then Oracle returns the following error:

```
ORA-12034 mview log is younger than last refresh
```

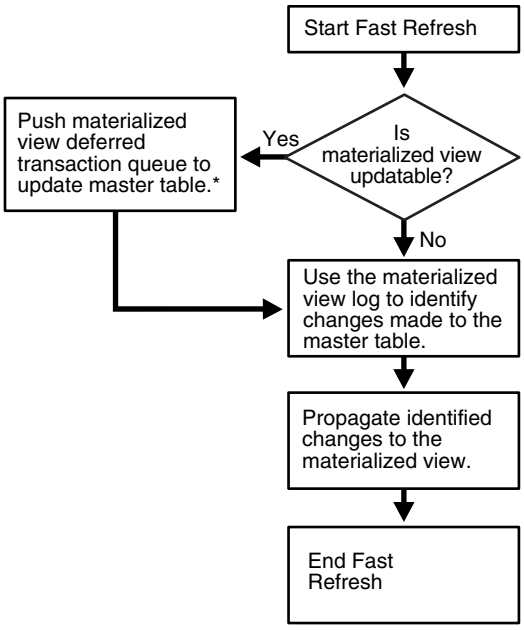
> **Note:** If complete refresh is used for a materialized view, then set its `PCTFREE` to `0` and `PCTUSED` to `99` for maximum efficiency.

Fast Refresh To perform a **fast refresh**, the master that manages the materialized view first identifies the changes that occurred in the master since the most recent refresh of the materialized view and then applies these changes to the materialized view. 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 materialized views only when the master table or master materialized view has a materialized view log. Also, for fast refreshes to be faster than complete refreshes, each join column in the `CREATE MATERIALIZED VIEW` statement must have an index on it.

After a direct path load on a master table or master materialized view 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 masters. Examples of these operations include `INSERT` statements with an `APPEND` hint and `INSERT . . . SELECT * FROM` statements.

Figure 3–14 Fast Refresh of a Materialized View



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

If you have materialized views based on partitioned master tables, then you might be able to use Partition Change Tracking (PCT) to identify which materialized view rows correspond to a particular partition. PCT is also used to support fast refresh after partition maintenance operations on a materialized view's master table. PCT-based refresh on a materialized view is possible only if a number of conditions are satisfied.

> **See Also:** *Oracle Database Data Warehousing Guide* for information about PCT and about PCT-based refresh

If you have updatable multitier materialized views, then DML changes made to the multitier materialized view can be pulled back to this materialized view multiple times to ensure data consistency after each refresh of a materialized view. This behavior is best illustrated through an example.

Consider a replication environment with the following characteristics:

- Master site `orc1.example.com` has the `oe.customers` table.
- Level 1 materialized view site `ca.us` has the `oe.customers_region` updatable materialized view based on the `oe.customers` table at `orc1.example.com`.
- Level 2 updatable materialized view site `sf.ca` has the `oe.customers_sf` updatable materialized view based on the `oe.customers_region` materialized view at `ca.us`.

Given these characteristics, the following scenario might follow:

1. The `credit_limit` for a customer is changed from `3000` to `5000` in the `oe.customers_sf` updatable materialized view at `sf.ca`.
2. Oracle enters the change in the deferred transaction queue at `sf.ca`.

3. A fast refresh of the level 2 materialized view `oe.customers_sf` pushes the new value for the `credit_limit` to `oe.customers_region` materialized view at `ca.us`.
4. The change is applied to the `oe.customers_region` materialized view at `ca.us`.
5. The update for the `credit_limit` at the `ca.us` site is recorded in both the deferred transaction queue and the materialized view log a this level 1 materialized view site.
6. A fast refresh of the level 2 materialized view `oe.customers_sf` pulls the `credit_limit` value of `5000` back down to this materialized view at `sf.ca`.
7. A fast refresh of the level 1 materialized view `oe.customers_region` pushes the new value for the `credit_limit` to `oe.customers` master table at `orc1.example.com`.
8. The change is applied to the `oe.customers` master table at `orc1.example.com`.
9. The update for the `credit_limit` at the `orc1.example.com` site is recorded in both the deferred transaction queue and the materialized view log a this master site.
10. A new fast refresh of the level 1 materialized view `oe.customers_region` pulls the `credit_limit` value of `5000` back down to this materialized view at `ca.us`.
11. The update for the `credit_limit` at the `ca.us` site is recorded in the materialized view log a this level 1 materialized view site.
12. A fast refresh of the level 2 materialized view `oe.customers_sf` pulls the `credit_limit` value of `5000` back down to this materialized view at `sf.ca` for a second time.

Force Refresh To perform a **force refresh** of a materialized view, the server that manages the materialized view attempts 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 materialized view to refresh if a fast refresh is not possible.

Initiating a Refresh

When creating a refresh group, you can configure the group so that Oracle automatically refreshes the group's materialized views at scheduled intervals. Conversely, you can omit scheduling information so that the refresh group must 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 automatic 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, then 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.

The following are examples of simple date expressions that you can use to specify an interval:

- An interval of one hour is specifies as:

```
SYSDATE + 1/24
```

- An interval of seven days is specifies as:

```
SYSDATE + 7
```

> **See Also:** *Oracle Database Administrator's Guide* and *Oracle Database SQL Language Reference* for more information about date arithmetic

On-Demand Refresh Scheduled materialized view refreshes might not always be the appropriate solution for your environment. For example, immediately following a bulk data load into a master table, dependent materialized views no longer represent the master table's data. Rather than wait for the next scheduled automatic group refreshes, you can manually refresh dependent materialized view groups to immediately propagate the new rows of the master table to associated materialized views.

You might also want to refresh your materialized views on-demand when your materialized views 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 materialized views when they are ready to transfer the day's orders to the server after establishing a dial-up network connection.

The following example illustrates an on-demand refresh of the `hr_refg` refresh group:

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

> **Note:** Do not use the `DBMS_MVIEW.REFRESH_ALL_MVIEWS` or the `DBMS_MVIEW.REFRESH_DEPENDENT` procedure to refresh materialized views used in a replication environment. Instead, use the `DBMS_REFRESH.REFRESH` or the `DBMS_MVIEW.REFRESH` procedure to refresh materialized views in a replication environment.

Constraints and Refresh

To avoid any integrity constraint violations during refresh of materialized views, make non primary key integrity constraints on each materialized view deferrable. This requirement includes LOB columns with `NOT NULL` constraints. In addition, all materialized views that are related by foreign key constraints should be refreshed together or in the same refresh group.

Note:

- Primary key constraints on materialized views might or might not be deferrable.
- A `DELETE CASCADE` constraint used with an updatable materialized view must be deferrable.

See Also: *Oracle Database SQL Language Reference* for information about making constraints deferrable

4

Deployment Templates Concepts and Architecture

This chapter introduces deployment templates and describes how to use them to easily and efficiently distribute materialized view environments.

This chapter contains these topics:

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

> **Note:** Read Chapter 3, "Materialized View Concepts and Architecture" before you create a deployment template. Understanding materialized views 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 materialized view environment. Before learning about the concepts, architecture, and use of deployment templates, consider 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, DBAs have been required to develop a deployment method of their own. Typically, the DBA was responsible for developing a very complex script to create the materialized view environment at the remote materialized view site. In addition to building the script, the DBA was often forced to customize data sets at the materialized view site. After 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 preceding 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 Advanced 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 accomplish the following objectives:

Define the materialized view environment once

You define the structure of a materialized view environment once using a deployment template so that each user (site) receives the database infrastructure to support the front-end application.

Customize materialized view sites individually

You use deployment template parameters to customize each materialized view 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 laptop users.

Oracle Deployment Templates Concepts

Oracle offers deployment templates to allow the database administrator to **package** a materialized view environment for easy, custom, and secure deployment. Packaging a deployment template is the process of defining the materialized view environment that will be created by the deployment template. Packaging a deployment template prepares it for **instantiation** at the remote materialized view site. Instantiation creates the materialized view site objects and populates the materialized views with data.

A deployment template can be as simple as a single materialized view with a fixed data set, or as complex as hundreds of materialized views with a dynamic data set based on one or more variables. Deployment template features include the following:

- Centralized control
- Ability to repeatedly deploy a materialized view 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 materialized view environment for deployment, create a deployment template at the master site. This template stores all of the information needed to deploy a materialized view 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 materialized view creation.

This section contains these topics:

- Deployment Templates and the Mass Deployment Goal
- Deployment Template Elements
- Deployment Template Packaging and Instantiation

Deployment Template Elements

Each deployment template contains the "blueprint" for creating the necessary materialized views and related objects at a materialized view site. Specifically, you create the deployment template at the master site, adding the necessary materialized views, triggers, views, and so on to the template as needed to create the materialized view 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
- Template 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 materialized view 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 Deployment Template View Relationships

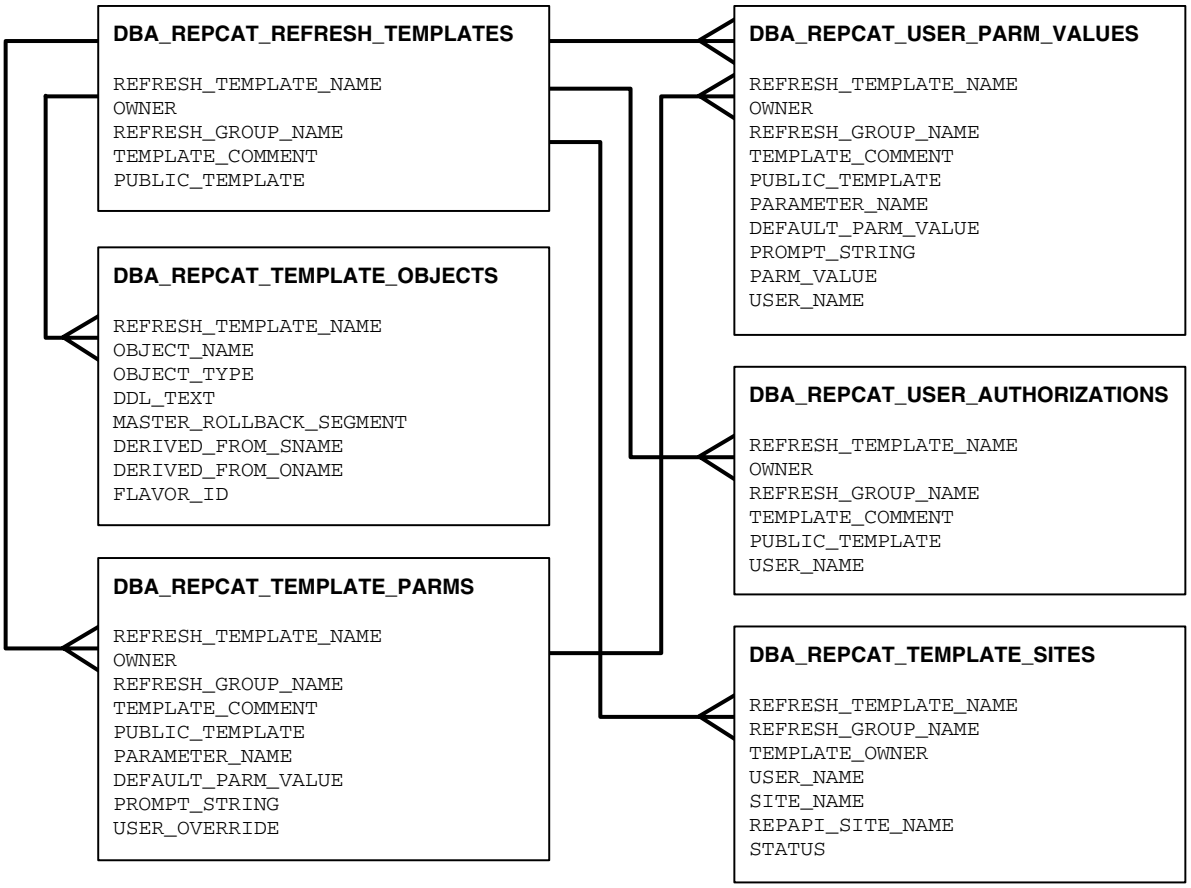
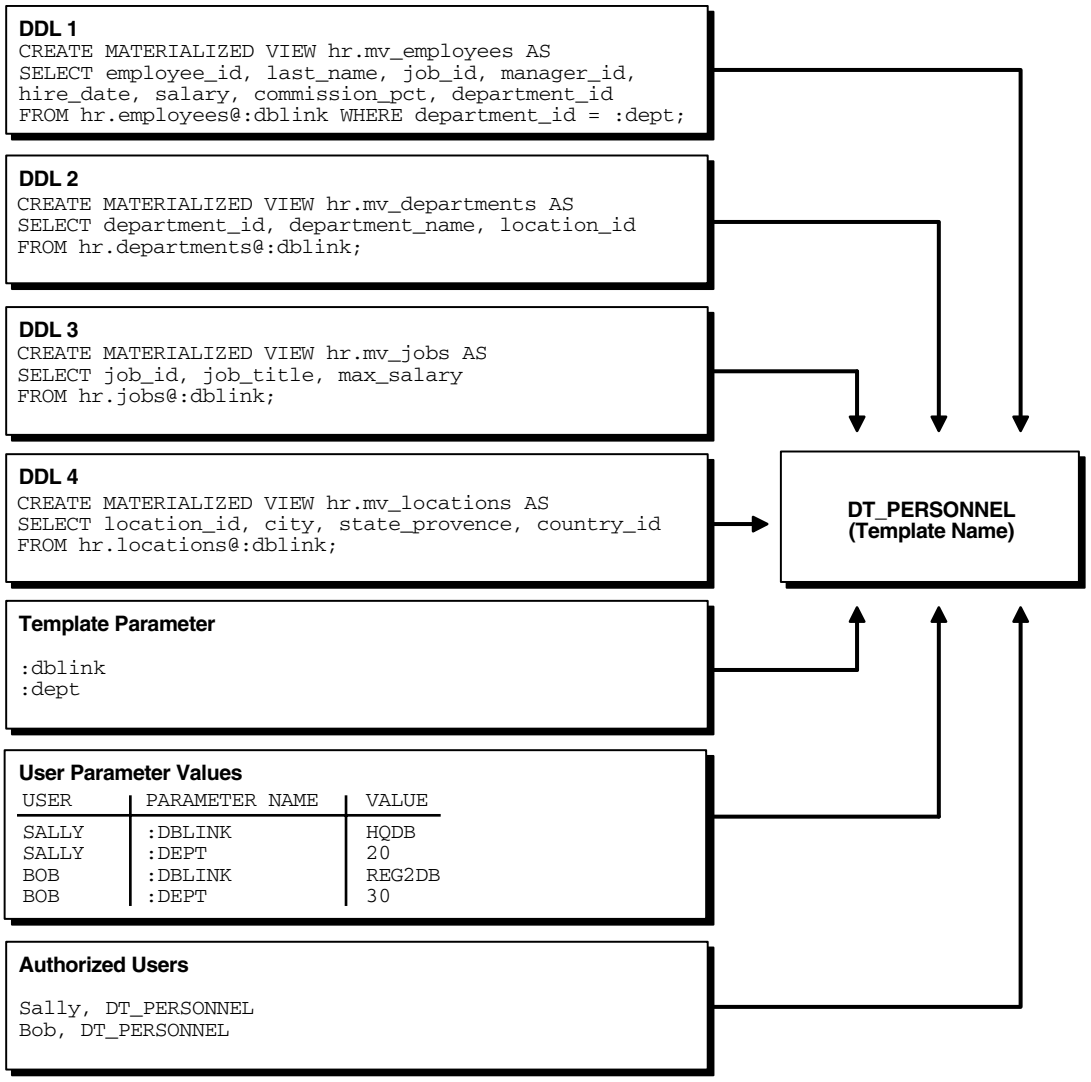


Figure 4–2 Deployment Template Elements Added to Template



Template Object Definitions

After the template has been defined, add objects to the template. When the template is instantiated at the materialized view site, the object DDL (that is, `CREATE MATERIALIZED VIEW`, `CREATE TABLE`, and so on) is executed to create the appropriate objects at the materialized view site.

You can add objects to a deployment template that are based on a existing master objects, but if necessary, you can create a new template object by defining DDL to create the object. Oracle checks any new object DDL to ensure that it is lexically correct, which prevents the execution of faulty DDL. Updatable materialized views added to a deployment template must be based on a table in a master group, but other objects, such as read-only materialized views, can be based on objects that are not in master groups.

In most cases, you add materialized views to the template, but if necessary, you can add other objects. For example, constraints can be added to enforce data integrity at the materialized view site, views can be added for displaying data, or tables can be added for local data storage. In some cases, you might even include all objects for an

application in a deployment template. Materialized views created using a deployment template are automatically added to the refresh group defined for the template.

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

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

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

> **See Also:** "General Template Information" on page 4-3 for more information about the refresh group

Template Parameters

If each target materialized view site requires a data set unique to its site, then 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 materialized view 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 enables 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
fay	east
baer	west

The defining `SELECT` statement for the materialized view is the following:

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

When users fay and baer instantiate template `sales_temp`, their resulting materialized view data sets are the following:

User fay		-	User baer	
cust_id	region	-	cust_id	region
a123	east	-	b123	west
a234	east	-	b234	west
a345	east	-	b345	west
a456	east	-	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 MATERIALIZED VIEW` statement to securely limit the materialized view 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
fay	east
baer	west

Users accessing the materialized view instantiated by user `fay` 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 materialized view cannot view, update, or delete data for region `west`, because the materialized view 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, then 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, create a list of authorized users at the master site. If an unauthorized user attempts to instantiate the target template, then 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 materialized view 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.

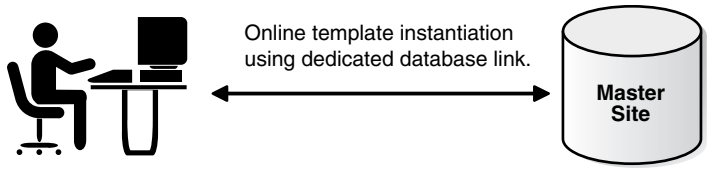
Deployment Template Packaging and Instantiation

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

Online Instantiation

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

Figure 4–3 Online Instantiation



Packaging a deployment template for online instantiation means generating a script file that, when run at the materialized view site, creates the materialized view objects and connects to the master site to populate the materialized views with data. SQL statements such as `CREATE MATERIALIZED VIEW ... AS SELECT` are used to populate the materialized views with data over a network from the master site.

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 materialized view and master sites, which, depending on the size of the materialized view environment created, might increase network traffic.

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

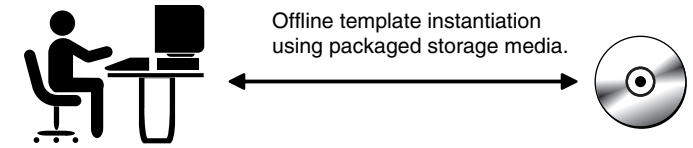
Offline Instantiation

To decrease server loads during peak usage periods and reduce remote connection times, you can 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 materialized view 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 materialized view site. Each materialized view site requires a separate offline instantiation script.

When you package a template for instantiation, the materialized view logs for each master table on which a materialized view is based in the template begin to log changes. The materialized view log for a particular master table does not clear these changes until every materialized view based on the master table refreshes after instantiation. Therefore, to prevent the materialized view log from growing large, the template should be instantiated, and the materialized views 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 operation has the benefit of reducing network traffic and eliminating the need for a constant network connection. However, after instantiation, the data in the materialized view 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.

Offline Instantiation of Multitier Materialized Views When you use deployment templates to create a materialized view site using offline instantiation, the conflict resolution methods defined on the master tables are not pulled down to the materialized view site. These conflict resolution methods might be required to ensure data consistency if you plan to create materialized views based on this materialized view site (multitier materialized views). If you use online instantiation, then the conflict resolution methods are pulled down during instantiation.

Scenarios for Instantiating a Deployment Template

Table 4–1 summarizes the scenarios for instantiating of a deployment template.

Table 4–1 Scenarios for Instantiating a Deployment Template

Type of Instantiation	Description
Offline	The user runs the offline instantiation script with SQL*Plus. The offline instantiation script contains both `CREATE` statements to create materialized view site objects and `INSERT` statements to populate the materialized views with data.
Online	The user runs the online instantiation script with SQL*Plus. The online instantiation script contains `CREATE` statements to create materialized view site objects. When materialized view objects are created, the online instantiation script connects to the master site and uses `CREATE MATERIALIZED VIEW . . . AS SELECT` statements to create the materialized views and populate them with data.

Either you (the DBA) or the target user can package the deployment template. Either use the Advanced Replication interface's Create Deployment Template 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 can 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 only):

- `DBMS_REPCAT_RGT.INSTANTIATE_OFFLINE` function
- `DBMS_REPCAT_RGT.INSTANTIATE_ONLINE` function

Public functions:

- `DBMS_REPCAT_INSTANTIATE.INSTANTIATE_OFFLINE` function
- `DBMS_REPCAT_INSTANTIATE.INSTANTIATE_ONLINE` function

Note: When you package a deployment template for offline instantiation, the related materialized view logs begin logging for the materialized views that were packaged in the template. This immediate logging enables the remote materialized view site to perform a fast refresh after completing the offline instantiation process. Monitor the materialized view logs to ensure that remote materialized view sites refresh in a timely manner after performing an offline instantiation. Remote materialized view sites that have not refreshed cause the materialized view log to grow quite large, because logging begins when the template is packaged.

See Also: "Preparing Materialized View Sites for Instantiation of Deployment Templates" on page 6-21, and see *Oracle Database Advanced Replication Management API Reference* for information about the functions

Deployment Template Architecture

Oracle uses standard materialized view architecture with deployment templates to distribute materialized view environments quickly and effectively. Deployment templates use the same methods in creating materialized view definitions, refresh characteristics, conflict resolution, and grouping as used when manually building a materialized view 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 is executed when the template is instantiated.

This section contains these topics:

Template Definitions Stored in System Tables

Instead of executing DDL at the materialized view site to immediately create a materialized view environment, the materialized view 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 materialized view site, which creates the necessary materialized view 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.

Use of Standard DDL

Template object definitions are created using the same DDL that is used to create the objects locally at the materialized view site. For example, you can issue the following statement to create a materialized view:

```
CREATE MATERIALIZED VIEW hr.departments_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      department_id, department_name, manager_id, location_id
    FROM hr.departments@orc1.example.com;
```

To add this same materialized view to a deployment template, you can use the Advanced Replication interface's Create Deployment Template Wizard, or execute the `CREATE_TEMPLATE_OBJECT` function, as shown in the following example:

```
DECLARE
   tempstring VARCHAR2(3000);
   a NUMBER;
BEGIN
   tempstring := 'CREATE MATERIALIZED VIEW hr.departments_mv
      REFRESH FAST WITH PRIMARY KEY FOR UPDATE AS SELECT
      department_id, department_name, manager_id, location_id
      FROM hr.departments@orc1.example.com';
   a := DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT (
            refresh_template_name => 'hr_refg_dt',
            object_name => 'departments_mv',
            object_type => 'MATERIALIZED VIEW',
            ddl_text => tempstring,
            master_rollback_seg => 'rbs');
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 preceding function adds the materialized view definition to the deployment template named `dt_mviewenv`. When this particular materialized view is instantiated, the materialized view `mview_test` is created. In addition to creating materialized views, you can add table, trigger, procedure, index, and other object definitions to the deployment template.

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

See Also: `DBMS_REPCAT_RGT.CREATE_TEMPLATE_OBJECT` in the *Oracle Database Advanced Replication Management API Reference* for information about using this function

Packaging and Instantiation Process

When a deployment template is packaged in preparation for remote materialized view site instantiation, the template is being prepared for online or offline instantiation. The instantiation procedure creates the remote materialized view 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 materialized view environment is generated and all template parameter substitutions are performed. Where this generated DDL is stored depends on the type of materialized view client.

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 materialized view site computer, then the online instantiation file must be transferred to the materialized view site for online instantiation.

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 materialized view 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 the Advanced Replication interface's Create Deployment Template Wizard or the replication management API can be used to package a deployment template for offline instantiation.

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 materialized view site computer, then the offline instantiation file must be transferred to the materialized view site for offline instantiation.

When the remote materialized view 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 materialized view 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

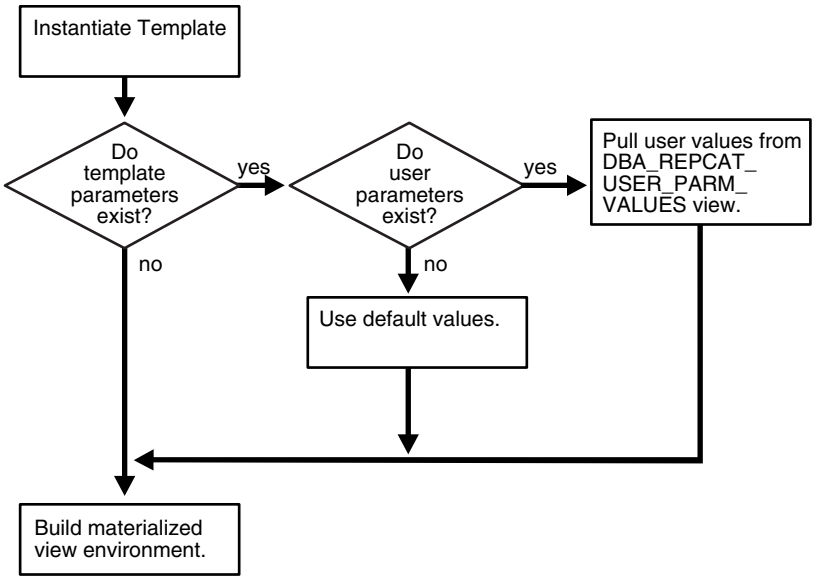
During the online instantiation process, the structure of the materialized view site is created, and the specified data subset is pulled from the master site and stored in the appropriate materialized views. Also, after the remote materialized view site begins the online instantiation process, Oracle evaluates the parameters that have been defined for the deployment template. Any values defined for these parameters are used when the object DDL in the template is executed so that custom data sets can be installed at the remote materialized view site. At the same time, the materialized views are registered at the master site, and the materialized view logs begin logging the changes to the master tables.

Two possible methods 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, then the user parameter values are used when instantiating the template. If no user parameter values have been defined, then 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 materialized view environments use the offline instantiation method to create the necessary materialized view environment. When you package the deployment template, a script or binary file is created to store the DDL needed to create the materialized view environment, the parameter values used during the instantiation process, and the DML necessary to populate the materialized view 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 materialized view site. It can also be transferred using the `DBMS_FILE_TRANSFER` package. 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

A number of possibilities for deployment template packaging and instantiation are available. 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

Type of Instantiation	Package Template Using	Packaging Documentation	Instantiate Template Using	Instantiating Documentation
Offline	Advanced Replication interface Create Deployment Template Wizard	See the Advanced Replication interface's online Help.	Offline Instantiation Script and SQL*Plus	See the Advanced Replication interface's online Help.
Offline	The Replication Management API (PL/SQL Packages and SQL*Plus)	See the instructions for packaging in *Oracle Database Advanced Replication Management API Reference*.	Offline Instantiation Script and SQL*Plus	See the instructions for instantiating a deployment template in *Oracle Database Advanced Replication Management API Reference*.
Online	The Replication Management API (PL/SQL Packages and SQL*Plus)	See the instructions for packaging in *Oracle Database Advanced Replication Management API Reference*.	Online Instantiation Script and SQL*Plus	See the instructions for instantiating a deployment template in *Oracle Database Advanced Replication Management API Reference*.

After Instantiation

After instantiating a deployment template at a remote materialized view site, the structure created is exactly the same as if you had created the materialized view environment locally at the materialized view site. Specifically, Oracle creates the materialized view, 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 materialized view 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 materialized view site. The materialized view site uses the materialized view 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 materialized view log as soon as you package the deployment template.

> **See Also:** "Materialized View Architecture" on page 3-35 for more information

Materialized View Groups

Objects created by an instantiated deployment template are added automatically to a materialized view group with the same name as the object's master group. For example, if you instantiated the `dt_mviewenv` deployment template, which contains objects from the `personnel` and `technical` master groups, then your template objects are added to materialized view groups `personnel` and `technical`, respectively. Remember that a materialized view group helps to maintain organizational consistency with the target master group and, more importantly, is required for updatable materialized views.

> **See Also:** "Materialized View Groups" on page 3-41 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 materialized view objects will be added. After the instantiation process is finished, you can specify that the materialized views in the

refresh group be refreshed automatically at set intervals, assuming a constant network connection to the master site.

You can use the Advanced Replication interface in Oracle Enterprise Manager, or `DBMS_REFRESH.CHANGE` procedure, to change the refresh interval and next refresh data of a refresh group. To change these settings in the Advanced Replication interface, 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 materialized view sites do not have a constant network connection to the master site, then they can refresh their refresh groups on-demand.

The following are examples of simple date expressions that you can use to specify `next_date` and `interval`:

- A `next_date` or `interval` of one hour is specifies as:

  ```
  SYSDATE + 1/24
  ```

- A `next_date` or `interval` of seven days is specifies as:

  ```
  SYSDATE + 7
  ```

> **See Also:** *Oracle Database Administrator's Guide* and *Oracle Database SQL Language Reference* for more information about date arithmetic

Deployment Template Design

The combination of deployment template parameters and subquery subsetting gives the database administrator a powerful tool to administer a widely distributed database environment using subqueries and row-subsetted data. Additional design consideration must be given to column subsetting requirements and data sets needed for a replication environment.

Materialized view data sets are defined based on the materialized view's query, meaning that the user only sees data that complies with the materialized view's defining query. Both row and column subsetting enable you to create materialized views that contain customized data sets. Such materialized views can be helpful for regional offices or sales forces that do not require the complete corporate data set.

This section contains these topics:

- Column Subsetting with Deployment Templates
- Row Subsetting
- Data Sets
- Additional Design Considerations

> **See Also:** "Data Subsetting with Materialized Views" on page 3-12 for more information about data subsetting

Column Subsetting with Deployment Templates

Column subsetting enables you to exclude columns that are in master tables from materialized views. You do this by specifying certain select columns in the `SELECT` statement during materialized view creation. Column subsetting is only possible

through the use of deployment templates. Before you begin assembling your deployment template, consider how to build your templates.

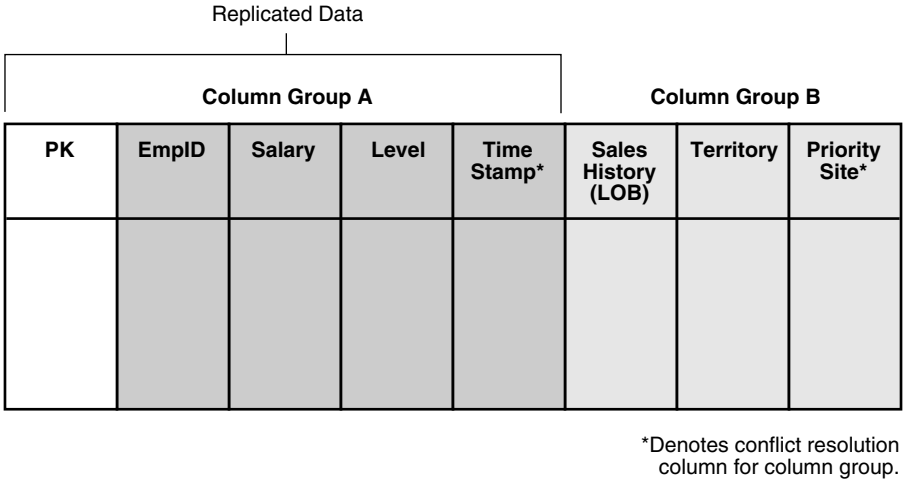
For example, in a mass deployment environment with many "lightweight" clients, you might need to replicate tables that contain LOB data without actually replicating the LOB data itself. This goal can be achieved by excluding the LOB column from the selected columns to be replicated when defining the column subset.

You can select any subset of columns in a read-only materialized view. For an updatable materialized view, the subset of columns must contain the following columns:

- Primary key column(s)
- All columns used for conflict resolution for the replicated columns (see Figure 4–6)

> **Note:** While it is possible to configure column subsetting within a column group, it is not recommended because it can result in data inconsistencies between sites. When using column subsetting, you should eliminate columns at the column group level.

Figure 4–6 Replicate Column-Subsetted Data



If you are adding a materialized view that replicates columns `pk`, `empid`, `salary`, and `level` (illustrated in Figure 4–6), then you also need to include the `Time Stamp` column because it is used for conflict resolution for columns contained in Column Group A.

Note:

- Column subsetting is only available when you add a materialized view to a deployment template using the Advanced Replication interface in Oracle Enterprise Manager. Column subsetting is not available when using the replication management API.
- The master definition site must be available when defining a column subset. If your deployment template contains column-subsetted materialized views from multiple master groups, then the master definition site for each group must be available.

Row Subsetting

Row subsetting enables you to exclude rows that are in master tables from materialized views by using a `WHERE` clause. For example, the following statement creates a materialized view based on the `oe.orders@orc1.example.com` master table and includes only the rows for the sales representative with a `sales_rep_id` number of `173`:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST FOR UPDATE AS
  SELECT * FROM oe.orders@orc1.example.com
  WHERE sales_rep_id = 173;
```

Rows of the `orders` table with a `sales_rep_id` number other than `173` are excluded from this materialized view.

Row Subsetting with an Assignment Table

In some situations, you can benefit from using row subsetting with an **assignment table**. An assignment table lets you relate one entity to another entity in your database, without storing the assignment information in either of the tables for the two entities. This technique is best illustrated through an example.

In the `oe` schema, the `product_id` column is the primary key in the `product_information` table, and the `warehouse_id` column is the primary key in the `warehouses` table. In this schema, the `inventories` table functions as an assignment table because it assigns a product to a warehouse using the `product_id` column and the `warehouse_id` column. These two columns form the primary key of the `inventories` table.

With these three tables in `oe` schema (`inventories`, `product_information`, and `warehouses`), you can track which products are in which warehouses without storing the `product_id` information in the `warehouses` table, nor the `warehouse_id` information in the `product_information` table. To illustrate why this is important, consider what would happen if the `inventories` table did not exist and the `warehouse_id` column was a foreign key in the `product_information` table.

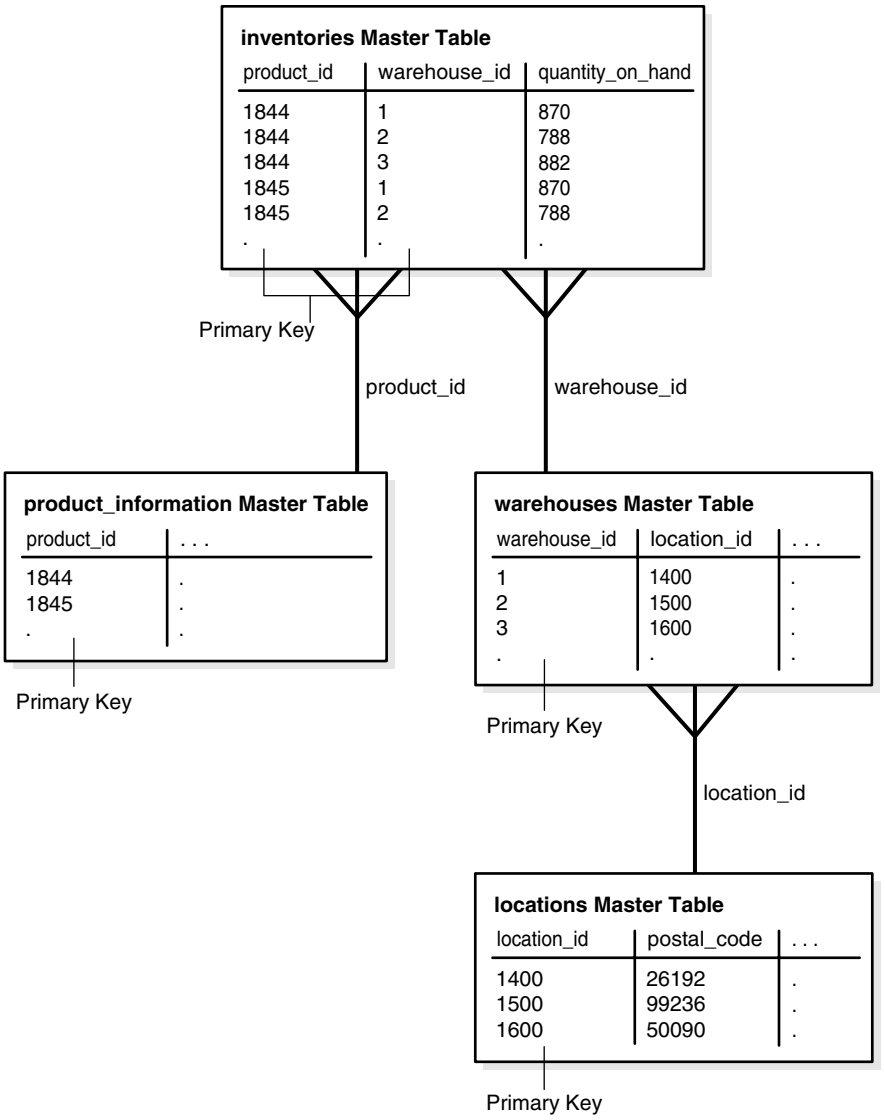
In this case, if a salesperson wants to store product information for the nearest warehouse, then the sales person would need to specify the `warehouse_id` for the warehouse in the `WHERE` clause of the `CREATE MATERIALIZED VIEW` statement. For example, the salesperson might create the materialized view using the following statement:

```
CREATE MATERIALIZED VIEW oe.product_information REFRESH FAST FOR UPDATE AS
  SELECT * FROM oe.product_information@orc1.example.com
  WHERE warehouse_id = 1;
```

The drawback to this configuration is that the `warehouse_id` is "hard coded" into the materialized view definition. If the company closes warehouse 1 or opens a new warehouse that is even closer to the salesperson, then the preceding materialized view definitions would need to be altered or re-created. With this in mind, if you use assignment tables in conjunction with row subsetting in a subquery, then you can easily control changes to a materialized view environment.

In the `oe` schema, the `warehouse_id` column is not part of the `product_information` table. Instead, a product is assigned to a warehouse through the `inventories` table. This relationship between products and warehouses is illustrated in Figure 4–7.

Figure 4–7 Product/Warehouse Relationship



If new warehouses are built or other warehouses are closed, then you can use the `inventories` table to assign products to different warehouses. Besides creating a single point of administration, assignment tables, such as the `inventories` table, used in conjunction with row subsetting in subqueries, can ensure security. For

example, if necessary, you can limit a certain salesperson to see data for some warehouses but not others.

If we assume that each salesperson is responsible for a particular location and only requires product information for products that are stored in a warehouse in that location, then we can use the `inventories` table as an assignment table along with row subsetting in subqueries to create the `product_information` materialized view that contains only the relevant information for a particular salesperson. The following statement provides a salesperson with the proper data:

```
CREATE MATERIALIZED VIEW oe.product_information REFRESH FAST FOR UPDATE AS
  SELECT * FROM oe.product_information@orc1.example.com pi
  WHERE EXISTS
    (SELECT * FROM oe.inventories@orc1.example.com inv
    WHERE pi.product_id = inv.product_id
    AND EXISTS
      (SELECT * FROM oe.warehouses@orc1.example.com w
      WHERE inv.warehouse_id = w.warehouse_id
      AND EXISTS
        (SELECT * FROM hr.locations@orc1.example.com loc
        WHERE w.location_id = loc.location_id
        AND loc.postal_code = :p_code)));
```

Note: To create this `oe.product_information` materialized view, `postal_code` in must be logged in the materialized view log for the `hr.locations` table. See "Logging Columns in the Materialized View Log" on page 6-13 for more information.

The `product_information` materialized view is populated with product information for the products that are stored in the warehouse located at the postal code specified with the *p_code* variable. Notice the *p_code* variable in the last line of the `CREATE MATERIALIZED VIEW` statement.

With this flexibility, managers can easily control materialized view data sets by making simple changes to the `inventories` table, without requiring modification of the SQL for the materialized view creation statements. For example, if a new product is added to a particular warehouse, then the manager would simply add a row to the `inventories` table that assigns the product to the warehouse. After the next materialized view refresh, the data for the product is added to the materialized view site that tracks product information for the warehouse.

Data Sets

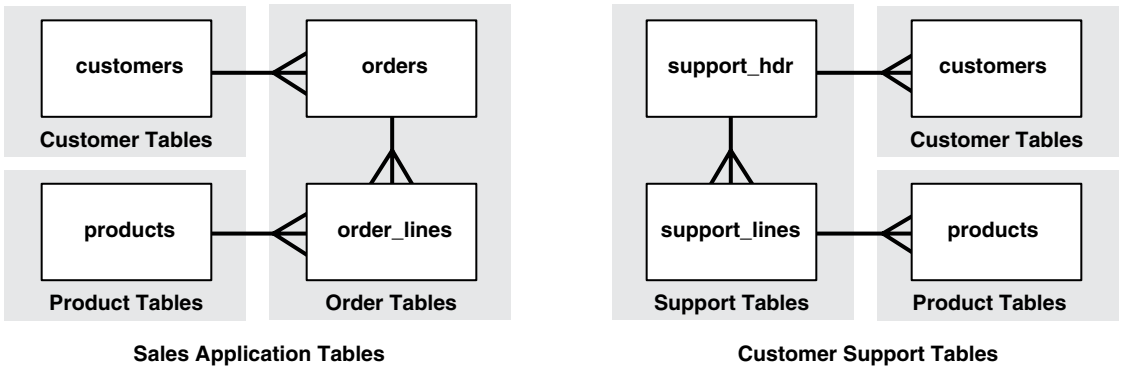
When designing your deployment templates, consider the different sets of users that need to access the target data. For example, both salespersons and technicians need customer information, but the technicians might not need sales information. You do not want users to instantiate deployment templates that contain extraneous data, because it requires extra storage space and incurs longer refresh time.

On the other hand, if you have users that require both sales and customer support information, then you do not want users to have to instantiate multiple deployment templates that share redundant data. Instantiating multiple templates might cause data consistency problems. Each deployment template uses a different refresh group, which means that data in the two deployment templates can 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 these 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 4–8 The Different Needs of Salespersons and Customer Support Technicians



Another design consideration is the usage of parameters. If many of the tables in Figure 4–8 use the `customer_id` field, then 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 materialized views that use subquery subsetting. One parameter allows a user to receive only the data for the customers that the user needs. Consider the following `CREATE MATERIALIZED VIEW` statements:

```
CREATE MATERIALIZED VIEW sales.orders AS
 SELECT * FROM sales.orders@orc1.example.com o
  -- conditions for customers
  WHERE EXISTS
  ( SELECT c_id FROM sales.customer@orc1.example.com c
    WHERE o.c_id = c.c_id
      AND EXISTS
    ( SELECT * FROM sales.assignment@orc1.example.com a
     WHERE a.c_id = c.c_id
     AND EXISTS
     ( SELECT * FROM sales.salesperson@orc1.example.com s
       WHERE s.s_id = :salesperson_id)));

CREATE MATERIALIZED VIEW sales.customer AS
SELECT c_id FROM sales.customer@orc1.example.com c
  -- conditions for customers
  WHERE EXISTS
   ( SELECT * FROM sales.assignment@orc1.example.com a
     WHERE a.c_id = c.c_id
     AND EXISTS
     ( SELECT * FROM sales.salesperson@orc1.example.com s
       WHERE s.s_id = :salesperson_id)));
```

Even though the two materialized views 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 materialized views easier for both the DBA and the user instantiating the deployment template.

Additional Design Considerations

Finally, consider what other objects need to be created at the remote materialized view site. Consider the following questions:

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

Local Control of Materialized View Creation

A deployment template is the most effective method of building and distributing a materialized view environment. Even if distribution is limited to only two or three sites, you still significantly reduce the amount of steps needed to build a materialized view environment by using deployment templates as opposed to individually creating the materialized view 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 materialized view environment, then when should you locally build the materialized view environment at the remote materialized view site? In most cases, you should build a materialized view environment using the Materialized View Group Wizard or locally at the materialized view site when local control must be maintained at the materialized view site.

One scenario where you might find local control of materialized view creation helpful is when it is desirable for the materialized view site to control what data it receives. For example, this is especially true of decision support sites (DSS), which are typically read-only materialized view sites. A DSS site might occasionally need to run complex queries and they do not want to slow the OLTP site, or bother the DBA at the OLTP site.

Local Materialized View Control

One of the major benefits of deployment templates is that control is maintained centrally by the DBA building the deployment template. In some cases, however, the materialized view site must retain some control.

Local control might be required if the materialized view site:

- Has an experienced DBA
- Is considered a trusted site
- Is a materialized view instead of a master site because of row subsetting requirements

Because materialized view groups are created with the Advanced Replication interface's Create Materialized View Group Wizard locally at the materialized view site by its DBA, or perhaps a systems analyst with SQL knowledge, control can also be maintained at the materialized view site.

Consider the following as a perfect example for maintaining local control. Because multimaster replication does not allow for row and column data subsetting, updatable materialized view sites are sometimes created primarily for their ability to subset data. These sites are typically secure, have experienced DBAs, and require the ability to maintain control locally to meet user and application requirements. Materialized view groups created with the Materialized View Group Wizard or with the replication management API allow for the localized control necessary to meet the requirements of the secure updatable materialized view sites.

Also, remember that when a materialized view environment is created with a deployment template, all objects in the materialized view environment are added to the same refresh group. While this might be fine for most installations, certain situations might require that the objects in a materialized view group are assigned to several different refresh groups.

5

Conflict Resolution Concepts and Architecture

Some replication environments must create conflict resolution methods to resolve possible data conflicts that can result from replicating data between multiple sites.

This chapter contains these topics:

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, then 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 replication 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 updatable at multiple sites at any time. If this is the case, then you must address the possibility of replication conflicts.

The next few sections introduce the following topics relating to replication conflicts:

- How to design a replication system with replication conflicts in mind
- How to determine the types of conflicts that are possible in your replication environment
- How you can avoid replication conflicts in designing your replication environment
- How Oracle can detect and resolve conflicts in designs where conflict avoidance is not possible

This section contains these topics:

- Conflict Detection
- Conflict Resolution
- Techniques for Avoiding Conflicts

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, you must investigate additional requirements 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 materialized views, 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 updatable 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.

Examples of Conflict Detection and Resolution

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 replication 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

You might encounter these types of data conflicts in a replicated database environment:

- Update Conflicts
- Uniqueness Conflicts
- Delete Conflicts

You will most likely encounter update conflicts in your replication environment, although you should always prepare to handle uniqueness and delete conflicts. Oracle 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

Ordering conflicts can occur in replication environments with three or more master sites. If propagation to master site X is blocked for any reason, then updates to replicated data can continue to be propagated among other master sites. When propagation resumes, these updates might be propagated to site X in a different order than they occurred on the other masters, and these updates might conflict. By default, the resulting conflicts are recorded in the error log and can be reexecuted after the transactions they depend upon are propagated and applied. See Table 5–1 on page 5-14 for 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 time stamp, minimum, maximum, priority group, additive).

The minimum, maximum, priority group, and additive conflict resolution methods guarantee data convergence with any number of master sites, as long as certain conditions exist. See the appropriate conflict resolution method in "Conflict Resolution Architecture" on page 5-14 for more information

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, then 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, then you can easily resolve the situation by reexecuting the transaction in error after the supporting data has been propagated to the remote site.

Conflict Detection

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 materialized view site pushes deferred transactions to its corresponding master site or master materialized view site, the receiving site performs conflict detection and resolution. A materialized view site refreshes its data by performing materialized view refreshes. The refresh mechanism ensures that, upon completion, the data at a materialized view is the same as the data at the corresponding master table or master materialized view, including the results of any conflict resolution. Therefore, it is not necessary for a materialized view site to perform work to detect or resolve replication conflicts.

How Oracle Detects Different Types of Conflicts

The receiving master site or master materialized view 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:

- If a column is updated and the column's old value equals its new value, then Oracle never detects a conflict for this column update.
- 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 "Send and Compare Old Values" on page 5-27.

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, Advanced Replication 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 uniquely identify rows in the table during data replication.

Caution: Do not permit applications to update the primary key or alternate key columns of a table. This precaution ensures that Oracle can identify rows and preserve the integrity of replicated data.

Conflict Resolution

After a conflict has been detected, 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, enable 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, then 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, then 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 time stamp conflict resolution method cannot resolve a conflict because the time stamps are identical, you might want to define a site priority conflict resolution method, which breaks the time stamp tie and resolves the data conflict.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* for information about modifying tables without replicating the modifications, which might be necessary when you manually resolve a conflict that could not be resolved automatically

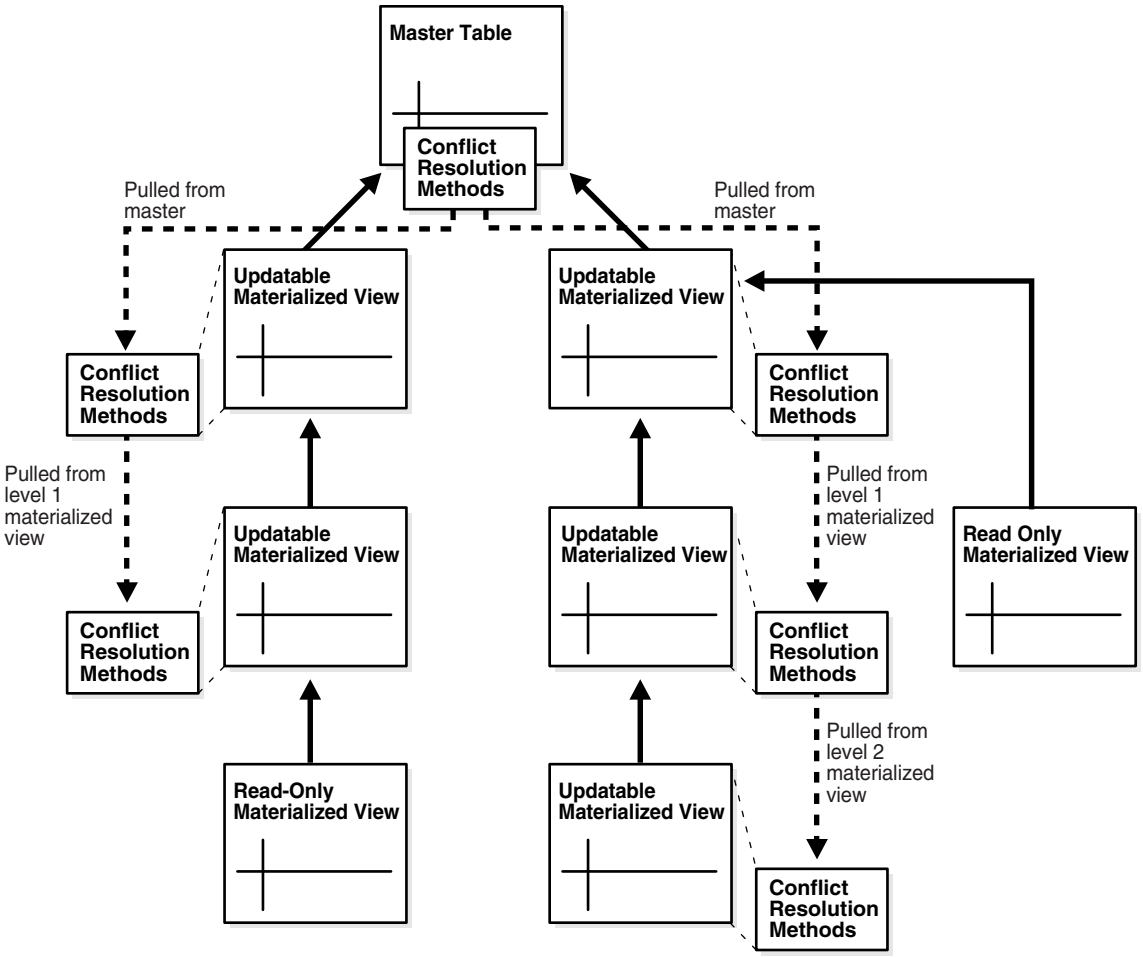
Multitier Materialized Views and Conflict Resolution

When you have a master table and an updatable materialized view based on that master table, a refresh of the materialized view pushes its changes to the master site, where the master site handles any conflicts resulting from the push with its configured conflict resolution methods. Then, the materialized view pulls changes at the master down when the materialized view completes the refresh. The refresh is always initiated at the materialized view site.

Similarly, the master materialized view of an updatable materialized view behaves in the same way as a master table. However, to handle conflicts resulting from a push from a materialized view, the master materialized view uses conflict resolution methods that it has pulled from its master. Here, the master can either be a master table at a master site or a master materialized view at another materialized view site. Conflict resolution methods cannot be configured directly at a materialized view site. Instead, the conflict resolution methods are pulled down from the immediate master automatically when you create an updatable materialized view and when you generate replication support for a materialized view. A read-only materialized view does not pull down conflict resolution methods from its master.

For example, suppose a level 3 materialized view pushes its changes to its level 2 master materialized view. This push might cause a conflict at the level 2 materialized view. To handle the conflict, the level 2 materialized view uses the conflict resolution methods that it previously pulled from its level 1 master materialized view. Similarly, the level 1 materialized view handles conflicts with the conflict resolution methods that it previously pulled from its master site. Figure 5–1 illustrates this configuration.

Figure 5–1 Conflict Resolution and Multitier Materialized Views



Notice that each updatable materialized view pulls-down conflict resolution methods from its master, even if the updatable materialized view does not have any materialized views based on it. Notice also that a read-only materialized view does not pull down conflict resolution methods from its master.

If you plan to change the conflict resolution methods for a master table in an environment with multitier materialized views, then complete the following general procedure:

1. If you are modifying either column groups or key columns and you are using minimum communication for any of the updatable materialized views based on the master table, then complete the following sub-steps:
   a. Refresh the materialized views that are the farthest removed from the master table you are altering. By refreshing, you push all the deferred transactions from each materialized view to its master. For example, if you have three levels of materialized views, then refresh the level 3 materialized views.
   b. Stop all data manipulation language (DML) changes at the materialized views you refreshed in Step a.
   c. Repeat Step a and Step b for each materialized view level until you complete these steps for the level 1 materialized views, which are based on a master table at a master site.
2. If necessary, then quiesce the master group.

3. Change the conflict resolution configuration at the master definition site.
4. Regenerate replication support for the affected objects at the master definition site using either the GENERATE_REPLICATION_SUPPORT procedure in the DBMS_REPCAT package or the Advanced Replication interface in Oracle Enterprise Manager.
5. If you quiesced the master group in Step 2, then resume replication activity for the master group.
6. Regenerate replication support for the materialized views with the smallest level number that have not yet regenerated replication support. The current conflict resolution methods are pulled down from the immediate master during regeneration. The first time you complete this step, it is for the level 1 materialized views, the second time for the level 2 materialized views, and so on. You regenerate replication support for a materialized view using either the GENERATE_MVIEW_SUPPORT procedure in the DBMS_REPCAT package or the Advanced Replication interface in Oracle Enterprise Manager.
7. If you completed the sub-steps in Step 1, then allow DML changes at the materialized views with the smallest level number that do not currently allow DML changes. The first time you complete this step, it is for the level 1 materialized views, the second time for the level 2 materialized views, and so on.
8. Repeat Step 6 and Step 7 for each level of materialized views until you complete these steps for the materialized views that are farthest removed from the master table. For example, if you have three levels of materialized views, then the last time you complete these steps it is for the level 3 materialized views.

This regeneration of replication support is not performed automatically. In an environment where different database administrators administer master sites and materialized view sites, the database administrator at the master sites must notify the database administrators at all of the affected materialized view sites of the changes in conflict resolution methods. Then, it is the responsibility of all of the database administrators to coordinate the previous procedure.

Column Subsetting of Updatable Materialized Views and Conflict Resolution

Column subsetting enables you to exclude columns in master tables from materialized views by identifying specific columns in the SELECT statement during materialized view creation. If only a subset of the columns in a column group are included in an updatable materialized view, then do not create a conflict resolution method on this column group, unless the conflict resolution method is either discard or site priority. If the conflict resolution method is site priority, then column subsetting should only be used in single master replication environments where the master site has a higher priority number than the materialized view site.

For any type of conflict resolution method other than discard and the variant of site priority described previously, the updatable materialized view sends information about changes for some of the columns in the column group but not others, causing Oracle to return an error when it tries to apply the conflict resolution method. Because discard and this variant of site priority do not depend on column information, you can use these methods along with column subsetting.

For example, suppose the employees master table has a column group that contains the employee_id, manager_id, department_id, and timestamp columns. You define a latest time stamp conflict resolution method on the column group at the master site. Then, you create an updatable materialized view called employees_mv based on the employees master table, but you use column subsetting to exclude the department_id column from the materialized view. When an update is made to the

employee_id or manager_id column at the materialized view, information about these changes are sent to the master site during a subsequent refresh. An error is returned at the master site because no information about the remaining column in the column group, department_id, is found when Oracle tries to apply the conflict resolution method.

Keep this in mind if you are using multitier materialized views. Because the conflict resolution methods are pulled down from the master site to a master materialized view, the same rules apply to master materialized view sites and updatable materialized views based on them.

> **See Also:** "Column Subsetting with Deployment Templates" on page 4-15

Nested Tables and Conflict Resolution

For each nested table column, Oracle creates a hidden column in the table called the NESTED_TABLE_ID column. Oracle also creates a separate table called a storage table to store the elements of the nested table. The storage table stores a row for each element of the nested table for each parent table row. The storage table also contains a NESTED_TABLE_ID column, which corresponds to the parent table's NESTED_TABLE_ID column and is used to identify the elements of the nested table for a particular parent row. Nested table columns require special consideration in Advanced Replication.The underlying storage tables require as much consideration for conflict resolution as the parent table, and there are additional issues to consider.

Replication handles data manipulation language (DML) statements on nested tables as separate DML statements on the parent table and storage table. When DML statements are executed on nested table columns, the actions performed by Oracle depend on the type of DML statement. The following table shows the actions performed by Oracle for each type of DML statement.

Insert Statements	Delete Statements	Update Statements
■ Inserts new rows into storage table (assuming nested table value is neither null nor empty). ■ Inserts new row into parent table, with the NESTED_TABLE_ID value referring to previously inserted storage rows.	■ Deletes any storage table rows associated with the parent table being deleted. ■ Deletes parent table row.	■ Inserts new rows into storage table (assuming the nested table column is set to a value that is neither null nor an empty table). ■ Updates the parent table row. ■ Deletes old storage table rows.

Example of Nested Table Conflicts The following example illustrates how DML statements on nested table columns can lead to conflicts that are difficult to resolve. Following the example is information about ways to minimize conflicts.

Suppose there is a university which stores information about its departments in a department table containing a nested table column that stores information about each department's courses:

```
CREATE TYPE Course AS OBJECT (
    course_no NUMBER(4),
    title VARCHAR2(35),
    credits NUMBER(1));
/
```

```
CREATE TYPE CourseList AS TABLE OF Course;
/

CREATE TABLE department (
      name VARCHAR2(20) primary key,
      director VARCHAR2(20),
      office VARCHAR2(20),
      courses CourseList)
      NESTED TABLE courses STORE AS courses_tab(
        (PRIMARY KEY(nested_table_id,course_no)));
```

The university has campuses across the United States and uses multimaster replication to support its different locations. Each location can update the `department` table, which is replicated. On `univ1.example.com`, one of the master sites, information is inserted about the `Psychology` department.

```
INSERT INTO department
      VALUES('Psychology', 'Irene Friedman', 'Fulton Hall 133',
      CourseList(Course(1000, 'General Psychology', 5),
      Course(2100, 'Experimental Psychology', 4),
      Course(2200, 'Psychological Tests', 3),
      Course(2250, 'Behavior Modification', 4),
      Course(3540, 'Groups and Organizations', 3),
      Course(3552, 'Human Factors in Business', 4),
      Course(4210, 'Theories of Learning', 4)));
```

Advanced Replication propagates the insert to all masters.

Then, change information arrives about the Psychology class offerings. That is, a class is added. The information is updated on `univ1.example.com`.

```
UPDATE department SET courses = CourseList(
      Course(1000, 'General Psychology', 5),
      Course(2100, 'Experimental Psychology', 4),
      Course(2200, 'Psychological Tests', 3),
      Course(2250, 'Behavior Modification', 4),
      Course(3540, 'Groups and Organizations', 3),
      Course(3552, 'Human Factors in Business', 4),
      Course(4210, 'Theories of Learning', 4),
      Course(4320, 'Cognitive Processes', 4))
      WHERE name = 'Psychology';
```

After `univ1.example.com` has committed the update, but before the change is propagated to other master sites, another master site, `univ2.example.com`, receives information that two more class have been added, both `4320` and `4410`.

```
UPDATE department SET courses = CourseList(
      Course(1000, 'General Psychology', 5),
      Course(2100, 'Experimental Psychology', 4),
      Course(2200, 'Psychological Tests', 3),
      Course(2250, 'Behavior Modification', 4),
      Course(3540, 'Groups and Organizations', 3),
      Course(3552, 'Human Factors in Business', 4),
      Course(4210, 'Theories of Learning', 4),
      Course(4320, 'Cognitive Processes', 4),
      Course(4410, 'Abnormal Psychology', 4))
      WHERE name = 'Psychology';
```

Both the update on `univ1.example.com` and the update on `univ2.example.com` are pushed.

There will be an update conflict on department table. Each user who made an update expects that it is the first update since the insert statement. But actually, the local update has taken place first, and therefore the NESTED_TABLE_ID has changed, because these are updates on the parent table. It is only updates on the nested table column (changing the storage table rows and NESTED_TABLE_ID) which are problematic. There is no problem updating other columns in the parent table.

Suppose this conflict is resolved by keeping the local table update. Delete conflict resolution would be required on the storage table to ignore the missing rows, which were already deleted by the local update. The new rows inserted into the storage table, due to the update at the remote site, now have no reference in the parent table. These new storage table rows must also be dealt with. Otherwise, they are orphaned. The storage table would grow with course rows which are not accessible from the department table.

Resolving conflicts by manipulating the storage table rows while updating the parent table is very difficult with two master sites in a multimaster replication environment and becomes nearly impossible as the number of master sites increases. If this type of update is necessary, then it might be best to not define any conflict resolution methods on the nested table and resolve conflicts manually. Incorrect conflict resolution could lead to divergence. That is, tables on different masters might no longer match.

Recommendations for Avoiding Problematic Updates The following recommendations enable you to avoid the problematic updates described in the preceding section:

- Use a foreign key constraint, initially deferred, on the nested table. This constraint prevents dangling rows in the storage table. The following is an example of such a foreign key constraint:

```
ALTER TABLE courses_tab add CONSTRAINT courses_fk
  FOREIGN KEY(NESTED_TABLE_ID) REFERENCES department(courses)
  INITIALLY DEFERRED;
```

- Ensure that all inserts on the parent table insert an empty nested table. Do not use a null nested table value. This practice helps to create a reusable NESTED_TABLE_ID. The following is an example of an insert that included an empty nested table:

```
INSERT INTO department (name, director, office, courses)
   VALUES('Psychology', 'Irene Friedman', 'Fulton Hall 133', CourseList());
```

- Ensure that all inserts, deletes, and updates are performed directly on the nested table rather than through DML on the parent table. This practice helps to reuse the present NESTED_TABLE_ID value.

  The following is an example of deleting rows directly from a nested table:

```
DELETE FROM TABLE
   (SELECT courses FROM department WHERE name = 'Psychology');
```

  Consider an example where the following rows are inserted directly into the nested table on univ1.example.com:

```
INSERT INTO TABLE
   (SELECT courses FROM department WHERE name = 'Psychology')
   VALUES (Course(5000, 'Social Psychology', 5));

INSERT INTO TABLE
   (SELECT courses FROM department WHERE name = 'Psychology')
   VALUES (Course(5100, 'Psychology of Personality', 4));
```

Then, the following rows are inserted directly into the nested table on `univ2.example.com` before the preceding inserts on univ1.example.com are pushed:

```
INSERT INTO TABLE
   (SELECT courses FROM department WHERE name = 'Psychology')
   VALUES (Course(5000, 'Social Psychology', 5));

INSERT INTO TABLE
   (SELECT courses FROM department WHERE name = 'Psychology')
   VALUES (Course(5100, 'Psychology of Personality', 4));

INSERT INTO TABLE
   (SELECT courses FROM department WHERE name = 'Psychology')
   VALUES (Course(5500, 'Cognitive Neuroscience', 5));
```

Here, primary key conflicts will occur on the inserted rows in the storage table for courses 5000 and 5100), but a conflict resolution on the storage table which allows the inserts from one site to fail should provide the proper results. However, these inserts do not result in the more complicated problem involving multiple tables described in "Example of Nested Table Conflicts" on page 5-8, but the `NESTED_TABLE_ID` value is not lost, because this value has not changed.

- Consider using a trigger on the parent table that prevents inserts and updates that include manipulation of the nested table column. The following is an example of such a trigger:

```
CREATE OR REPLACE TRIGGER depart_trig
  AFTER INSERT OR UPDATE ON department
  FOR EACH ROW
  DECLARE
  new_ntid raw(100);
  old_ntid raw(100);
  BEGIN
-- obtain the nested table ids
  SELECT sys_op_tosetid(:new.courses) INTO new_ntid from dual;
  SELECT sys_op_tosetid(:old.courses) INTO old_ntid from dual;
  IF INSERTING THEN
-- raise error on insert of a null nested table column
    IF :new.courses IS NULL THEN
      raise_application_error(-20011, 'inserting null nested table ref');
    END IF;
-- raise error if new rows are inserted in the storage table
-- this is not strictly necessary, but it does enforce DML access
-- semantics of separate DMLS on parent table and storage table
    IF :new.courses.count != 0 THEN
      raise_application_error(-20012,
      'inserting rows into storage table while inserting parent table row');
    END IF;
  ELSE
-- raise error if update has caused the NESTED_TABLE_ID to change
    IF new_ntid != old_ntid THEN
      raise_application_error(-20013,
        'updating storage table reference while updating parent table row');
    END IF;
  END IF;
END;
/
```

These recommendations continue to apply with multilevel nesting, where the storage table row becomes a parent to another storage table's rows. All of these recommendations are good strategies at each level of nesting.

Techniques for Avoiding Conflicts

Although 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.

Use Column Groups

Column groups can help you avoid conflicts even if you do not apply any conflict resolution methods to the column groups. When your replicated table contains multiple column groups, each group is viewed independently when analyzing updates for conflicts.

For example, consider a replicated table with column group `a_cg` and column group `b_cg`. Column group `a_cg` contains the following columns: `a1`, `a2`, and `a3`. Column group `b_cg` contains the following columns: `b1`, `b2`, and `b3`.

The following updates occur at replication sites `sf.example.com` and `la.example.com`:

- User `wsmith` updates column `a1` in a row at `sf.example.com`.
- At exactly the same time, user `mroth` updates column `b2` in the same row at `la.example.com`.

In this case, no conflicts result because Oracle analyzes the updates separately in column groups `a_cg` and `b_cg`. If, however, column groups `a_cg` and `b_cg` did not exist, then all of the columns in the table would be in the same column group, and a conflict would have resulted. Also, with the column groups in place, if user `mroth` had updated column `a3` instead of column `b2`, then a conflict would have resulted, because both `a1` and `a3` are in the `a_cg` column group.

> **See Also:** "Column Groups" on page 5-15 for more information about column groups

Use 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 row and column subsetting 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:** *Oracle Database Advanced Replication Management API Reference* 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, you can select a globally unique value using the `SYS_GUID` function. Using the selected value as the primary key (or unique) value globally avoids uniqueness conflicts.

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

> **See Also:** "Alternatives to Replicating Sequences" on page 2-21 for more information about sequences and *Oracle Database SQL Language Reference* for more information about the `SYS_GUID` function

Avoiding Delete Conflicts Always avoid delete conflicts 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 instructions for creating conflict avoidance methods for delete conflicts in the *Oracle Database Advanced 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, then

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, avoid or automatically resolve ordering conflicts. For example, 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

Very few architectural mechanisms and processes are visible when implementing conflict resolution into your replication environment. This section describes the few supporting mechanisms involved in conflict resolution and describes different aspects of Oracle's prebuilt conflict resolution methods.

This section contains these topics:

Support Mechanisms

The most important mechanism involved in Oracle conflict resolution is the column group because it is the basis for all update conflict detection and resolution. 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.

Column groups have the following characteristics:

- A column can belong only to one column group.
- A column group can consist of one or more columns of a table.
- Conflict resolution is applicable only to columns in a column group.

> **See Also:** "Use Column Groups" on page 5-12 for information about using column groups to avoid conflicts

Ensuring Data Integrity with Multiple Column Groups Having column groups enables 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 time stamp 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, then 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, then the sites could converge on a postal code that does not 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, ensure that to include a column in a column group when conflict resolution is necessary for the column. Oracle detects conflicts that involve columns in the shadow column group but does not attempt to apply any conflict resolution methods to resolve these conflicts.

Column Objects and Column Groups An Oracle object based on a user-defined type that occupies a single column in a table is a column object. A column object cannot span column groups. That is, given a column group and a column object, either the column object and all of its attributes must be within the column group, or the column object and all of its attributes must be excluded from a column group.

Oracle's prebuilt conflict resolution methods cannot resolve conflicts based on undefined column object attribute values. If a column object is `NULL`, then its attributes are undefined.

Object Tables and Column Groups An object table is a special kind of table in which each row represents an object based on a user-defined type. You can specify column groups that include a subset of the columns in an object table.

Nested Tables and Column Groups A nested table's storage table is treated as an independent table in conflict resolution. Therefore, you can create a column group based on a subset of the columns in a storage table.

Error Queue

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

> **See Also:** "Error Queue" on page 2-22 for more information about the error queue

Common Update Conflict Resolution Methods

Although Oracle provides eight prebuilt update conflict resolution methods, the latest time stamp and the overwrite conflict resolution methods are the most commonly implemented resolution methods.

These methods are the most common because they are easy to use and, in the proper environments, can guarantee data convergence. The latest time stamp 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	Convergence with Multiple Master Sites
Latest time stamp	YES (with backup method)
Overwrite	NO

> **Note:** All of Oracle's prebuilt conflict resolution methods provide convergence in an environment with a single master site that has one or more materialized view sites.

Latest Timestamp

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

The following example demonstrates an appropriate application of the latest time stamp 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 time stamp method, Oracle selects the most recent update, and applies the address with the correct postal code.

Target Environments The latest time stamp 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 materialized views.

Support Mechanisms To use the time stamp 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 time stamp column with the local `SYSDATE`. For a change applied from another site, the time stamp value should be set to the time stamp value from the originating site.

Note: When you use a time stamp conflict resolution method, you should designate a backup method, such as site priority, to be called if two sites have the same time stamp.

Timestamp Configuration Issues When you use time stamp resolution, you must carefully consider how time is measured on the different sites managing replicated data. For example, if a replication environment crosses time zones, then applications that use the system should convert all time stamps 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, then time stamp comparisons might not be accurate enough to satisfy application requirements.

You can maintain time stamp columns if you use the `EARLIEST` or `LATEST` time stamp update conflict resolution methods in the following ways:

- Each application can include logic to synchronize time stamps.
- You can create a trigger for a replicated table to synchronize time stamps automatically for all applications.

A clock counts seconds as an increasing value. Assuming that you have properly designed your time-stamping mechanism and established a backup method in case two sites have the same time stamp, the latest time stamp method (like the maximum value method) guarantees convergence. The earliest time stamp method, however, *cannot* guarantee convergence for more than one master site.

Implement Latest Timestamp See the Advanced Replication interface's online Help to learn how to define a latest time stamp conflict resolution method with the Advanced Replication interface in Oracle Enterprise Manager.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* to learn how to define this type of conflict resolution method with the replication management API

Overwrite

The **overwrite** method replaces the current value at the destination site with the new value 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 materialized view 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 materialized view sites, then you might select the overwrite method. The overwrite method is also useful if:

- Your primary concern is data convergence.
- You have a single master site.
- No particular business rule exists 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` data dictionary view and leaving the resolution to your local database administrator.

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

If a conflict is detected, then the value originating from the materialized view site is used, which means that priority is given to the most recently refreshed materialized views.

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

Implement Overwrite See the Advanced Replication interface's online Help to learn how to define an overwrite conflict resolution method with the Advanced Replication interface in Oracle Enterprise Manager.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* to learn how to define this type of conflict resolution method with the replication management API

Additional Update Conflicts Resolution Methods

If the latest time stamp or the overwrite conflict resolution methods do not meet your needs to resolve data conflicts that are encountered in your replication environment, then Oracle offers six additional prebuilt update conflict resolution methods.

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

Resolution Methods	Convergence with Multiple Master Sites
Additive	YES
Average	NO
Discard	NO
Earliest time stamp	NO
Maximum	YES (column values must always increase)
Minimum	YES (column values must always decrease)
Priority group	YES (with ordered update values)
Site priority	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, then 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 according to this formula:

```
current value = current value + (new value - old value)
```

The additive conflict resolution method provides convergence for any number of master sites and materialized view sites.

Target Environments The additive conflict resolution method is designed to conserve data rather than choose the most appropriate data. This method might be useful in a financial environment where deposits and withdrawals happen so frequently that conflicts can 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 withdrawals).

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

Implement Additive See the Advanced Replication interface's online Help to learn how to define an additive conflict resolution method with the Advanced Replication interface in Oracle Enterprise Manager.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* 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 replication 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 updatable materialized views.

The average method might be useful for scientific applications that would rather average two values than choose 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 Advanced Replication interface's online Help to learn how to define an average conflict resolution method with the Advanced Replication interface in Oracle Enterprise Manager.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* 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. The discard method ignores the new value from the originating site and retains the value at the destination site. This method is designed to be used by a single master site and multiple materialized view sites, or with some form of a user-defined notification facility.

For example, if you have a single master site and multiple materialized view sites based on it, and you expect the materialized view sites to be used primarily for queries with all updates being performed at the master site, then you might select the discard method. The discard methods is 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 materialized view sites. If a conflict is detected, then the value originating from the materialized view site is ignored, which means that priority is given to materialized views that refresh first.

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

Implement Discard See the Advanced Replication interface's online Help to learn how to define a discard conflict resolution method with the Advanced Replication interface in Oracle Enterprise Manager.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* to learn how to define this type of conflict resolution method with the replication management API

Earliest Timestamp

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

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

Support Mechanisms To use the time stamp 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 time stamp column with the local `SYSDATE`. For a change applied from another site, the time stamp value should be set to the time stamp value from the originating site. Be sure to review "Timestamp Configuration Issues" on page 5-17.

> **Note:** When you use a time stamp conflict resolution method, you should designate a backup method, such as site priority, to be called if two sites have the same time stamp.

Implement Earliest Timestamp See the Advanced Replication interface's online Help to learn how to define an earliest time stamp conflict resolution method with the Advanced Replication interface in Oracle Enterprise Manager.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* to learn how to define this type of conflict resolution method with the replication management API

Maximum

When Advanced Replication 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 maximum value conflict resolution method.

If the new value of the designated column is *greater than* the current value, then the column group values from the originating site are applied at the destination site, assuming that all other errors were successfully resolved for the row. If the new value of the designated column is less than the current value, then 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), then 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 data types 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, then this method guarantees data convergence with any number of master sites and materialized view sites.

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

Implement Maximum See the Advanced Replication interface's online Help to learn how to define a maximum conflict resolution method with the Advanced Replication interface in Oracle Enterprise Manager.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* to learn how to define this type of conflict resolution method with the replication management API

Minimum

When Advanced Replication 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, then the column group values from the originating site are applied at the destination site, assuming that all other errors were successfully resolved for the row. If the new value of the designated column is greater than the current value, then 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), then 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 data types 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-decreasing 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, then this method guarantees data convergence with any number of master sites and materialized view sites.

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

Implement Minimum See the Advanced Replication interface's online Help to learn how to define a minimum conflict resolution method with the Advanced Replication interface in Oracle Enterprise Manager.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* book to learn how to define minimum and maximum methods conflict resolution methods with the replication management API

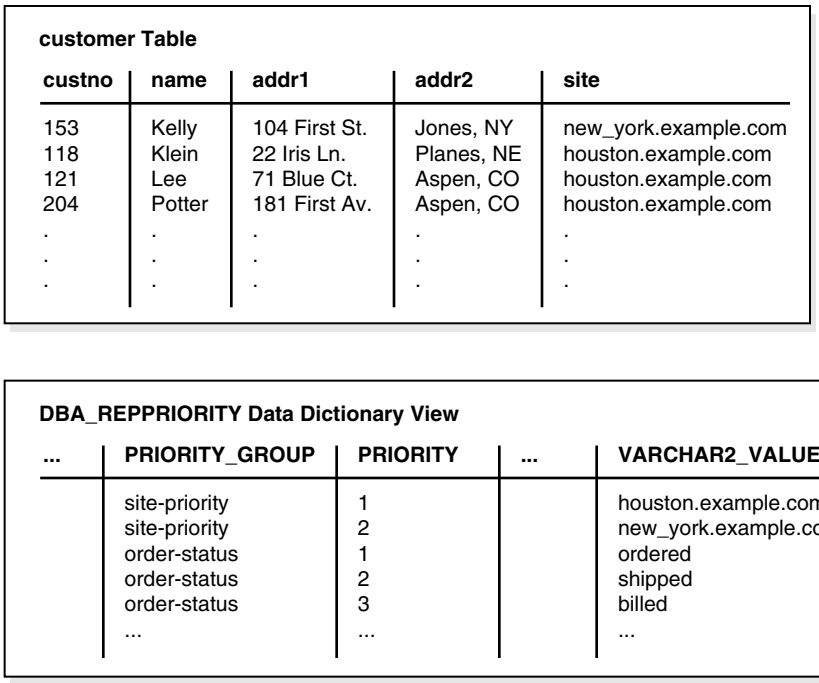
Priority Groups

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

You can guarantee convergence with more than one master site 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.

As shown in Figure 5–2, 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 5–2 Using Priority Groups



customer Table

custno	name	addr1	addr2	site
153	Kelly	104 First St.	Jones, NY	new_york.example.com
118	Klein	22 Iris Ln.	Planes, NE	houston.example.com
121	Lee	71 Blue Ct.	Aspen, CO	houston.example.com
204	Potter	181 First Av.	Aspen, CO	houston.example.com
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

DBA_REPPRIORITY Data Dictionary View

...	PRIORITY_GROUP	PRIORITY	...	VARCHAR2_VALUE
	site-priority	1		houston.example.com
	site-priority	2		new_york.example.com
	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 5–2, 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 the Advanced Replication interface in Oracle Enterprise 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 replication 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 Advanced Replication interface's online Help to learn how to define a priority group conflict resolution method with the Advanced Replication interface in Enterprise Manager.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* 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 5–2 on page 5-24, the new_york.example.com site (priority value = 2) is corporate headquarters, while the houston.example.com site (priority value = 1) is an updatable materialized view 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 alone, convergence with more than one master site is not guaranteed, but site priority can be a good backup method in a multimaster environment, especially for breaking latest time stamp ties.

Similar to priority groups, you must complete several preparatory steps before using the Advanced Replication interface in Oracle Enterprise Manager to select site priority conflict resolution for a column group.

Target Environments As with 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 materialized views), 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 in a multimaster environment.

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 *Oracle Database Advanced 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 Advanced Replication interface's online Help to learn how to define a site priority conflict resolution method with the Advanced Replication interface in Oracle Enterprise Manager.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* 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 replication 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.

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

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 is 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.example.com` 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 is 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, then 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 master sites 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 "Avoiding Delete Conflicts" on page 5-13, you should design your database and front-end application to avoid delete conflicts. You can achieve this goal by marking rows for deletion and at regular intervals, using procedural replication to purge such marked rows.

See Also:

- "Avoiding Delete Conflicts" on page 5-13 to learn how to avoid encountering delete conflicts
- *Oracle Database Advanced Replication Management API Reference* to learn how to build conflict avoidance into your replication environment

Send and Compare Old Values

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.

You can 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 time stamp conflict detection and resolution method does not require old

values for nonkey and non time stamp columns in a column group if the columns are guaranteed to be updated whenever the timestamp column is updated.

Suggestion: Further minimizing propagation of old values is particularly valuable if you are replicating LOB data types 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_UTILITY.VARCHAR2s | DBMS_UTILITY.LNAME_ARRAY,}
   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_UTILITY.VARCHAR2s | DBMS_UTILITY.LNAME_ARRAY,}
   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 enables you to decide whether or not to transmit old values for nonkey columns when rows are deleted or when nonkey columns are updated or both. 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.

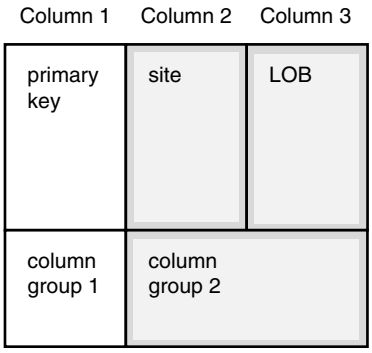
The specified behavior for old column values is exposed in two columns in the DBA_REPCOLUMN data dictionary view: COMPARE_OLD_ON_DELETE (Y or N) and COMPARE_OLD_ON_UPDATE (Y or N).

Send and Compare Example

The following example shows how you can further reduce data propagation by using these procedures. Consider a table called rsmith.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).

Figure 5–3 Column Groups and Data Propagation



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 rsmith.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:

```
BEGIN
  DBMS_REPCAT.SEND_OLD_VALUES(
     sname           =>  'rsmith',
     oname           =>  'reports',
     column_list     =>  'c3',
     operation       =>  'UPDATE',
     send            =>  FALSE );
END;
/

BEGIN
  DBMS_REPCAT.COMPARE_OLD_VALUES(
     sname           =>  'rsmith',
     oname           =>  'reports',
     column_list     =>  'c3',
     operation       =>  'UPDATE',
     compare         =>  FALSE);
```

```
END;
/
```

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

Send and Compare When Using Column Objects

You can specify leaf attributes of a column object when you send and compare old values if the attributes are not replication key columns. For example, suppose you create the following cust_address_typ object type.

```
CREATE TYPE cust_address_typ AS OBJECT
     (street_address     VARCHAR2(40),
      postal_code        VARCHAR2(10),
      city               VARCHAR2(30),
      state_province     VARCHAR2(10),
      country_id         CHAR(2));
/
```

You create the customers table using this type as a column object:

```
CREATE TABLE customers
     (customer_id        NUMBER(6),
      cust_first_name    VARCHAR2(20),
      cust_last_name     VARCHAR2(20),
      cust_address       cust_address_typ,
      phone_numbers      phone_list_typ);
```

If you want to send and compare old values for the street_address attribute of the cust_address_typ type in the customers table, then you run the following procedures to specify that you do want to send or compare the attribute value:

```
BEGIN
  DBMS_REPCAT.SEND_OLD_VALUES(
     sname           =>  'oe',
     oname           =>  'customers',
     column_list     =>  'cust_address.street_address', -- object attribute
     operation       =>  'UPDATE',
     send            =>   TRUE );
END;
/

BEGIN
  DBMS_REPCAT.COMPARE_OLD_VALUES(
     sname           =>  'oe',
     oname           =>  'customers',
     column_list     =>  'cust_address.street_address', -- object attribute
     operation       =>  'UPDATE',
     compare         =>   TRUE);
END;
```

```
/
```

> **Note:** If you have multiple levels of object attributes in one column object, then you can only specify the final (or leaf) attribute for the `column_list` parameter. You cannot specify middle attributes.

You can also specify that you want to send and compare an entire column object. For example, the following procedures specify the entire `cust_address` column object:

```
BEGIN
  DBMS_REPCAT.SEND_OLD_VALUES(
     sname           =>  'oe',
     oname           =>  'customers',
     column_list     =>  'cust_address', -- entire column object
     operation       =>  'UPDATE',
     send            =>   TRUE );
END;
/

BEGIN
  DBMS_REPCAT.COMPARE_OLD_VALUES(
     sname           =>  'oe',
     oname           =>  'customers',
     column_list     =>  'cust_address', -- entire column object
     operation       =>  'UPDATE',
     compare         =>   TRUE);
END;
/
```

> **See Also:** The *Oracle Database Advanced Replication Management API Reference* for details about the `DBMS_REPCAT.SEND_OLD_VALUES` procedure and the `DBMS_REPCAT.COMPARE_OLD_VALUES` procedure

6

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 contains these topics:

Considerations for Replicated Tables

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

Primary Keys and Replicated Tables

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 the tables that you plan to use in your replication environment do not have a primary key or a set of unique columns, then

alter these tables accordingly. In addition, if you plan to create any primary key materialized views based on a master table or master materialized view, then that master must have a primary key.

Foreign Keys and Replicated Tables

When replicating tables with foreign key referential constraints, Oracle recommends that you always index foreign key columns and replicate these indexes, unless no updates and deletes are allowed in the parent table. Indexes are not replicated automatically. To replicate an index, add it to the master group containing its table using either the Advanced Replication interface in Oracle Enterprise Manager or the `CREATE_MASTER_REPOBJECT` procedure in the `DBMS_REPCAT` package.

Data Type Considerations for Replicated Tables

Advanced Replication supports the replication of tables and materialized views with columns that use the following data types:

- `VARCHAR2`
- `NVARCHAR2`
- `NUMBER`
- `DATE`
- `TIMESTAMP`
- `TIMESTAMP WITH TIME ZONE`
- `TIMESTAMP LOCAL TIME ZONE`
- `INTERVAL YEAR TO MONTH`
- `INTERVAL DAY TO SECOND`
- `RAW`
- `ROWID`
- `CHAR`
- `NCHAR`
- `CLOB` with `BASICFILE` storage
- `NCLOB` with `BASICFILE` storage
- `BLOB` with `BASICFILE` storage
- `XMLType` stored as `CLOB`
- User-defined types that do not use type inheritance or type evolution
- Oracle-supplied types that do not use type inheritance or type evolution

The deferred and synchronous remote procedure call mechanism used for multimaster replication propagates only the piece-wise changes to the supported LOB data types when piece-wise updates and appends are applied to these LOB columns. Also, you cannot reference LOB columns in a `WHERE` clause of a materialized view's defining query.

You can replicate tables and materialized views that use user-defined types, including column objects, object tables, `REF`s, varrays, and nested tables.

Advanced Replication does not support the replication of tables and materialized views with columns that use the following data types:

- FLOAT
- BINARY_FLOAT
- BINARY_DOUBLE
- LONG
- LONG RAW
- CLOB with SECUREFILE storage
- NCLOB with SECUREFILE storage
- BLOB with SECUREFILE storage
- BFILE
- XMLType stored object relationally or as binary XML
- Expression type
- User-defined types that use type inheritance or type evolution
- Oracle-supplied types that use type inheritance or type evolution

For Advanced Replication support, you should convert LONG data types to LOBs with BASICFILE storage.

Oracle also does not support the replication of UROWID columns in master tables or updatable materialized views. However, UROWID columns are allowed in read-only materialized views.

See Also:

- *Oracle Database Advanced Replication Management API Reference* for information about converting a LONG column into a LOB column in a replicated table
- *Oracle Database SQL Language Reference* for information about data types

Unsupported Table Types

Advanced Replication does not support the replication of the following types of tables and does not support materialized views based on these types of tables:

- Tables that have been compressed with the table compression feature
- Tables with columns that have been encrypted using transparent data encryption
- Tables with virtual columns
- Temporary tables
- Tables in a flashback data archive
- Tables stored using Oracle Automatic Storage Management (ASM)

Row-Level Dependency Tracking

When you create a table, you can specify the following options for tracking system change numbers (SCNs):

- NOROWDEPENDENCIES, the default, specifies that the SCN is tracked at the data block level.
- ROWDEPENDENCIES specifies that the SCN is tracked for each row in the table.

Using the ROWDEPENDENCIES option improves performance and scalability when using parallel propagation, but this option also requires six bytes of additional storage space for each row.

The following SQL statement creates a table with the ROWDEPENDENCIES option:

```
CREATE TABLE order_items
  (order_id          NUMBER(12),
  line_item_id       NUMBER(3)  NOT NULL,
  product_id         NUMBER(6)  NOT NULL,
  unit_price         NUMBER(8,2),
  quantity           NUMBER(8)
  ) ROWDEPENDENCIES;
```

Oracle tracks the SCN for each row in this order_items table. You can also use the ROWDEPENDENCIES option in a CREATE CLUSTER statement if your tables are part of a cluster.

> **See Also:** "Data Propagation Dependency Maintenance" on page 2-38 for more information about the ROWDEPENDENCIES option

Initialization Parameters

Table 6–1 lists initialization parameters that are important for the operation, reliability, and performance of a replication environment. This table specifies whether each parameter is modifiable. A modifiable initialization parameter can be modified using the ALTER SYSTEM statement while an instance is running. Some of the modifiable parameters can also be modified for a single session using the ALTER SESSION statement.

Table 6–1 Initialization Parameters Important for Advanced Replication

Parameter	Values	Description	Recommendation
GLOBAL_NAMES	**Default:** false **Range:** true or false **Modifiable?:** Yes	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 at each database that is participating in your replication environment, including both master sites and materialized view sites.
JOB_QUEUE_PROCESSES	**Default:** 1000 **Range:** 0 to 1000 **Modifiable?:** Yes	Specifies the number of J*n* job slaves for each instance (J000 . . . J999). Job slaves handle requests created by DBMS_JOB. When JOB_QUEUE_PROCESSES is set to 0 at a site, you must apply administrative requests manually for all groups at the site, and you must manually push and purge the deferred transaction queue.	This parameter should either be unset or set to at least 1. If it is set, then it should be set to the same value as the maximum number of jobs that can run simultaneously plus one.
MEMORY_MAX_TARGET	**Default:** 0 **Range:** 0 to the physical memory size available to the Oracle Database **Modifiable?:** No	Specifies the maximum system-wide usable memory for the Oracle database.	If the MEMORY_TARGET parameter is set to a nonzero value, then set this parameter to a large nonzero value if you need to specify the maximum memory usage of the Oracle database.
MEMORY_TARGET	**Default:** 0 **Range:** 152 MB to MEMORY_MAX_TARGET setting **Modifiable?:** Yes	Specifies the system-wide usable memory for the Oracle database.	Oracle recommends enabling the autotuning of the memory usage of the Oracle database by setting MEMORY_TARGET to a large nonzero value (if this parameter is supported on your platform).
OPEN_LINKS	**Default:** 4 **Range:** 0 to 255 **Modifiable?:** No	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.	If you are using synchronous replication, OPEN_LINKS must be set to at least the number of master sites. For example, an environment with five master sites requires that OPEN_LINKS be set to at least 5.

Table 6–1 (Cont.) Initialization Parameters Important for Advanced Replication

Parameter	Values	Description	Recommendation
PARALLEL_MAX_SERVERS	**Default:** Derived automatically **Range:** 0 to 3600 **Modifiable?:** Yes	Specifies the maximum number of parallel execution processes and parallel recovery processes for an instance. As demand increases, Oracle increases the number of processes from the number created at instance startup up to this value.	If you use parallel propagation, then ensure that 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 **Modifiable?:** Yes	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, then ensure that you have at least one process for each stream.
PROCESSES	**Default:** 100 **Range:** 6 to operating system dependent limit **Modifiable?:** No	Specifies the maximum number of operating system user processes that can simultaneously connect to Oracle.	Ensure that the value of this parameter allows for all background processes, such as locks, job slaves, and parallel execution processes.
REPLICATION_DEPENDENCY_TRACKING	**Default:** true **Range:** true or false **Modifiable?:** No	Enables or disables dependency tracking for read/write operations to the database. Dependency tracking is essential for propagating changes in a replication 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.	Typically, specify true. Do not specify false unless you are sure that your application will perform no read/write operations to the replicated tables.

Table 6–1 (Cont.) Initialization Parameters Important for Advanced Replication

Parameter	Values	Description	Recommendation
SGA_TARGET	**Default:** 0 **Range:** 64 MB to operating system dependent limit **Modifiable?:** Yes	Specifies the total size of all SGA components.	If MEMORY_MAX_TARGET and MEMORY_TARGET are set to 0 (zero), then Oracle recommends enabling the autotuning of SGA memory by setting SGA_TARGET to a large nonzero value.
SHARED_POOL_SIZE	**Default:** 0 If SGA_TARGET is set to a nonzero value: If the parameter is not specified, then the default is 0 (internally determined by the Oracle database). If the parameter is specified, then the user-specified value indicates a minimum value for the memory pool. If SGA_TARGET is not set (32-bit platforms): 64 MB, rounded up to the nearest granule size If SGA_TARGET is not set (64-bit platforms): 128 MB, rounded up to the nearest granule size **Range:** The granule size to operating system-dependent limit **Modifiable?:** Yes	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 multiuser systems. Smaller values use less memory.	Typically, the shared pool should be larger for an Oracle database in a replication environment than in a nonreplication environment. You can monitor utilization of the shared pool by querying the view V$SGASTAT.

See Also:

- *Oracle Database Reference* for more information about these initialization parameters
- *Oracle Database Administrator's Guide* for more information about the MEMORY_TARGET, MEMORY_MAX_TARGET, and SGA_TARGET parameters

Master Sites and Materialized View 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 materialized view sites. Consider the characteristics and advantages of both types of replication sites when you are deciding whether a particular site in your replication environment should be a master site or a materialized view site. One replication environment can support both master sites and materialized view sites.

Table 6–2 Characteristics of Master Sites and Materialized View Sites

Master Sites	Materialized View Sites
Typically communicate with a small number of other master sites, and might communicate with a large number of materialized view sites	Communicate with one master site or one master materialized view 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 site's or master materialized view site's data
Typically communicate continuously with short intervals between data propagation	Communicate periodically with longer intervals between bulk data transfers

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, use either the Advanced Replication interface's Configure Master Sites for Replication Wizard or the replication management API.

See Also:

- The Advanced Replication interface's online Help for instructions on using the Configure Master Sites for Replication Wizard to set up master sites in Oracle Enterprise Manager
- The *Oracle Database Advanced Replication Management API Reference* for instructions on using the replication management API to set up a master site
- "Designing for Survivability" on page 6-24 for information about designing your replication environment for failover protection

Advantages of Materialized View Sites

Materialized view sites have the following advantages:

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

To set up a materialized view site, you can use either the Advanced Replication interface's Configure Master and Materialized View Sites for Replication Wizard or the replication management API.

See Also:

- The Advanced Replication interface's online Help for instructions on using the Configure Materialized View Sites for Replication Wizard to set up materialized view sites in Oracle Enterprise Manager
- *Oracle Database Advanced Replication Management API Reference* for instructions on using the replication management API to set up a materialized view site

Preparing for Materialized Views

Most problems encountered with materialized view replication result from not preparing the environment properly. There are four essential tasks that you must perform before you begin creating your materialized view environment:

- Create the necessary schema.
- Create the necessary database links.
- Assign the appropriate privileges.
- Allocate sufficient job processes.

The Advanced Replication interface's Configure Master and Materialized View Sites for Replication Wizard automatically performs these tasks. The following discussion is provided to help you understand the replication environment and to help those who use the replication management API. After running Setup Wizard, create the necessary materialized view logs. See the Advanced Replication interface's online Help in Oracle Enterprise Manager for instructions on using the interface to set up your materialized view site.

See Also: "Creating a Materialized View Log" on page 6-12

If you are not able to use the Advanced Replication interface, then review the "Set Up Materialized View Sites" section in Chapter 2 of the *Oracle Database Advanced Replication Management API Reference* for detailed instructions on setting up your materialized view site using the replication management API.

The following sections describe what the Advanced Replication interface's Configure Master and Materialized View Sites for Replication Wizard or the script in the *Oracle Database Advanced Replication Management API Reference* does to set up your materialized view site.

Create Materialized View Site Users

Each materialized view site needs several users to perform the administrative and refreshing activities at the materialized view site. You must create and grant the necessary privileges to the materialized view 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 materialized view site users. Usually, a proxy materialized view administrator and a proxy refresher are created.

Create Schemas at Materialized View Site

A schema containing a materialized view 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 materialized views. After 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 `ny.example.com` database, then create a corresponding `sales` schema in the materialized view database `sf.example.com`.

> **See Also:** If you are reviewing the steps in *Oracle Database Advanced Replication Management API Reference*, then the necessary schemas are created as part of the script described in the instructions for creating a materialized view group

Create Database Links

The defining query of a materialized view can use one or more database links to reference remote table data. Before creating materialized views, 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 materialized view.

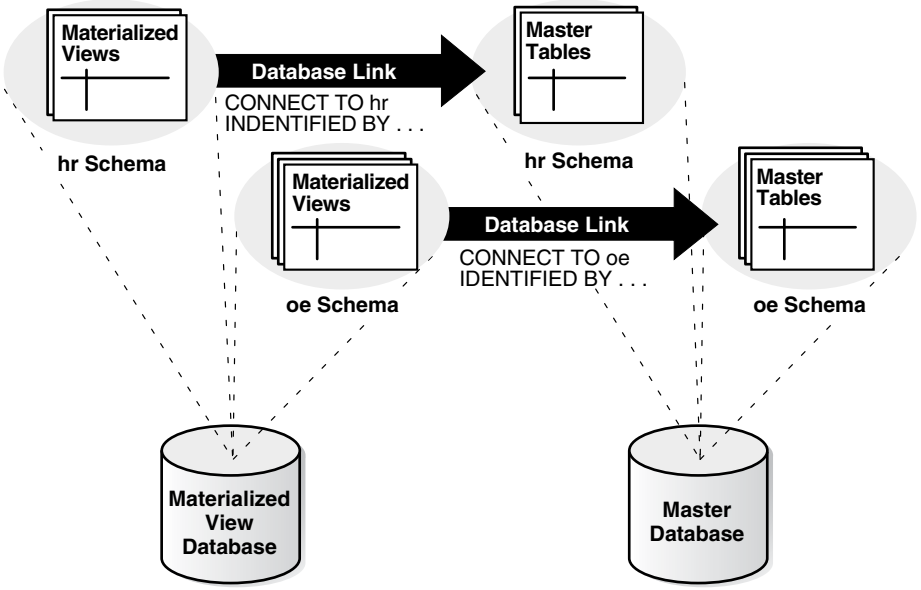
To ensure proper behavior, a materialized view's defining query must use a database link that includes an embedded user name and password in its definition; you cannot use a public database link when creating a materialized view. A database link with an embedded name and password always establishes connections 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 materialized view's defining query.

Before creating your materialized views, you need to create several administrative database links. Specifically, you should create a `PUBLIC` database link from the materialized view 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 materialized view administrator to the proxy administrator and from the propagator to the receiver, but, if you use the Advanced Replication interface's Configure Master and Materialized View Sites for Replication Wizard, then these database links are created for you automatically.

> **See Also:** The information about security options in *Oracle Database Advanced 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 materialized view schema at the materialized view 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 materialized view database. For example, the `hr` schema at a materialized view database should have a private database link to the master database that connects using the `hr` user name and password.

Figure 6–1 Recommended Schema and Database Link Configuration



For multimaster replication, there must be no Virtual Private Database (VPD) restrictions on the replication propagator and receiver schemas. For materialized views, the defining query for the materialized view cannot be modified by VPD. VPD must return a `NULL` policy for the schema that performs both the create and refresh of the materialized view. Creating a remote materialized view with a non-`NULL` VPD policy will not generate an error but might yield incorrect results.

See Also:

- *Oracle Database Administrator's Guide* for more information about database links
- *Oracle Database Security Guide* for more information about VPD
- *Oracle Label Security Administrator's Guide* for information about Advanced Replication and Oracle Label Security

Assign Privileges

Both the creator and the owner of the materialized view must be able to issue the defining `SELECT` statement of the materialized view. The owner is the schema that contains the materialized view. If a user other than the replication or materialized view administrator creates the materialized view, then that user must have the `CREATE MATERIALIZED VIEW` privilege and the appropriate `SELECT` privileges to execute the defining `SELECT` statement.

See Also: If you are reviewing the steps in *Oracle Database Advanced Replication Management API Reference*, then the necessary privileges are granted as part of the script described in instructions for creating a materialized view group. Privilege requirements are also described in "Required Privileges for Materialized View Operations" on page 3-10

Schedule Purge at Master Site

To keep the size of the deferred transaction queues in check, schedule a purge operation to remove all successfully completed deferred transactions from the

deferred transaction queue. This operation might have already been performed at the master site. Scheduling the purge operation again does not harm the master site, but might change the purge scheduling characteristics.

Schedule Push

Scheduling a push at the materialized view site automatically propagates the deferred transactions at the materialized view site to the associated target master site using a database link. These types of database links are called scheduled links. Typically, there is only a single scheduled link for each materialized view group at a materialized view site, because a materialized view group only has a single target master site.

Allocate Job Slaves

It is important that you have allocated sufficient job slaves to handle the automation of your replication environment. The job slaves automatically propagate the deferred transaction queue, purge the deferred transaction queue, refresh materialized views, and so on.

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

By the nature of materialized view replication, each materialized view site typically has one scheduled link to the master database and requires at least one job process. Materialized view sites typically require between one and three job processes, depending on purge scheduling, user-defined jobs, and the scheduled link. Also, you need at least one job slave for each degree of parallelism.

Alternatively, if your users are responsible for manually refreshing the materialized view through an application interface, then you do not need to create a scheduled link and your materialized view site requires one less job process.

The job slaves are defined using the `JOB_QUEUE_PROCESSES` initialization parameter in the initialization parameter file for your database. This initialization parameter is modifiable. Therefore, you can modify it while an instance is running. Oracle automatically determines the interval for job slaves. That is, Oracle determines when the job slaves should "wake up" to execute jobs.

> **See Also:** "Initialization Parameters" on page 6-4 and the *Oracle Database Reference* for information about `JOB_QUEUE_PROCESSES`

Creating a Materialized View Log

Before creating materialized view groups and materialized views for a remote materialized view site, ensure that you create the necessary materialized view logs at the master site or master materialized view site. A materialized view log is necessary for every master table or master materialized view that supports at least one materialized view with fast refreshes.

To create a materialized view log, you need the following privileges:

- `CREATE ANY TABLE`
- `CREATE ANY TRIGGER`
- `SELECT` (on the materialized view log's master)
- `COMMENT ANY TABLE`

> **See Also:** The Advanced Replication interface's online Help for detailed information about creating materialized view logs at the master site or master materialized view site with the Advanced Replication interface in Oracle Enterprise Manager.

Logging Columns in the Materialized View Log

When you create a materialized view log, you can add columns to the log when necessary. To perform a fast refresh on a materialized view, the following types of columns must be added to the materialized view log:

- A column referenced in the `WHERE` clause of a subquery that is not part of an equi-join and is not a primary key column. These columns are called filter columns.
- A column in an equi-join that is not a primary key column, if the subquery is either many to many or one to many. If the subquery is many to one, then you do not need to add the join column to the materialized view log.

A collection column cannot be added to a materialized view log. Also, materialized view logs are not required for materialized views that use complete refresh.

For example, consider the following DDL:

```
1) CREATE MATERIALIZED VIEW oe.customers REFRESH FAST AS
2)  SELECT * FROM oe.customers@orc1.example.com c
3)  WHERE EXISTS
4)    (SELECT * FROM oe.orders@orc1.example.com o
5)     WHERE c.customer_id = o.customer_id AND o.order_total > 20000);
```

Notice in line 5 of the preceding DDL that three columns are referenced in the `WHERE` clause. Columns `orders.customer_id` and `customers.customer_id` are referenced as part of the equi-join clause. Because `customers.customer_id` is a primary key column, it is logged by default, but `orders.customer_id` is not a primary key column and so must be added to the materialized view log. Also, the column `orders.order_total` is an additional filter column and so must be logged.

Therefore, add `orders.customer_id` and `orders.order_total` the materialized view log for the `oe.orders` table.

To create the materialized view log with these columns added, issue the following statement:

```
CREATE MATERIALIZED VIEW LOG ON oe.orders
  WITH PRIMARY KEY (customer_id,order_total);
```

If a materialized view log already exists on the `oe.customers` table, you can add these columns by issuing the following statement:

```
ALTER MATERIALIZED VIEW LOG ON oe.orders ADD (customer_id,order_total);
```

If you are using user-defined data types, then the attributes of column objects can be logged in the materialized view log. For example, the `oe.customers` table has the `cust_address.postal_code` attribute, which can be logged in the materialized view log by issuing the following statement:

```
ALTER MATERIALIZED VIEW LOG ON oe.customers ADD (cust_address.postal_code);
```

You are encouraged to analyze the defining queries of your planned materialized views and identify which columns must be added to your materialized view logs. If you try to create or refresh a materialized view that requires an added column without

adding the column to the materialized view log, then your materialized view creation or refresh might fail.

Note: To perform a fast refresh on a materialized view, you must add join columns in subqueries to the materialized view log if the join column is not a primary key and the subquery is either many to many or one to many. If the subquery is many to one, then you do not need to add the join column to the materialized view log.

See Also:

- "Data Subsetting with Materialized Views" on page 3-12 for information about materialized views with subqueries
- "Restrictions for Materialized Views with Subqueries" on page 3-19 for additional information about materialized views with subqueries
- "Creating a Materialized View Log" on page 6-12 for information about creating a materialized view log

Creating a Materialized View Environment

Materialized view 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 precreate a materialized view environment that will be individually deployed to the target materialized view site.

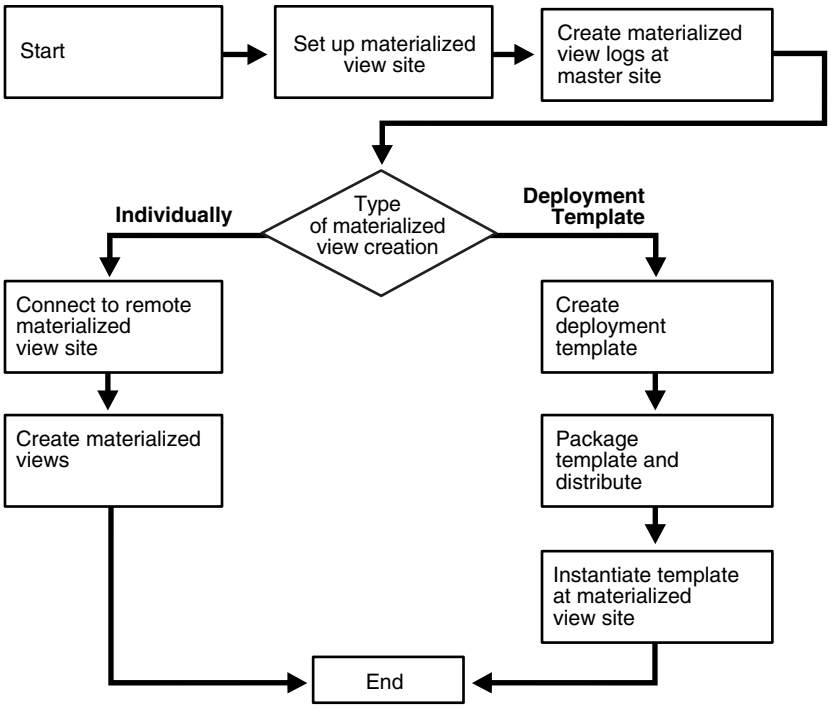
You can also individually create the materialized view environment by establishing a connection to the materialized view site and building the materialized view environment directly.

Creating a Materialized View Environment Using the Replication Management Interface

See the Advanced Replication interface's online Help in Oracle Enterprise Manager for information about using deployment templates to centrally create a materialized view environment using the Advanced Replication interface.

See the Advanced Replication interface's online Help in Enterprise Manager for information about individually creating the materialized view environment with a direct connection to the remote materialized view site using the Advanced Replication interface.

Figure 6–2 Flowchart for Creating Materialized Views



Creating a Materialized View Environment Using the Replication Management API

The instructions for creating a deployment template in the *Oracle Database Advanced Replication Management API Reference* manual for information about using deployment templates to centrally precreate a materialized view environment using the replication management API.

The instructions for creating a materialized view group the *Oracle Database Advanced Replication Management API Reference* manual for information about individually creating the materialized view environment with a direct connection to the remote materialized view site using the replication management API.

Avoiding Problems When Adding a New Materialized View Site

After you have created a materialized view environment with one or more materialized view sites, you might need to add new materialized view sites. You might encounter problems when you try to perform a fast refresh on the materialized views you create at a new materialized view site if both of the following conditions are true:

- Materialized views at the new materialized view site and existing materialized views at other materialized view sites are based on the same master table or master materialized view.
- Existing materialized views can be refreshed while you create the new materialized views at the new materialized view site.

The problem arises when the materialized view logs for the masters are purged before a new materialized view can perform its first fast refresh. If this happens and you try to perform a fast refresh on the materialized views at the new materialized view site, then you might encounter the following errors:

```
ORA-12004 REFRESH FAST cannot be used for materialized view materialized_view_name
ORA-12034 materialized view log on materialized_view_name younger than last
refresh
```

If you receive these errors, then the only solution is to perform a complete refresh of the new materialized view.

To avoid this problem, choose one of the following options:

- Use deployment templates to create the materialized view environment at materialized view sites. You will not encounter this problem if you use deployment templates.

  See Also: Chapter 4, "Deployment Templates Concepts and Architecture" for information about deployment templates

- Create a dummy materialized view at the new materialized view site before you create your production materialized views. The dummy materialized view ensures that the materialized view log will not be purged while your production materialized views are being created.

If you choose to create a dummy materialized view at the materialized view site, complete the following steps:

1. Create a dummy materialized view called `dummy_mview` based on the master table or master materialized view. For example, to create a dummy materialized view based on a master table named `sales`, issue the following statement at the new materialized view site:

   ```
   CREATE MATERIALIZED VIEW dummy_mview REFRESH FAST AS
     SELECT * FROM pr.sales@orc1.example.com WHERE 1=0;
   ```

2. Create your production materialized views at the new materialized view site.
3. Perform fast refresh of your production materialized views at the new materialized view site.
4. Drop the dummy materialized view.

Interoperability in an Advanced Replication Environment

If you plan to configure an Advanced Replication environment that involves different releases of Oracle Database at different replication sites, then your environment must meet the following requirements:

- Oracle Database 11*g* master sites can only interact with Oracle9*i* Release 2 (9.2) or later master sites.
- Oracle Database 11*g* materialized view sites can only interact with Oracle9*i* Release 2 (9.2) or later master sites.
- Oracle Database 11*g* master sites can only interact with Oracle9*i* Release 2 (9.2) or later materialized view sites.

  See Also: "Replication Support for Unicode" on page B-5 for information about interoperability in Advanced Replication environments that use `NCHAR` or `NVARCHAR` data types

Guidelines for Scheduled Links

A scheduled link determines how a master site propagates its deferred transaction queue to another master site, or how a materialized view site propagates its deferred transaction queue to its master site or master materialized view 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 the `parallelism` parameter to 1.

Before creating the scheduled links for a replication environment, carefully consider how you want replication to occur globally throughout the system. For example, you can 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, then you might want to have each scheduled link continuously push a master site's deferred transaction queue to its destination.

Also, you might want to schedule pushes 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 6-21 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, you can use the `DBMS_DEFER_SYS.SCHEDULE_PUSH` procedure and specify the settings shown in Table 6–3.

Table 6–3 Settings to Schedule Periodic Pushes

SCHEDULE_PUSH Procedure Parameter	Value
`delay_seconds`	0
`interval`	An appropriate date expression; for example, to specify an interval of one hour, use `'sysdate + 1/24'`

You can also use the Advanced Replication interface in Oracle Enterprise Manager to schedule periodic pushes. To do so, set delay seconds to 0 (zero) when configuring a scheduled link.

Then configure the interval (the "then push every" control) to push the deferred transaction queue periodically.

The following is an example that schedules a periodic push once an hour:

```
BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PUSH (
      destination => 'orc2.example.com',
      interval => 'SYSDATE + (1/24)',
      next_date => SYSDATE,
```

```
      delay_seconds => 0);
END;
/
```

See Also:

- "Delay Seconds" on page 2-36 for more information about setting delay seconds
- *Oracle Database Advanced Replication Management API Reference* for information about the `DBMS_DEFER_SYS.SCHEDULE_PUSH` procedure
- The Advanced Replication interface online Help for information about using this interface in Enterprise Manager

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, use the `DBMS_DEFER_SYS.SCHEDULE_PUSH` procedure and specify the settings shown in Table 6–4.

Table 6–4 Settings to Simulate Continuous Push

SCHEDULE_PUSH Procedure Parameter	Value
`delay_seconds`	1200
`interval`	Lower than the `delay_seconds` setting
`parallelism`	1 or higher
`execution_seconds`	Higher 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 the amount of time specified by the `delay_seconds` parameter, then Oracle releases the resources used by the job and starts fresh when the next job slave becomes available.

If you are using serial propagation by setting the `parallelism` parameter to 0 (zero), then you can simulate continuous push by reducing the settings of the `delay_seconds` and `interval` parameters to an appropriate value for your environment. However, if you are using serial propagation, simulating continuous push is costly when the push job must initiate often.

The following is an example that simulates continual pushes:

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

See Also:

- "Delay Seconds" on page 2-36 for more information about setting delay seconds
- "Serial and Parallel Propagation" on page 6-21 for more information about issues related to serial and parallel propagation
- *Oracle Database Advanced Replication Management API Reference* for information about the `DBMS_DEFER_SYS.SCHEDULE_PUSH` procedure

Guidelines for Scheduled Purges of a Deferred Transaction Queue

A scheduled purge determines how a master site or materialized view site purges applied transactions from its deferred transaction queue. When you use a configuration wizard in the Advanced Replication interface in Oracle Enterprise Manager to set up a master site or materialized view 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 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 can 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 if you can store an entire day's deferred transactions.

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. To do so, you can use the `DBMS_DEFER_SYS.SCHEDULE_PURGE` procedure and specify the settings shown in Table 6–5.

Table 6–5 Settings to Schedule Periodic Purges

SCHEDULE_PURGE Procedure Parameter	Value
`delay_seconds`	0
`interval`	An appropriate date expression; for example, to specify an interval of one day, use `'sysdate + 1'`

You can also use the Advanced Replication interface in Oracle Enterprise Manager to schedule periodic purges. To do so, set delay seconds to 0 (zero). Then configure the interval (the "then purge every" control) to purge the deferred transaction queue.

The following is an example that schedules a periodic purge once a day:

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

See Also:

- *Oracle Database Advanced Replication Management API Reference* for information about the `DBMS_DEFER_SYS.SCHEDULE_PURGE` procedure
- The Advanced Replication interface online Help for information about using this interface in Enterprise Manager

Scheduling Continuous Purges

To configure continuous purging of a site's deferred transaction queue, you can use the `DBMS_DEFER_SYS.SCHEDULE_PURGE` procedure and specify the settings shown in Table 6–6.

Table 6–6 Settings to Schedule Continuous Purges

SCHEDULE_PURGE Procedure Parameter	Value
`delay_seconds`	500000
`interval`	Lower than the `delay_seconds` setting
`purge_method`	`dbms_defer_sys.purge_method_quick` setting

You can also use the Advanced Replication interface to configure continuous purge. To do so, on the Purge Schedule page, set **Delay Seconds** to 500,000 and set interval (the **Every** field) to a value less than the **Delay Seconds** setting.

The following is an example that simulates continuous purges:

```
BEGIN
   DBMS_DEFER_SYS.SCHEDULE_PURGE (
      next_date => SYSDATE,
      interval => 'SYSDATE + (1/144)',
      purge_method => dbms_defer_sys.purge_method_quick,
      delay_seconds => 500000);
END;
/
```

See Also:

- "Delay Seconds" on page 2-36 for more information about setting delay seconds
- *Oracle Database Advanced Replication Management API Reference* for information about the `DBMS_DEFER_SYS.SCHEDULE_PURGE` procedure
- The Advanced Replication interface online Help for information about using this interface in Enterprise Manager

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 scheduled link with serial propagation, set the `parallelism` parameter to 0 (zero) in the `DBMS_DEFER_SYS.SCHEDULE_PUSH` procedure. Or, using the Advanced Replication interface in Oracle Enterprise Manager, set the **Propagation Processes** control to 0 (zero) in the Edit Push Schedule page.
- With parallel propagation, Oracle propagates replicated transactions using multiple parallel streams for higher throughput. When necessary, Oracle orders the execution of dependent transactions to preserve data integrity. To configure a scheduled link with parallel propagation, set the `parallelism` parameter to 1 or higher in the `DBMS_DEFER_SYS.SCHEDULE_PUSH` procedure. Or, using the Advanced Replication interface in Oracle Enterprise Manager, set the **Propagation Processes** control to 1 or higher in the Edit Push Schedule page. Typically, you should use parallel propagation.

See Also:

- "Parallel Propagation" on page 2-33
- *Oracle Database Advanced Replication Management API Reference* for information about the `DBMS_DEFER_SYS` package
- The Advanced Replication interface online Help for information about using this interface in Enterprise Manager

Deployment Templates

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

See Also: Chapter 4, "Deployment Templates Concepts and Architecture" for information about deployment templates

Preparing Materialized View Sites for Instantiation of Deployment Templates

If you decide to use deployment templates, then you need to prepare your materialized view sites for instantiation. If a deployment template has been designed well, then little preparation is necessary at the remote materialized view site. This section describes the most common preparations that must be performed at the remote materialized view site. After 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 materialized view site for instantiation:

- Does the remote materialized view site have network connectivity to the target master site?
- Does the materialized view site have an Oracle9*i* Release 2 (9.2) or later database?

- Has the remote materialized view site been set up to support materialized view replication?
- Do the schemas required by the deployment template exist at the materialized view site?
- If required database links are not part of the deployment template, then do the required database links from the materialized view site to the master site exist?
- Will you use online instantiation or offline instantiation to instantiate the deployment template at the materialized view sites?
- Do the rollback segments that might be required by the deployment template exist at the materialized view site and are they online?

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

Network Connectivity

As with all replication environments, network connectivity is a key component in Advanced Replication. Verify that the remote materialized view site has a proper Oracle Net connection to the target master site.

> **See Also:** *Oracle Database Net Services Administrator's Guide* for information about setting up an Oracle network connection

Database Version

The materialized view site must have an Oracle9*i* Release 2 (9.2) or later database to work with Oracle Database 11*g*. If your materialized view site does not meet the database version requirements, then you need to upgrade your database at the materialized view site before instantiating a deployment template.

Materialized View Site Setup

Each materialized view site needs several users that have special privileges to support a materialized view site. In addition to having the administrative privileges, these users also participate in the propagation and refreshing of data.

Materialized view site setup also includes scheduling several automated jobs to handle the automatic refreshing of the materialized view (optional) and the purging of the deferred transaction queue.

You can set up your materialized view site with:

- **Advanced Replication Interface:** You can connect to the remote materialized view site with the Advanced Replication interface and use the Configure Master and Materialized View Sites for Replication Wizard.

  > **See Also:** The Advanced Replication interface's online Help for instructions on setting up your materialized view site with the Advanced Replication interface in Oracle Enterprise Manager

- **Replication Management API:** Using the replication management API to setup your materialized view site is an ideal solution when you are not able to connect to the remote materialized view site with the Advanced Replication interface in Enterprise Manager. When you build a SQL script containing the API calls to setup your materialized view site, you can also add the DDL and API calls to complete the remaining preparation (such as creating any necessary schemas, 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:** *Oracle Database Advanced Replication Management API Reference* for instructions on setting up your materialized view site with the replication management API

Create Necessary Schemas

If the deployment template that you are instantiating creates objects in multiple schemas, then ensure 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 `oe` and the user `hr` is instantiating the template, then `hr` must have the `CREATE ANY PROCEDURE` privilege on schema `oe`.

Create Database Links

While it is advantageous to include the DDL to create all necessary database links for a remote materialized view site in the deployment template, it is not required. If the database link DDL is not in the deployment template, then manually create the database links to the target master site prior to instantiating the deployment template. The database links are required to populate the materialized views during an online instantiation and are required for the proper maintenance of the materialized view environment.

Online or Offline Instantiation

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

> **See Also:** "Deployment Template Packaging and Instantiation" on page 4-7 for more information about online and offline instantiation

Create Necessary Rollback Segments

If the deployment template that you are instantiating will use specific rollback segments that do not currently exist at the remote materialized view site, then 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` data dictionary view.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* for information about data dictionary views related to replication

Conflict Resolution

Asynchronous multimaster and updatable materialized view replication environments must address the possibility of replication conflicts that can 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 can occur in your replication environment, then 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:** Chapter 5, "Conflict Resolution Concepts and Architecture", for more information about avoiding conflicts and for information about the conflict resolution methods available to you if conflicts can occur

Security and Replication

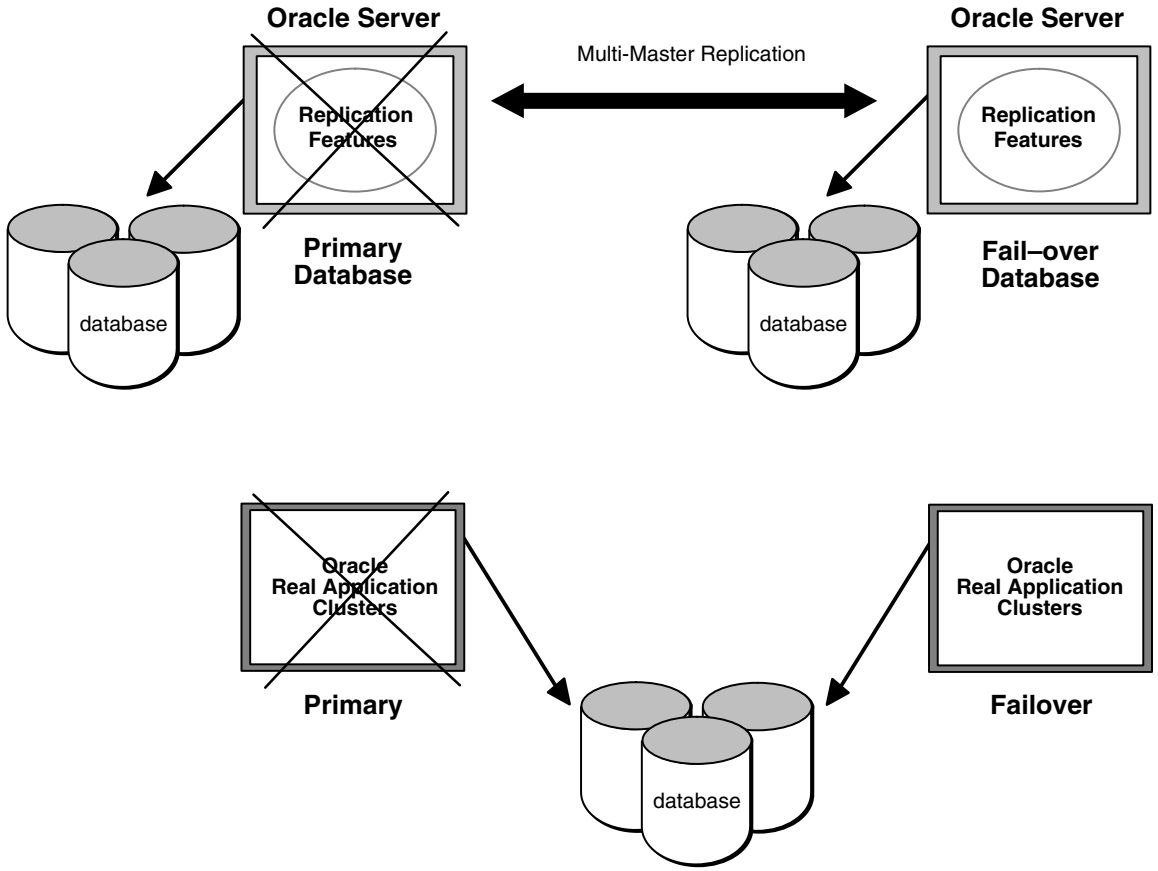
Security might be a concern in both multimaster and materialized view replication environments. You should plan your security strategy before you configure your replication environment.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* for information about security options in a replication environment

Designing for Survivability

Survivability is the capability to continue running applications despite system or site failures. Survivability enables 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 6–3, the Oracle server provides two different technologies for accomplishing survivability: multimaster replication and Oracle Real Application Clusters (Oracle RAC).

Figure 6–3 Survivability Methods: Replication Or Oracle Real Application Clusters



Oracle Real Application Clusters versus Replication

Oracle Real Application Clusters (Oracle RAC) supports fail over to surviving systems when a system supporting an instance of the Oracle server fails. Oracle RAC 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, Oracle RAC is a good 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 in an Oracle RAC environment, then a surviving instance automatically recovers any incomplete transactions. Applications running on the failed system can execute on the fail over system, accessing all data in the database.

Oracle RAC does not, however, provide survivability for site failures (such as power outages, 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 multimaster replication to maintain a replica of a database at a geographically remote location. In addition, you can use multimaster replication to replicate data between sites running different operating systems or different releases of Oracle or both.

Should the local system fail, the application can continue to execute at the remote site. Using multimaster replication, some administrative procedures might be necessary to recover transactions at the failed site and to prevent data inconsistencies when restarting the failed site.

Note: You can also configure standby database to protect an Oracle database from site failures.

See Also:

- *Oracle Real Application Clusters Administration and Deployment Guide*
- *Oracle Data Guard Concepts and Administration* for more information about standby database

Designing a Replication Environment for Survivability

If you choose to use the replication facility for survivability, then consider the following issues:

- The replication facility must be able to keep up with the transaction volume of the primary system.
- If a failure occurs at the primary site, then recently committed transactions at the primary site might not have been asynchronously propagated to the failover 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 failover site. The first of these was a transaction that entered a new order, and the second was a transaction that canceled an existing order.

  In the first case, someone might notice the absence of the new order when processing continues on the fail over system, and reenter it. In the second case, the cancellation of the order might not be noticed, and processing of the order might proceed; that is, the canceled item might 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 failover system back to the primary system, then you will encounter conflicts.

  Specifically, duplicate orders exist for the item originally ordered at the primary system just before it failed. Additionally, data changes result 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

Advanced Replication provides survivability against site failures by using multiple replicated master sites. You must configure your system using one of the following methods, which 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 Oracle Net to configure automatic connect-time failover, which enables Oracle Net 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:** *Oracle Database Net Services Administrator's Guide* for more information about configuring connect-time failover

Database Recovery in Replication Environments

If you recover a database that is a master site in a replication environment, then the replicated data might not be consistent, and you might encounter replication errors. For example, a restored master site might have propagated different transactions to different masters. You might need to perform extra steps to correct for an incorrect recovery operation. One such method is to drop and re-create all replicated objects in the recovered database.

Before dropping and re-creating replicated objects, remove pending deferred transactions and deferred error records from the restored database, and resolve any outstanding distributed transactions. If the restored database was a master definition site for some replication environments, then you should designate a new master definition site before dropping and creating objects. Any materialized views that are mastered at the restored database should be fully refreshed, as well as any materialized views in the restored database.

To provide continued access to your data, you might need to change master definition sites (assuming the database being recovered was the master definition site), or change the master site of materialized view 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 Advanced Replication, such as the `DBA_REPSITES` data dictionary 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 `hr_repg` replication group at a materialized view site to ensure that they have the appropriate object identifiers and status values:

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

> **See Also:** The `REPCAT_IMPORT_CHECK` procedure in *Oracle Database Advanced Replication Management API Reference*

A

Troubleshooting Replication Problems

This appendix contains troubleshooting guidelines for managing a replication environment.

This appendix contains these topics:

Diagnosing Problems with Database Links

If you think a database link is not functioning properly, then you can drop and re-create it using Oracle Enterprise Manager, SQL*Plus, or another tool.

- Ensure that the database link name is the same as the global name of the target database.
- Ensure that the scheduled interval is what you want.
- Ensure 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, then the other sites that link to that site must have the same connection qualifier. For example, suppose you create a database link as follows:

```
CREATE DATABASE LINK dbs1.example.com@myethernet CONNECT TO repadmin
  IDENTIFIED BY password USING 'connect_string_myethernet'
```

All the sites, whether masters or materialized views, associated with `dbs1.example.com@myethernet` must include `myethernet` as the connection qualifier.

> **See Also:** *Oracle Database Administrator's Guide* for more information database links and connection qualifiers

Diagnosing Problems with Master Sites

Problems can arise in a multimaster replication system. The following sections discuss some problems and ways to solve 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, then try the following:

- Ensure that the necessary private database links exist between the new master site and the existing master sites. If you used a configuration wizard in the Advanced Replication interface in Oracle Enterprise Manager to set up your sites, then 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.
- Ensure that the administrative requests at all sites have completed successfully. If requests have not been executed yet, then you can manually execute pending administrative 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, then first ensure that the administrative requests at all sites have completed successfully. If requests are pending execution, then 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 supply the following as the `ddl_text` parameter to the `DBMS_REPCAT.CREATE_MASTER_ REPOBJECT` procedure:

```
CREATE TABLE oe.new_employees AS SELECT * FROM hr.employees WHERE ...;
```

Because each table name contains a schema name, this statement works whether the replication administrator is `oe`, `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 asynchronously propagated to the other sites in your replication environment, then try the following:

- Use the Advanced Replication interface in Oracle Enterprise Manager to check whether the corresponding deferred transaction has been pushed to the destination. If not, then 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, then you can execute deferred transaction manually.
- If a scheduled link's interval has passed and corresponding deferred transactions have not been pushed, then check the corresponding job for the link.
- Even after propagating a deferred transaction to a destination, it might not execute because of an error. Check the DEFERROR data dictionary view at the destination site for errors.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* for information about modifying tables without replicating the modifications, which might be necessary when you need to manually synchronize the data in replicated tables

DML Cannot be Applied to Replicated Table

If you receive the deferred_rpc_quiesce exception when you attempt to modify a replicated table, then the master group to which your replicated object belongs is quiescing or quiesced. To proceed, your replication administrator must resume replication activity for the master 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, then update one row at a time in an appropriate order. This lets you define the order of update requests in the deferred transactions queue.

Re-creating a Replicated Object

If you add an object such as a package, procedure, or view to a master group, then the status of the object must be valid. If the status of an object is invalid, then recompile the object or drop and re-create the object before adding it to a master group. Check the DBA_REPOBJECT data dictionary view for the status of replication objects.

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 configuration wizards in the Advanced Replication interface in Oracle Enterprise Manager and the DBMS_REPCAT_ADMIN package both perform 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, then carefully review the procedure to ensure that the replicated procedure did not cause the conflict. Complete the following checks:

- 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 reenabled at the end.
- Ensure that row-level replication is reenabled even if exceptions occur when the procedure executes.
- Check to be sure that the replicated procedure executed at all master sites.

You should perform similar checks on any replicated triggers that you have defined on replicated tables.

Diagnosing Problems with the Deferred Transaction Queue

If deferred transactions at a site are not being pushed to their destinations, then the following sections explain some possible causes for the problem:

- Check Jobs for Scheduled Links
- Distributed Transaction Problems with Synchronous Replication
- Incomplete Database Link Specifications
- Incorrect Replication Catalog Views

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, then you should first be certain that you are not experiencing a problem with the job queue.

Distributed Transaction Problems with Synchronous Replication

When you use synchronous replication, Oracle uses a distributed transaction to ensure that the transaction has been properly committed at the remote site. Distributed transactions use two-phase commit. Asynchronous replication does not use two-phase commit.

> **See Also:** *Oracle Database Administrator's Guide* for information about diagnosing problems with distributed transactions

Incomplete Database Link Specifications

If you notice that transactions are not being pushed to a given remote site, then you might 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.

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, then ensure that the view definitions in `catrepc.sql` are subsequently loaded.

Diagnosing Problems with Materialized Views

There are a number of problems that might happen with materialized view sites in a replication system. The following sections discuss some problems and ways to troubleshoot them:

- Problems Creating Replicated Objects at Materialized View Site
- Refresh Problems
- Advanced Troubleshooting of Refresh Problems

Problems Creating Replicated Objects at Materialized View Site

If you unsuccessfully attempt to create a new object at a materialized view site, then try the following:

- For an updatable materialized view, check that the associated master table or master materialized view has a materialized view log.
- Ensure that you have the necessary privileges to create the object. For a materialized view, you need `SELECT` privilege on the master table or master materialized view and its materialized view log. See "Assign Privileges" on page 6-11 for more information.
- If you are trying to add an existing materialized view to a materialized view group, then try re-creating the materialized view when you add it to the group.
- If you are trying to create a fast refresh primary key or subquery materialized view, then ensure that the materialized view log on the master table or master materialized view logs primary keys.
- If you are trying to create a fast refresh rowid materialized view, then ensure that the materialized view log on the master table logs rowids.
- Check if the materialized view log has the required columns added for subquery materialized views. See "Logging Columns in the Materialized View Log" on page 6-13 for information.
- Check if the materialized view log exists for all tables that are involved in a fast refresh materialized view. If the materialized view contains a subquery, then each table referenced in the subquery should have a materialized view log.

Problems Performing Offline Instantiation of a Deployment Template

If you receive and error stating that Oracle is unable to initialize the extent in the temporary tablespace when you try to instantiate a deployment template offline, then you might need to adjust the data file for the temporary database so that it auto extends.

For example, issue the following statement to adjust the data file:

```
ALTER DATABASE TEMPFILE '/u02/oracle/rbdb1/temp.dbf'
    AUTOEXTEND ON
    NEXT 10M;
```

After you have made this adjustment, instantiate the deployment template offline at the materialized view site.

Refresh Problems

The following sections explain several common materialized view refresh problems.

Common Refresh Problems

Several common factors can prevent the automatic refresh of a group of materialized views:

- The lack of a job slave at the materialized view database
- An intervening network or server failure
- An intervening server shutdown

When a materialized view refresh group is experiencing problems, ensure that none of the preceding 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 16 attempts to refresh a refresh group Oracle continues to encounter errors, then 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 materialized view 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 materialized view trace file is of the form j*n*, where *n* is operating system specific. See the Oracle documentation for your operating system for the name on your system.

Fast Refresh Errors at New Materialized View Sites

In some cases, a materialized view log for a master table or master materialized view might be purged during the creation of a materialized view at a new materialized view site. When this happens, you might encounter the following errors:

```
ORA-12004 REFRESH FAST cannot be used for materialized view materialized_view_name
```

```
ORA-12034 materialized view log on materialized_view_name younger than last
refresh
```

> **See Also:** "Avoiding Problems When Adding a New Materialized View Site" on page 6-15 for a complete description of how to avoid this problem.

Materialized Views Continually Refreshing

If you encounter a situation where Oracle continually refreshes a group of materialized views, then 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 materialized views in the group, then Oracle continually starts a group refresh each time the job slave checks the queue of outstanding jobs.

Materialized View Logs Growing Too Large

If a materialized view log at a master site or master materialized view site is growing too large, then check to see whether a network or site failure has prevented the master site or master materialized view site from becoming aware that a materialized view has been dropped. You might need to purge part of the materialized view log or unregister the unused materialized view site.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* for more information about managing materialized view logs

Advanced Troubleshooting of Refresh Problems

If you have a problem refreshing a materialized view, then try the following:

- Check the `NEXT_DATE` value in the `DBA_REFRESH_CHILDREN` view to determine if the refresh has been scheduled.
- If the refresh interval has passed, then check the `DBA_REFRESH` view for the associated job number for the materialized view refresh and then diagnose the problem with job queues.
- Check if there are job slaves running. Check the `JOB_QUEUE_PROCESSES` initialization parameter, query the `DBA_JOBS_RUNNING` view, and use your operating system to check if the job slaves are still running.
- You also might encounter an error if you attempt to define a master detail relationship between two materialized views. You should define master detail relationships only on the master tables by using declarative referential integrity constraints. The related materialized views should then be placed in the same refresh group to preserve this relationship. However, you can define deferred (or deferrable) constraints on materialized views.
- If there are any outstanding conflicts recorded at the master site or master materialized view site for the materialized views, then you can only refresh the materialized views by setting the parameter `REFRESH_AFTER_ERRORS` to `TRUE`. This parameter can be set when you create or alter a materialized view refresh group. There is a corresponding parameter for the Advanced Replication interface in Oracle Enterprise Manager.

- Materialized views 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 materialized view 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, then the master site or master materialized view site might have run out of rollback segments when trying to maintain the materialized view log, or the materialized view log might be out of date. For example, the materialized view log might have been purged or re-created.
- Complete refreshes of a single materialized view internally use the `TRUNCATE` feature to increase speed and reduce rollback segment requirements. However, until the materialized view refresh is complete, users might temporarily see no data in the materialized view. Refreshes of multiple materialized views (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 materialized views to do complete refreshes. Otherwise, the materialized views have incorrect references to master table rowids. You use the `BEGIN_TABLE_REORGANIZATION` and `END_TABLE_REORGANIZATION` procedures in the `DBMS_MVIEW` package to reorganize a master table. See the *Oracle Database Advanced Replication Management API Reference* for more information.
- If while refreshing you see an `ORA-00942` (table or view does not exist), then check your database links and ensure that you still have the required privileges on the master table or master materialized view and the materialized view log.
- If a fast refresh was succeeding but then fails, then check whether:
  - The materialized view log was truncated, purged, or dropped.
  - You still have the required privileges on the materialized view log.
- If a force refresh takes an inordinately long time, then check if the materialized view log used by the refresh has been dropped.
- If the materialized view was created with `BUILD DEFERRED`, and its first fast refresh fails, then ensure that a previous complete refresh was done successfully before checking for other problems.

> **See Also:** *Oracle Database Advanced Replication Management API Reference* for information about managing materialized view logs

B

Column Length Semantics and Unicode

This appendix contains information about replication support for column length semantics and Unicode.

This appendix contains these topics:

> **See Also:** The following documents contain more information about length semantics and Unicode:
>
> - *Oracle Database Globalization Support Guide*
> - *Oracle Database SQL Language Reference*

Column Length Semantics for Replication Sites and Table Columns

Column length semantics determine whether the length of a column is specified in bytes or in characters. You use `BYTE` to specify that the length is in bytes, and you use `CHAR` to specify that the length is in characters. `CHAR` length semantics is also known as codepoint length semantics.

Because some character sets require more than one byte for each character, a specification of `10 BYTE` for a column might actually store less than 10 characters for certain character sets, but a `10 CHAR` specification ensures that the column can store 10 characters, regardless of the character set. Only Oracle9*i* Database or later databases can specify `CHAR` length semantics.

You set the length semantics for an Oracle database using the `NLS_LENGTH_SEMANTICS` initialization parameter, and all `VARCHAR2` and `CHAR` columns use the setting specified for this initialization parameter as the default. If this initialization parameter is not set, then the default setting is `BYTE`.

An individual column can override the length semantics for the database. For example, if the length semantics for a site is `CHAR`, then you can still specify `BYTE` for the length semantics of an individual column using the `CREATE TABLE` or `ALTER TABLE` statement.

The following statement creates a table and specifies the column length in bytes:

```
CREATE TABLE byte_col (a VARCHAR2(10 BYTE));
```

The following statement creates a table and specifies the column length in characters:

```
CREATE TABLE char_col (a VARCHAR2(10 CHAR));
```

Multimaster Support for Column Length Semantics

All master sites in a master group must have the same length semantics, and the individual columns of a master table must have the same length semantics at all master sites. When you have a table in a master group at a master definition site and you want to replicate that table to a new master site, you can create the table at the new site in one of the following ways:

- Specify that Advanced Replication generate the table at the new master site when adding the new master site to the master group.
- Manually precreate the table at the new master site before adding the master site to the master group.

The following sections describe column length semantics support for each table creation method.

Column Length Semantics Support for Tables Generated by Advanced Replication

When you specify that Advanced Replication generate the table at the new master site, and you are using `CHAR` length semantics, then both the master definition site and the new master site must be running Oracle9*i* Database or later. If you specify `BYTE` length semantics, then these sites can be running a previous Oracle release.

This support is summarized in Table B–1.

Table B–1 Column Length Semantics Support for Generated Tables

Master Definition Site Release	Master Definition Site Column Semantics	New Master Site Release	Resulting Column Semantics at New Master Site
9.2 or later	`CHAR`	9.2 or later	`CHAR`
9.2 or later	`CHAR`	Prior to 9.2	Not supported
Any release	`BYTE`	Any release	`BYTE`

Column Length Semantics Support for Precreated Tables

When you precreate the table at the new master site, and you are using `CHAR` length semantics, then both the master definition site and the new master site must be running Oracle9*i* Database or later. If you specify `BYTE` length semantics, then these sites can be running a previous Oracle release.

Also, because you precreated the table manually, it is possible that you specified a different length semantics for a column in the new master table than was specified for the column in the table at the master definition site. If so, Oracle raises an error because a column in a master table must be using the same length semantics at each master site.

This support is summarized in Table B–2.

Table B–2 Column Length Semantics Support for Precreated Tables

Master Definition Site Release	Master Definition Site Column Semantics	New Master Site Release	New Master Site Column Semantics	Supported?
9.2 or later	CHAR	9.2 or later	CHAR	Yes
9.2 or later	CHAR	9.2 or later	BYTE	No
9.2 or later	BYTE	9.2 or later	CHAR	No
9.2 or later (Multibyte character set)	CHAR	Prior to 9.2	BYTE	No
9.2 or later (Single-byte character set)	CHAR	Prior to 9.2	BYTE	Yes
Prior to 9.2	BYTE	9.2 or later (Multibyte character set)	CHAR	No
Prior to 9.2	BYTE	9.2 or later (Single-byte character set)	CHAR	Yes
Any release	BYTE	Any release	BYTE	Yes

Materialized View Support for Column Length Semantics

When you create a materialized view, Oracle determines the length semantics of the columns in the materialized view in the following way:

- If the master column is explicitly specified as either BYTE or CHAR, then the column in the materialized view retains that specification. In the following example, CHAR length semantics is explicitly specified for the a column:

```
CREATE TABLE char_col (a VARCHAR2(10 CHAR));
```

- If the master column is not explicitly specified, then the column in the materialized view uses the default length semantics of the materialized view site. In the following example, length semantics is not explicitly specified for the a column:

```
CREATE TABLE char_col (a VARCHAR2(10));
```

Materialized view creation fails if an Oracle9*i* Database or later master has a column with an explicit CHAR specification and a materialized view site running a release prior to Oracle9*i* Database attempts to create a materialized view based on this master.

Materialized Views with Prebuilt Container Tables

If you prebuild a container table at a materialized view site before you create the materialized view, then the length semantics of the columns in the container table must match the length semantics of the columns in the master. If the length semantics do not match, then an Oracle returns an ORA-12060 error during materialized view creation. You use the ON PREBUILT TABLE clause of the CREATE MATERIALIZED VIEW statement to prebuild a table for a materialized view.

See Also: The *Oracle Database SQL Language Reference* for more information about the `ON PREBUILT TABLE` clause in the `CREATE MATERIALIZED VIEW` statement

Column Length Semantics Support for Updatable Materialized Views

The following operations are always supported if the length semantics of the columns of an updatable materialized view matches the length semantics of the columns of the materialized view's master:

- Refreshing the updatable materialized view
- Pushing DML changes made at the materialized view to the master

If, however, the length semantics do not match and the master is Oracle9*i* Database or later, then Oracle raises an error when you try to add the materialized view to a materialized view group. To be updatable, a materialized view must belong to a materialized view group. If you use the replication management API, then you run the `CREATE_MVIEW_REPOBJECT` procedure in the `DBMS_REPCAT` package to add the materialized view to a materialized view group.

Table B–3 summarizes the length semantics support for updatable materialized views.

Table B–3 Column Length Semantics Support for Updatable Materialized Views

Master Site Release	Master Site Column Semantics	Materialized View Site Release	Materialized View Site Column Semantics	Updatable Materialized View Supported?
9.2 or later	`CHAR`	9.2 or later	`CHAR`	Yes
9.2 or later	`CHAR`	9.2 or later	`BYTE`	No
9.2 or later	`BYTE`	9.2 or later	`CHAR`	No
9.2 or later (Multibyte character set)	`CHAR`	Prior to 9.2	`BYTE`	No
9.2 or later (Single-byte character set)	`CHAR`	Prior to 9.2	`BYTE`	Yes
Prior to 9.2	`BYTE`	9.2 or later	`CHAR`	Yes
Any release	`BYTE`	Any release	`BYTE`	Yes

Note: The master site in Table B–3 can be either a master site in a multimaster replication environment or a master materialized view site.

DDL Propagation and Column Length Semantics

You can use the `DBMS_REPCAT` package to propagate a data definition language (DDL) statement that creates a new replicated table or adds columns to an existing replicated table. If you want any of the new columns created by these DDL statements to use `CHAR` column length semantics, then ensure that you specify `CHAR` column length semantics explicitly. Otherwise, the column always has `BYTE` length semantics, even if the replication site itself has `CHAR` column length semantics set as the default.

The following procedures in the DBMS_REPCAT package enable you to propagate DDL statements:

- ALTER_MASTER_REPOBJECT
- CREATE_MASTER_REPOBJECT
- EXECUTE_DDL

> **See Also:** "Column Length Semantics for Replication Sites and Table Columns" on page B-1 for more information about specifying CHAR column length semantics explicitly

Replication Support for Unicode

Unicode is a universal encoded character set that enables you to store information from any language using a single character set. Unicode provides a unique code value for every character, regardless of the platform, program, or language. Unicode is supported in both multimaster and materialized view replication environments. In Oracle9*i* Database or later, all columns specified as NCHAR or NVARCHAR2 data type are stored in Unicode format.

For both master sites and materialized view sites, replication is possible in an environment with different releases of Oracle using an NCHAR or NVARCHAR2 data type. However, replication is not recommended when one of the replication sites is a release prior to Oracle9*i* Database and uses a variable width character set because, in this case, there is a possibility of data loss.

Table B–4 summarizes when replication is recommended.

Table B–4 Replication Support for Globalization Support Character Sets

Release of Local Database with NCHAR or NVARCHAR2 Columns	Release of Remote Database with NCHAR and NVARCHAR2 Columns	Replication Recommended?
9.2 or later (Stored in Unicode format)	9.2 or later (Stored in Unicode format)	Yes
Prior to 9.2 (Fixed or variable width national character set format)	Prior to 9.2 (Fixed or variable width national character set format)	Yes
9.2 or later (Stored in Unicode format)	Prior to 9.2 (Variable width national character set format)	Not Recommended
9.2 or later (Stored in Unicode format)	Prior to 9.2 (Fixed width national character set format)	Yes
Prior to 9.2 (Variable width national character set format)	9.2 or later (Stored in Unicode format)	Not Recommended
Prior to 9.2 (Fixed width national character set format)	9.2 or later (Stored in Unicode format)	Yes

Caution: Where Table B–4 specifies that replication is not supported, Oracle does not detect an error when you set up replication between the two sites, but data loss can occur later. If data loss occurs, then an error is raised.

Replication of NCLOB Data Type Columns

`NCLOB` data type columns are always fixed width. Therefore, replication of `NCLOB` data type columns is supported without restrictions.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W