# Oracle8™

Replication

Release 8.0

December, 1997

Part No. A58245-01

ORACLE®

Enabling the Information Age™

Oracle8 Replication

Part No. A58245-01

Release 8.0

Copyright © 199x, 1997, Oracle Corporation. All rights reserved.

Printed in the U.S.A

Primary Authors: Steve Bobrowski, Gordon Smith.

Contributing Authors: Alan Downing, Thomas Albert, Sandy Dreskin, Harry Sun, Tom Bishop, Horst Heistermann, Al Demers, Maria Pratt, Peter Vasterd, Jim Stamos, and others.

Graphic Designer: Valarie Moore.

**The programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be licensee's responsibility to take all appropriate fail-safe, back up, redundancy and other measures to ensure the safe use of such applications if the programs are used for such purposes, and Oracle disclaims liability for any damages caused by such use of the programs.**

This program contains proprietary information of Oracle Corporation; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright patent and other intellectual property law. Reverse engineering of the software is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free.

If this program is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

**Restricted Rights Legend** Programs delivered subject to the DOD FAR Supplement are 'commercial computer software' and use, duplication and disclosure of the programs shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, programs delivered subject to the Federal Acquisition Regulations are 'restricted computer software' and use, duplication and disclosure of the programs shall be subject to the restrictions in FAR 52..227-14, Rights in Data -- General, including Alternate III (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

Oracle, Net8, and SQL*Plus are registered trademarks of Oracle Corporation. Oracle8, Server Manager, Enterprise Manager, Replication Manager, Oracle Parallel Server and PL/SQL are trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

# Contents

**Send Us Your Comments** ..... xxiii

**Preface** ..... i

**Overview of The Oracle8 Replication Manual** ..... ii
**Audience** ..... iii
Knowledge Assumed of the Reader ..... iii
**How The Oracle8 Replication Manual Is Organized** ..... iii
**Conventions Used in This Manual** ..... v
Special Notes ..... v
Text of the Manual ..... v
Code Examples ..... vi
**Your Comments Are Welcome** ..... vi

**1 Understanding Replication**

**What Is Replication?** ..... 1-2
Basic Replication ..... 1-2
Advanced Replication ..... 1-3
**Basic Replication Concepts** ..... 1-4
Uses of Basic Replication ..... 1-4
Read-Only Table Snapshots ..... 1-6
Snapshot Refreshes ..... 1-9
Other Basic Replication Options ..... 1-11
**Advanced Replication Concepts** ..... 1-12
Uses for Advanced Replication ..... 1-13

Advanced Replication Configurations ..... 1-14
Advanced Replication and Oracle's Replication Manager ..... 1-18
Replication Objects, Groups, Sites, and Catalogs ..... 1-18
Oracle's Advanced Replication Architecture ..... 1-20
Replication Administrators, Propagators, and Receivers ..... 1-23
Replication Conflicts ..... 1-24
Unique Advanced Replication Options ..... 1-28

## 2 Using Basic Replication

**Quick Start: Building a Basic Replication Environment** ..... 2-2
A Simple Example ..... 2-2
**Preparing a Database for Snapshots** ..... 2-4
Necessary Schemas ..... 2-4
Necessary Database Links ..... 2-5
Planning for Snapshot Refreshes ..... 2-5
**Creating Snapshot Logs** ..... 2-6
Snapshot Log Names ..... 2-8
Required Privileges ..... 2-8
Timing of Snapshot Log Creation ..... 2-8
Special Requirements for Primary Key Snapshots ..... 2-8
Snapshot Log Storage Parameters ..... 2-8
Special Requirements for Subquery Snapshots ..... 2-9
Internal Mechanisms of Snapshot Log Creation ..... 2-10
**Creating Simple Snapshots** ..... 2-10
Snapshot Names ..... 2-11
Required Privileges ..... 2-11
Special Requirements for Primary Key Snapshots ..... 2-12
Snapshot Refresh Settings ..... 2-12
A Snapshot's Defining Query ..... 2-12
Snapshot Storage Settings ..... 2-14
Clustered Snapshots ..... 2-15
Data Load Options ..... 2-15
**Creating Snapshots with Subqueries** ..... 2-16
Advanced Subsetting with Subqueries ..... 2-16
Restrictions for Snapshots for Subquery ..... 2-23

**Creating Refresh Groups** ... 2-24
Refresh Settings ... 2-26
Refresh Types ... 2-27
Rollback Segment Setting ... 2-27
**Managing a Basic Replication Environment** ... 2-27
Managing Snapshot Logs ... 2-28
Managing Read-Only Snapshots ... 2-34
Managing Snapshot Refreshes and Refresh Groups ... 2-37
**Tuning Performance for Snapshots** ... 2-40
Indexing Snapshots ... 2-40
Tuning Subquery Snapshots ... 2-41
**Other Basic Replication Options** ... 2-41
Complex Snapshots ... 2-41
ROWID Snapshots ... 2-43
Individual Snapshot Refreshes ... 2-44
**Monitoring Basic Replication Environments** ... 2-44

## 3 Using Multimaster Replication

**Quick Start: Building a Multimaster Replication Environment** ... 3-2
A Simple Example ... 3-2
**Preparing for Multimaster Replication** ... 3-4
The Replication Setup Wizard ... 3-4
Starting SNP Background Processes ... 3-8
**Managing Scheduled Links** ... 3-9
Creating a Scheduled Link ... 3-9
Editing a Scheduled Link ... 3-11
Viewing the Status of a Scheduled Link ... 3-12
Deleting a Scheduled Link ... 3-12
**Purging a Site's Deferred Transaction Queue** ... 3-12
Specifying a Site's Purge Schedule ... 3-14
Manually Purging a Site's Deferred Transaction Queue ... 3-14
**Managing Master Groups** ... 3-15
Creating a Master Group ... 3-15
Deleting a Master Group ... 3-17
Suspending Replication Activity for a Master Group ... 3-17

Resuming Replication Activity for a Master Group ..... 3-19
Adding Objects to a Master Group ..... 3-19
Altering Objects in a Master Group ..... 3-23
Removing Objects from a Master Group ..... 3-24
Adding a Master Site to a Master Group ..... 3-24
Removing a Master Site from a Master Group ..... 3-25
Generating Replication Support for Master Group Objects ..... 3-26
Viewing Information About Master Groups ..... 3-29
Other Master Site Administration Issues ..... 3-31
**Advanced Multimaster Replication Options** ..... 3-31
Planning for Parallel Propagation ..... 3-31
Understanding Replication Protection Mechanisms ..... 3-33

## 4 Using Snapshot Site Replication

**Quick Start: Adding a Snapshot Site to an Advanced Replication System** ..... 4-2
A Simple Example ..... 4-2
**Preparing for Snapshot Site Replication** ..... 4-5
The Replication Setup Wizard ..... 4-5
Planning Scheduled Links and Snapshot Refreshes ..... 4-9
Starting SNP Background Processes ..... 4-10
**Managing Snapshot Logs** ..... 4-11
Creating a Snapshot Log ..... 4-11
Altering a Snapshot Log ..... 4-11
Deleting a Snapshot Log ..... 4-12
**Managing Snapshot Groups** ..... 4-12
Creating a Snapshot Group ..... 4-12
Registering a Snapshot Group at its Master Site ..... 4-14
Unregistering a Snapshot Group at its Master Site ..... 4-14
Adding Objects to a Snapshot Group ..... 4-14
Altering Objects in a Snapshot Group ..... 4-15
Deleting Objects from a Snapshot Group ..... 4-15
Editing a Snapshot Group ..... 4-15
Deleting a Snapshot Group ..... 4-16
**Managing Snapshots** ..... 4-16
Creating a Snapshot ..... 4-17

Creating Snapshots with Subqueries ..... 4-20
Regenerating Replication Support for an Updatable Snapshot ..... 4-20
Altering a Snapshot ..... 4-20
Deleting a Snapshot ..... 4-21
**Managing Refresh Groups** ..... 4-22
Creating a Refresh Group ..... 4-22
Adding Snapshots to a Refresh Group ..... 4-22
Deleting Snapshots from a Refresh Group ..... 4-23
Changing Refresh Settings for a Snapshot Group ..... 4-23
Manually Refreshing a Group of Snapshots ..... 4-23
Deleting a Refresh Group ..... 4-23
**Other Snapshot Site Administration Issues** ..... 4-24
**Data Dictionary Views** ..... 4-24

## 5 Conflict Resolution

**Introduction to Replication Conflicts** ..... 5-2
Understanding Your Data and Application Requirements ..... 5-2
Types of Replication Conflicts ..... 5-3
Avoiding Conflicts ..... 5-4
Conflict Detection at Master Sites ..... 5-5
Conflict Resolution ..... 5-6
Conflict Resolution Methods ..... 5-8
**Overview of Conflict Resolution Configuration** ..... 5-12
Design and Preparation Guidelines for Conflict Resolution ..... 5-12
Implementing Conflict Resolution ..... 5-13
**Configuring Update Conflict Resolution** ..... 5-13
Creating a Column Group ..... 5-13
Adding and Removing Columns in a Column Group ..... 5-14
Dropping a Column Group ..... 5-14
Managing a Group's Update Conflict Resolution Methods ..... 5-15
Prebuilt Update Conflict Resolution Methods ..... 5-16
Using Priority Groups for Update Conflict Resolution ..... 5-22
Using Site Priority for Update Conflict Resolution ..... 5-28
Sample Timestamp and Site Maintenance Trigger ..... 5-31
**Configuring Uniqueness Conflict Resolution** ..... 5-33

Assigning a Uniqueness Conflict Resolution Method ..... 5-33
Removing a Uniqueness Conflict Resolution Method ..... 5-33
Prebuilt Uniqueness Resolution Methods ..... 5-34
**Configuring Delete Conflict Resolution** ..... 5-36
Assigning a Delete Conflict Resolution Method ..... 5-36
Removing a Delete Conflict Resolution Method ..... 5-36
**Guaranteeing Data Convergence** ..... 5-37
Avoiding Ordering Conflicts ..... 5-38
**Minimizing Data Propagation for Update Conflict Resolution** ..... 5-40
Minimizing Communication Examples ..... 5-41
Further Reducing Data Propagation ..... 5-43
**User-Defined Conflict Resolution Methods** ..... 5-46
Conflict Resolution Method Parameters ..... 5-46
Resolving Update Conflicts ..... 5-47
Resolving Uniqueness Conflicts ..... 5-47
Resolving Delete Conflicts ..... 5-48
Restrictions ..... 5-48
Example User-Defined Conflict Resolution Method ..... 5-49
**User-Defined Conflict Notification Methods** ..... 5-50
Creating a Conflict Notification Log ..... 5-51
Creating a Conflict Notification Package ..... 5-51
**Viewing Conflict Resolution Information** ..... 5-54

## 6 Administering a Replicated Environment

**Advanced Management of Master and Snapshot Groups** ..... 6-2
Executing DDL Within a Master Group ..... 6-2
Validating a Master Group ..... 6-2
Relocating a Master Group's Definition Site ..... 6-3
Changing a Snapshot Group's Master Site ..... 6-3
**Monitoring an Advanced Replication System** ..... 6-4
Managing Administration Requests ..... 6-4
Managing Deferred Transactions ..... 6-8
Managing Error Transactions ..... 6-10
Managing Local Jobs ..... 6-12
**Database Backup and Recovery in Replication Systems** ..... 6-15

Performing Checks on Imported Data ..... 6-15
**Auditing Successful Conflict Resolution** ..... 6-16
Gathering Conflict Resolution Statistics ..... 6-16
Viewing Conflict Resolution Statistics ..... 6-16
Canceling Conflict Resolution Statistics ..... 6-17
Deleting Statistics Information ..... 6-17
**Determining Differences Between Replicated Tables** ..... 6-17
DIFFERENCES ..... 6-17
RECTIFY ..... 6-18
**Troubleshooting Common Problems** ..... 6-22
Diagnosing Problems with Database Links ..... 6-22
Diagnosing Problems with Master Sites ..... 6-22
Diagnosing Problems with the Deferred Transaction Queue ..... 6-25
Diagnosing Problems with Snapshots ..... 6-26
**Updating The Comments Fields in Views** ..... 6-28

## 7 Advanced Techniques

**Using Procedural Replication** ..... 7-2
Restrictions on Procedural Replication ..... 7-2
Serialization of Transactions ..... 7-3
Generating Support for Replicated Procedures ..... 7-3
**Using Synchronous Data Propagation** ..... 7-6
Understanding Synchronous Data Propagation ..... 7-6
Adding New Sites to an Advanced Replication Environment ..... 7-8
Altering a Master Site's Data Propagation Mode ..... 7-10
**Designing for Survivability** ..... 7-11
Oracle Parallel Server versus Advanced Replication ..... 7-12
Designing for Survivability ..... 7-13
Implementing a Survivable System ..... 7-14
**Snapshot Cloning and Offline Instantiation** ..... 7-14
Snapshot Cloning for Basic Replication Environments ..... 7-15
Offline Instantiation of a Master Site in an Advanced Replication System ..... 7-16
Offline Instantiation of a Snapshot Site in an Advanced Replication System ..... 7-17
**Security Setup for Multimaster Replication** ..... 7-18
**Security Setup for Snapshot Replication** ..... 7-22

**Alternative Security Setup for an Advanced Replication Snapshot Site** ..... 7-24
**Avoiding Delete Conflicts** ..... 7-25
**Using Dynamic Ownership Conflict Avoidance** ..... 7-27
Workflow ..... 7-27
Token Passing ..... 7-28
Locating the Owner of a Row ..... 7-31
Obtaining Ownership ..... 7-31
Applying the Change ..... 7-32
**Modifying Tables without Replicating the Modifications** ..... 7-32
Disabling the Advanced Replication Facility ..... 7-33
Re-enabling the Advanced Replication Facility ..... 7-34
Triggers and Replication ..... 7-34
Enabling / Disabling Replication for Snapshots ..... 7-35
..... 7-35

## 8 Using Deferred Transactions

**Listing Information about Deferred Transactions** ..... 8-2
**Creating a Deferred Transaction** ..... 8-3
Security ..... 8-3
Specifying a Destination ..... 8-3
Initiating a Deferred Transaction ..... 8-4
Deferring a Remote Procedure Call ..... 8-5
Queuing a Parameter Value for a Deferred Call ..... 8-5
Adding a Destination to the DEFDEFAULTDEST View ..... 8-6
Removing a Destination from the DEFDEFAULTDEST View ..... 8-6
Executing a Deferred Transaction ..... 8-7
**LOB Storage** ..... 8-7
DEFLOB View of Storage for RPC ..... 8-7

## 9 Replication Management API Reference

**Packages** ..... 9-2
**Examples of Using Oracle's Replication Management API** ..... 9-2
Prerequisites to Consider ..... 9-3
Replication Manager and Oracle Replication Management API ..... 9-3
**DBMS_DEFER Package** ..... 9-4

**DBMS_DEFER.CALL** ... 9-5
Purpose ... 9-5
Syntax ... 9-5
**DBMS_DEFER.COMMIT_WORK** ... 9-7
Purpose ... 9-7
Syntax ... 9-7
**DBMS_DEFER.*datatype*_ARG** ... 9-8
Purpose ... 9-8
**DBMS_DEFER.TRANSACTION** ... 9-9
Purpose ... 9-9
Syntax ... 9-9
**DBMS_DEFER_QUERY Package** ... 9-10
**DBMS_DEFER_QUERY.GET_ARG_FORM** ... 9-11
Purpose ... 9-11
Syntax ... 9-11
**DBMS_DEFER_QUERY.GET_ARG_TYPE** ... 9-12
Purpose ... 9-12
Syntax ... 9-12
**DBMS_DEFER_QUERY.GET_CALL_ARGS** ... 9-14
Purpose ... 9-14
Syntax ... 9-14
**DBMS_DEFER_QUERY.GET_*datatype*_ARG** ... 9-16
Purpose ... 9-16
Syntax ... 9-16
**DBMS_DEFER_SYS Package** ... 9-18
**DBMS_DEFER_SYS.ADD_DEFAULT_DEST** ... 9-19
Purpose ... 9-19
Syntax ... 9-19
**DBMS_DEFER_SYS.DELETE_DEFAULT_DEST** ... 9-20
Purpose ... 9-20
Syntax ... 9-20
**DBMS_DEFER_SYS.DELETE_DEF_DESTINATION** ... 9-21
Purpose ... 9-21
Syntax ... 9-21
**DBMS_DEFER_SYS.DELETE_ERROR** ... 9-22

Purpose ... 9-22
Syntax ... 9-22
**DBMS_DEFER_SYS.DELETE_TRAN** ... 9-23
Purpose ... 9-23
Syntax ... 9-23
**DBMS_DEFER_SYS.DISABLED** ... 9-24
Purpose ... 9-24
Syntax ... 9-24
**DBMS_DEFER_SYS.EXCLUDE_PUSH** ... 9-25
Purpose ... 9-25
Syntax ... 9-25
**DBMS_DEFER_SYS.EXECUTE_ERROR** ... 9-26
Purpose ... 9-26
Syntax ... 9-26
**DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER** ... 9-27
Purpose ... 9-27
Syntax ... 9-27
**DBMS_DEFER_SYS.PURGE** ... 9-28
Purpose ... 9-28
Syntax ... 9-28
**DBMS_DEFER_SYS.PUSH** ... 9-31
Purpose ... 9-31
Syntax ... 9-31
**DBMS_DEFER_SYS.REGISTER_PROPAGATOR** ... 9-34
Purpose ... 9-34
Syntax ... 9-34
**DBMS_DEFER_SYS.SCHEDULE_PURGE** ... 9-35
Purpose ... 9-35
Syntax ... 9-35
**DBMS_DEFER_SYS.SCHEDULE_PUSH** ... 9-37
Purpose ... 9-37
Syntax ... 9-37
**DBMS_DEFER_SYS.SET_DISABLED** ... 9-40
Purpose ... 9-40
Syntax ... 9-40

**DBMS_DEFER_SYS.UNREGISTER_PROPAGATOR** ..... 9-41
Purpose ..... 9-41
Syntax ..... 9-41
**DBMS_DEFER_SYS.UNSCHEDULE_PURGE** ..... 9-42
Purpose ..... 9-42
Syntax ..... 9-42
**DBMS_DEFER_SYS.UNSCHEDULE_PUSH** ..... 9-43
Purpose ..... 9-43
Syntax ..... 9-43
**DBMS_OFFLINE_OG Package** ..... 9-44
**DBMS_OFFLINE_OG.BEGIN_INSTANTIATION** ..... 9-45
Purpose ..... 9-45
Syntax ..... 9-45
**DBMS_OFFLINE_OG.BEGIN_LOAD** ..... 9-46
Purpose ..... 9-46
Syntax ..... 9-46
**DBMS_OFFLINE_OG.END_INSTANTIATION** ..... 9-47
Purpose ..... 9-47
Syntax ..... 9-47
**DBMS_OFFLINE_OG.END_LOAD** ..... 9-48
Purpose ..... 9-48
Syntax ..... 9-48
**DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS** ..... 9-49
Purpose ..... 9-49
Syntax ..... 9-49
**DBMS_OFFLINE_SNAPSHOT Package** ..... 9-50
**DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD** ..... 9-51
Purpose ..... 9-51
Syntax ..... 9-51
**DBMS_OFFLINE_SNAPSHOT.END_LOAD** ..... 9-53
Purpose ..... 9-53
Syntax ..... 9-53
**DBMS_RECTIFIER_DIFF Package** ..... 9-54
**DBMS_RECTIFIER_DIFF.DIFFERENCES** ..... 9-55
Purpose ..... 9-55

Syntax ..... 9-55
Restrictions ..... 9-58
**DBMS_RECTIFIER_DIFF.RECTIFY** ..... 9-59
Purpose ..... 9-59
Syntax ..... 9-59
**DBMS_REFRESH Package** ..... 9-62
**DBMS_REFRESH.ADD** ..... 9-63
Purpose ..... 9-63
Syntax ..... 9-63
**DBMS_REFRESH.CHANGE** ..... 9-64
Purpose ..... 9-64
Syntax ..... 9-64
**DBMS_REFRESH.DESTROY** ..... 9-66
Purpose ..... 9-66
Syntax ..... 9-66
**DBMS_REFRESH.MAKE** ..... 9-67
Purpose ..... 9-67
Syntax ..... 9-67
**DBMS_REFRESH.REFRESH** ..... 9-70
Purpose ..... 9-70
Syntax ..... 9-70
**DBMS_REFRESH.SUBTRACT** ..... 9-71
Purpose ..... 9-71
Syntax ..... 9-71
**DBMS_REPCAT Package** ..... 9-72
**DBMS_REPCAT.ADD_GROUPED_COLUMN** ..... 9-75
Purpose ..... 9-75
Syntax ..... 9-75
**DBMS_REPCAT.ADD_MASTER_DATABASE** ..... 9-77
Purpose ..... 9-77
Syntax ..... 9-77
**DBMS_REPCAT.ADD_PRIORITY_*datatype*** ..... 9-79
Purpose ..... 9-79
Syntax ..... 9-79
**DBMS_REPCAT.ADD_SITE_PRIORITY_SITE** ..... 9-81

Purpose ..... 9-81
Syntax ..... 9-81
**DBMS_REPCAT.ADD_*conflicttype*_RESOLUTION** ..... 9-83
Purpose ..... 9-83
Syntax ..... 9-83
**DBMS_REPCAT.ALTER_MASTER_PROPAGATION** ..... 9-87
Purpose ..... 9-87
Syntax ..... 9-87
**DBMS_REPCAT.ALTER_MASTER_REPOBJECT** ..... 9-89
Purpose ..... 9-89
Syntax ..... 9-89
**DBMS_REPCAT.ALTER_PRIORITY** ..... 9-91
Purpose ..... 9-91
Syntax ..... 9-91
**DBMS_REPCAT.ALTER_PRIORITY_*datatype*** ..... 9-93
Purpose ..... 9-93
Syntax ..... 9-93
**DBMS_REPCAT.ALTER_SITE_PRIORITY** ..... 9-95
Purpose ..... 9-95
Syntax ..... 9-95
**DBMS_REPCAT.ALTER_SITE_PRIORITY_SITE** ..... 9-97
Purpose ..... 9-97
Syntax ..... 9-97
**DBMS_REPCAT.ALTER_SNAPSHOT_PROPAGATION** ..... 9-99
Purpose ..... 9-99
Syntax ..... 9-99
**DBMS_REPCAT.CANCEL_STATISTICS** ..... 9-100
Purpose ..... 9-100
Syntax ..... 9-100
**DBMS_REPCAT.COMMENT_ON_COLUMN_GROUP** ..... 9-101
Purpose ..... 9-101
Syntax ..... 9-101
**DBMS_REPCAT.COMMENT_ON_PRIORITY_GROUP/** ..... 9-102
**DBMS_REPCAT.COMMENT_ON_SITE_PRIORITY** ..... 9-102
Purpose ..... 9-102

Syntax ..... 9-102
**DBMS_REPCAT.COMMENT_ON_REPGROUP** ..... 9-104
Purpose ..... 9-104
Syntax ..... 9-104
**DBMS_REPCAT.COMMENT_ON_REPSITES** ..... 9-105
Purpose ..... 9-105
Syntax ..... 9-105
**DBMS_REPCAT.COMMENT_ON_REPOBJECT** ..... 9-106
Purpose ..... 9-106
Syntax ..... 9-106
**DBMS_REPCAT.COMMENT_ON_*conflicttype*_RESOLUTION** ..... 9-107
Purpose ..... 9-107
Syntax ..... 9-107
**DBMS_REPCAT.CREATE_MASTER_REPGROUP** ..... 9-109
Purpose ..... 9-109
Syntax ..... 9-109
**DBMS_REPCAT.CREATE_MASTER_REPOBJECT** ..... 9-110
Purpose ..... 9-110
Syntax ..... 9-110
**DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP** ..... 9-113
Purpose ..... 9-113
Syntax ..... 9-113
**DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT** ..... 9-115
Purpose ..... 9-115
Syntax ..... 9-115
**DBMS_REPCAT.DEFINE_COLUMN_GROUP** ..... 9-118
Purpose ..... 9-118
Syntax ..... 9-118
**DBMS_REPCAT.DEFINE_PRIORITY_GROUP** ..... 9-119
Purpose ..... 9-119
Syntax ..... 9-119
**DBMS_REPCAT.DEFINE_SITE_PRIORITY** ..... 9-121
Purpose ..... 9-121
Syntax ..... 9-121
**DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN** ..... 9-122

Purpose ..... 9-122
Syntax ..... 9-122
**DBMS_REPCAT.DROP_COLUMN_GROUP** ..... 9-123
Purpose ..... 9-123
Syntax ..... 9-123
**DBMS_REPCAT.DROP_GROUPED_COLUMN** ..... 9-124
Purpose ..... 9-124
Syntax ..... 9-124
**DBMS_REPCAT.DROP_MASTER_REPGROUP** ..... 9-126
Purpose ..... 9-126
Syntax ..... 9-126
**DBMS_REPCAT.DROP_MASTER_REPOBJECT** ..... 9-128
Purpose ..... 9-128
Syntax ..... 9-128
**DBMS_REPCAT.DROP_PRIORITY** ..... 9-129
Purpose ..... 9-129
Syntax ..... 9-129
**DBMS_REPCAT.DROP_PRIORITY_GROUP** ..... 9-130
Purpose ..... 9-130
Syntax ..... 9-130
**DBMS_REPCAT.DROP_PRIORITY_*datatype*** ..... 9-131
Purpose ..... 9-131
Syntax ..... 9-131
**DBMS_REPCAT.DROP_SITE_PRIORITY** ..... 9-133
Purpose ..... 9-133
Syntax ..... 9-133
**DBMS_REPCAT.DROP_SITE_PRIORITY_SITE** ..... 9-134
Purpose ..... 9-134
Syntax ..... 9-134
**DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP** ..... 9-135
Purpose ..... 9-135
Syntax ..... 9-135
**DBMS_REPCAT.DROP_SNAPSHOT_REPOBJECT** ..... 9-136
Purpose ..... 9-136
Syntax ..... 9-136

**DBMS_REPCAT.DROP_*conflicttype*_RESOLUTION** ..... 9-137
Purpose ..... 9-137
Syntax ..... 9-137
**DBMS_REPCAT.EXECUTE_DDL** ..... 9-139
Purpose ..... 9-139
Syntax ..... 9-139
**DBMS_REPCAT.GENERATE_REPLICATION_PACKAGE** ..... 9-141
Purpose ..... 9-141
Syntax ..... 9-141
**DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT** ..... 9-142
Purpose ..... 9-142
Syntax ..... 9-142
**DBMS_REPCAT.GENERATE_REPLICATION_TRIGGER** ..... 9-144
Purpose ..... 9-144
Syntax ..... 9-144
Altering Propagation Mode ..... 9-146
**DBMS_REPCAT.GENERATE_SNAPSHOT_SUPPORT** ..... 9-147
PURPOSE ..... 9-147
SYNTAX ..... 9-147
**DBMS_REPCAT.MAKE_COLUMN_GROUP** ..... 9-149
Purpose ..... 9-149
Syntax ..... 9-149
**DBMS_REPCAT.PURGE_MASTER_LOG** ..... 9-151
Purpose ..... 9-151
Syntax ..... 9-151
**DBMS_REPCAT.PURGE_STATISTICS** ..... 9-152
Purpose ..... 9-152
Syntax ..... 9-152
**DBMS_REPCAT.REFRESH_SNAPSHOT_REPGROUP** ..... 9-153
Purpose ..... 9-153
Syntax ..... 9-153
**DBMS_REPCAT.REGISTER_SNAPSHOT_REPGROUP** ..... 9-155
Purpose ..... 9-155
Syntax ..... 9-155
**DBMS_REPCAT.REGISTER_STATISTICS** ..... 9-156

Purpose ..... 9-156
Syntax ..... 9-156
**DBMS_REPCAT.RELOCATE_MASTERDEF** ..... 9-157
Purpose ..... 9-157
Syntax ..... 9-157
Usage Notes ..... 9-158
**DBMS_REPCAT.REMOVE_MASTER_DATABASES** ..... 9-159
Purpose ..... 9-159
Syntax ..... 9-159
**DBMS_REPCAT.REPCAT_IMPORT_CHECK** ..... 9-161
Purpose ..... 9-161
Syntax ..... 9-161
**DBMS_REPCAT.RESUME_MASTER_ACTIVITY** ..... 9-162
Purpose ..... 9-162
Syntax ..... 9-162
**DBMS_REPCAT.SEND_AND_COMPARE_OLD_VALUES** ..... 9-163
Purpose ..... 9-163
Syntax ..... 9-163
**DBMS_REPCAT.SET_COLUMNS** ..... 9-165
Purpose ..... 9-165
Syntax ..... 9-165
**DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY** ..... 9-167
Purpose ..... 9-167
Syntax ..... 9-167
**DBMS_REPCAT.SWITCH_SNAPSHOT_MASTER** ..... 9-168
Purpose ..... 9-168
Syntax ..... 9-168
**DBMS_REPCAT.UNREGISTER_SNAPSHOT_REPGROUP** ..... 9-169
Purpose ..... 9-169
Syntax ..... 9-169
**DBMS_REPCAT.VALIDATE** ..... 9-170
Purpose ..... 9-170
Syntax ..... 9-170
**DBMS_REPCAT.WAIT_MASTER_LOG** ..... 9-173
Purpose ..... 9-173

Syntax ..... 9-173
**DBMS_REPCAT_ADMIN Package** ..... 9-174
**DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA** ..... 9-175
Purpose ..... 9-175
Syntax ..... 9-175
**DBMS_REPCAT_ADMIN.GRANT_ADMIN_SCHEMA** ..... 9-176
Purpose ..... 9-176
Syntax ..... 9-176
**DBMS_REPCAT_ADMIN.REVOKE_ADMIN_ANY_SCHEMA** ..... 9-177
Purpose ..... 9-177
Syntax ..... 9-177
**DBMS_REPCAT_ADMIN.REVOKE_ADMIN_SCHEMA** ..... 9-178
Purpose ..... 9-178
Syntax ..... 9-178
**DBMS_REPCAT_AUTH Package** ..... 9-179
**DBMS_REPCAT_AUTH.GRANT_SURROGATE_REPCAT** ..... 9-180
Purpose ..... 9-180
Syntax ..... 9-180
**DBMS_REPCAT_AUTH.REVOKE_SURROGATE_REPCAT** ..... 9-181
Purpose ..... 9-181
Syntax ..... 9-181
**DBMS_REPUTIL Package** ..... 9-182
**DBMS_REPUTIL.REPLICATION_OFF** ..... 9-183
Purpose ..... 9-183
Syntax ..... 9-183
**DBMS_REPUTIL.REPLICATION_ON** ..... 9-184
Purpose ..... 9-184
Syntax ..... 9-184
**DBMS_SNAPSHOT Package** ..... 9-185
**DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION** ..... 9-186
Purpose ..... 9-186
Syntax ..... 9-186
**DBMS_SNAPSHOT.END_TABLE_REORGANIZATION** ..... 9-187
Purpose ..... 9-187
Syntax ..... 9-187

**DBMS_SNAPSHOT.I_AM_A_REFRESH** ..... 9-188
Purpose ..... 9-188
Syntax ..... 9-188
**DBMS_SNAPSHOT.PURGE_LOG** ..... 9-189
Purpose ..... 9-189
Syntax ..... 9-189
**DBMS_SNAPSHOT.REFRESH** ..... 9-190
Purpose ..... 9-190
Syntax ..... 9-190
**DBMS_SNAPSHOT.REGISTER_SNAPSHOT** ..... 9-193
Purpose ..... 9-193
Syntax ..... 9-193
**DBMS_SNAPSHOT.SET_I_AM_A_REFRESH** ..... 9-195
Purpose ..... 9-195
Syntax ..... 9-195
**DBMS_SNAPSHOT.UNREGISTER_SNAPSHOT** ..... 9-196
Purpose ..... 9-196
Syntax ..... 9-196
**Package Variables** ..... 9-197

## 10 Data Dictionary Views

**Replication Catalog Views** ..... 10-2
REPGROUP View ..... 10-3
REPCATLOG View ..... 10-5
REPCOLUMN View ..... 10-6
REPCOLUMN_GROUP View ..... 10-6
REPCONFLICT View ..... 10-7
REPDDL View ..... 10-7
REPGENERATED View ..... 10-8
REPGROUPED_COLUMN View ..... 10-9
REPKEY_COLUMNS View ..... 10-9
REPOBJECT View ..... 10-10
REPPARAMETER_COLUMN View ..... 10-11
REPPRIORITY View ..... 10-12
REPPRIORITY_GROUP View ..... 10-13

REPPROP View ..... 10-13
REPRESOLUTION View ..... 10-14
REPRESOL_STATS_CONTROL View ..... 10-15
REPRESOLUTION_METHOD View ..... 10-15
REPRESOLUTION_STATISTICS View ..... 10-16
REPSITES View ..... 10-17
REPGENOBJECTS View ..... 10-18
**Deferred Transaction Views** ..... 10-19
DEFCALL View ..... 10-20
DEFCALLDEST View ..... 10-20
DEFDEFAULTDEST View ..... 10-21
DEFERRCOUNT View ..... 10-21
DEFERROR View ..... 10-22
DEFLOB View ..... 10-22
DEFPROPAGATOR View ..... 10-23
DEFSCHEDULE View ..... 10-23
DEFTRAN View ..... 10-24
DEFTRANDEST View ..... 10-24
**Snapshots and Snapshot Refresh Group Views** ..... 10-25
SNAPSHOTS View ..... 10-26
REGISTERED_SNAPSHOTS View ..... 10-27
SNAPSHOTS_LOGS View ..... 10-28
SNAPSHOT_REFRESH_TIMES View ..... 10-29
REFRESH View ..... 10-30
REFRESH_CHILDREN View ..... 10-31

## A New Features

**Performance Enhancements** ..... A-2
Parallel Propagation of Deferred Transactions ..... A-2
Internalized Replication Triggers ..... A-2
Reduced Data Propagation ..... A-2
**Data Subsetting Based on Subqueries** ..... A-2
**Large Object Datatypes (LOBs) Support** ..... A-3
**Improved Management and Ease of Use** ..... A-3
Fine grained Quiesce ..... A-3

Primary Key Snapshots ... A-3
Snapshot Registration at Master Sites ... A-3
Reorganizing Tables With Capability of Fast Refresh ... A-4
Support for Offline Instantiation ... A-4
Deferred Constraints for Updatable Snapshots ... A-4
Validate Procedure ... A-4
Partitioned Tables and Indexes ... A-4
**Enhanced, System-Based Security Model** ... A-4
**New Replication Manager Features** ... A-5

## B Migration and Compatibility

**Migration Overview** ... B-2
**Migrating All Sites at Once** ... B-2
**Incremental Migration** ... B-4
Preparing Oracle7 Master Sites for Incremental Migration ... B-5
Incremental Migration of Snapshot Sites ... B-5
Incremental Migration of Master Sites ... B-6
**Migration Using Export/ Import** ... B-9
**Upgrading to Primary Key Snapshots** ... B-10
Primary Key Snapshots Conversion at Master Site(s) ... B-10
Primary Key Snapshot Conversion at Snapshot Site(s) ... B-11
**Features Requiring Migration to Oracle8** ... B-11
**Obsolete procedures** ... B-12

## Index

# Send Us Your Comments

***Oracle8 Replication*, Release 8.0**

**Part No. A58245-01**

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

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

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

- infodev@us.oracle.com
- FAX - 650.506.7200 Attn: Oracle8 Distributed Database Systems
- Postal service:
  Server Technologies Documentation Manager
  Oracle Corporation
  Oracle8 Documentation
  500 Oracle Parkway
  Redwood Shores, CA 94065
  USA

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

# Preface

This Preface contains the following topics:

- Overview of The Oracle8 Replication Manual.
- Audience.
- How The Oracle8 Replication Manual Is Organized.
- Conventions Used in This Manual.
- Your Comments Are Welcome.

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

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

# Overview of The Oracle8 Replication Manual

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

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

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

# Audience

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

## Knowledge Assumed of the Reader

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

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

- *Oracle8 Concepts*.
- *Oracle8 Administrator's Guide*.
- *Oracle8 Distributed Database Systems*.
- *PL/SQL User's Guide and Reference* (if you plan to use procedural replication).

# How The Oracle8 Replication Manual Is Organized

This manual contains 10 chapters and two appendices as described below.

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

**Chapter 2, "Using Basic Replication"**
Explains how to use read-only snapshots to perform basic primary site replication.

**Chapter 3, "Using Multimaster Replication"**
Describes how to create and maintain a multi-master replicated environment using the Oracle advanced replication facility.

**Chapter 4, "Using Snapshot Site Replication"**
Describes how to create and maintain snapshot sites using the Oracle advanced replication facility. This chapter also describes updatable snapshots.

**Chapter 5, "Conflict Resolution"**
Describes how to use Oracle-supplied conflict resolution methods to resolve conflicts resulting from dynamic or shared ownership of data in replicated environments.

**Chapter 6, "Administering a Replicated Environment"**
Describes how to detect and resolve unresolved replication errors as well as how to monitor successful conflict resolution.

**Chapter 7, "Advanced Techniques"**
Describes advanced replication techniques, including how to do the following: create your own conflict resolution routines, implement fail-over sites, implement token passing, use procedural replication, and manage deletes.

**Chapter 8, "Using Deferred Transactions"**
Describes how to build transactions for deferred execution at remote locations.

**Chapter 9, "Replication Management API Reference"**
Describes parameters for packaged procedures used to implement a replicated environment as well as exceptions these procedures might raise.

**Chapter 10, "Data Dictionary Views"**
Describes views of interest to users of deferred transactions, read-only snapshots, and the advanced replication facility.

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

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

# Conventions Used in This Manual

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

## Special Notes

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

**Note:** Indicates special or auxiliary information.

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

**Additional Information:** Indicates where to get more information.

## Text of the Manual

The following sections describe conventions used this manual.

### UPPERCASE Characters

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

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

### *Italicized* Characters

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

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

An example of a reference to another book is the following: “For more information, see the book *Oracle8 Tuning*.”

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

## Code Examples

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

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

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

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

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

# Your Comments Are Welcome

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

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

You can also e-mail comments to:

infodev@us.oracle.com

# 1

# Understanding Replication

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

- What Is Replication?
- Basic Replication Concepts.
- Advanced Replication Concepts.

**Notes:** If you are using Trusted Oracle, see the Trusted Oracle documentation for information about using replication in that environment.

Advanced Replication, as described in Chapter Three and the remainder of this book, is only available with the Oracle8 Enterprise Edition.

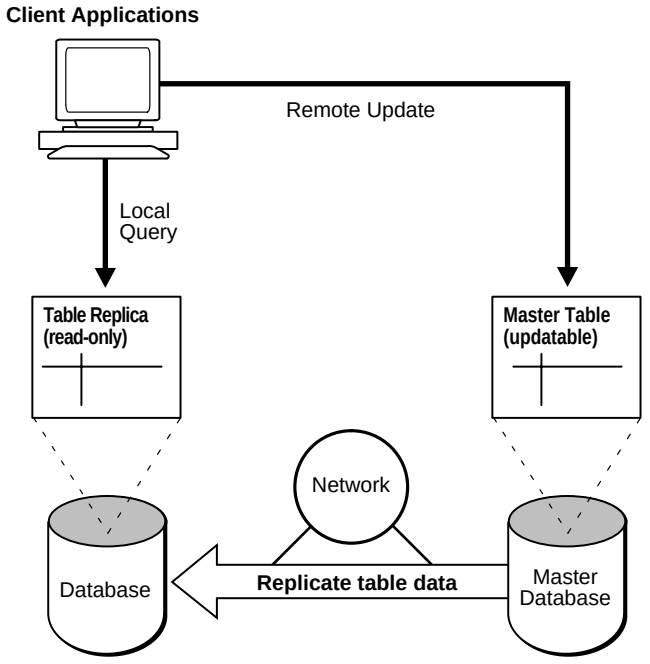
# What Is Replication?

*Replication* is the process of copying and maintaining schema objects in multiple databases that make up a distributed database system. Replication can improve the performance and protect the availability of applications because alternate data access options exist. For example, an application can normally access a local database rather than a remote server to minimize network traffic and achieve the best performance. Furthermore, the application can continue to function if the local server fails while other servers with replicated data remain accessible. Oracle supports two forms of replication: basic and advanced replication.

## Basic Replication

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

***Figure 1–1 Basic, Read-Only Replication***

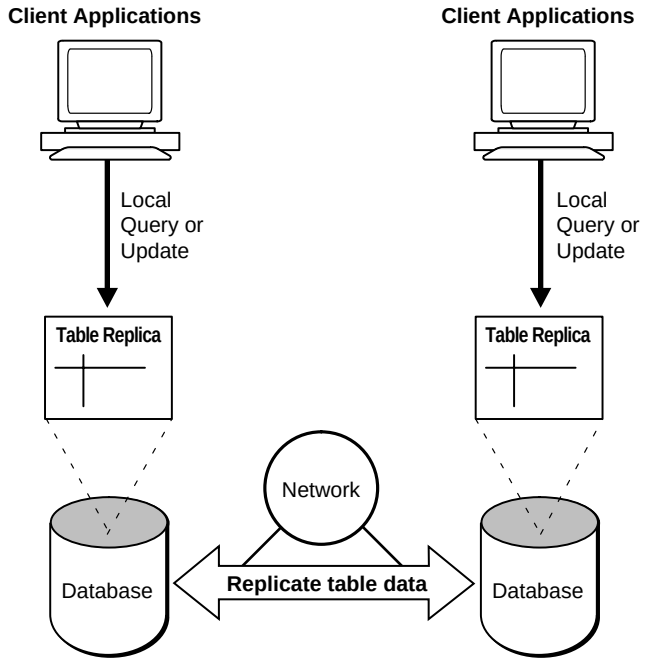


Oracle Server can support basic, read-only replication environments using *read-only table snapshots*. To learn more about basic replication and read-only snapshots, read "Basic Replication Concepts" on page 1-4.

## Advanced Replication

The Oracle *advanced replication* features extend the capabilities of basic read-only replication by allowing applications to update table replicas throughout a replicated database system. With advanced replication, data replicas anywhere in the system can provide both read and update access to a table's data. Participating Oracle database servers automatically work to converge the data of all table replicas, and ensure global transaction consistency and data integrity. Figure 1–2 illustrates advanced replication.

***Figure 1–2 Advanced Replication***




Oracle Server can support the requirements of advanced replication environments using several configurations. To learn more about advanced replication systems, read "Advanced Replication Concepts" on page 1-12.

# Basic Replication Concepts

Basic replication environments support applications requiring read-only access to table data originating from a primary site. The following sections explain the fundamental concepts of basic replication environments.

- Uses of Basic Replication.
- Read-Only Table Snapshots.
- Snapshot Refreshes.

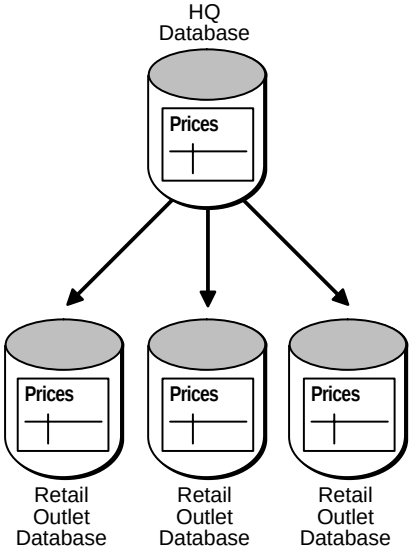
## Uses of Basic Replication

Basic, read-only data replication is useful for several types of applications as discussed under the following headings.

### Information Distribution

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

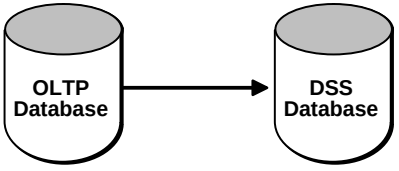
***Figure 1–3 Information Distribution***




### Information Off-Loading

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

***Figure 1–4 Information Off-Loading***



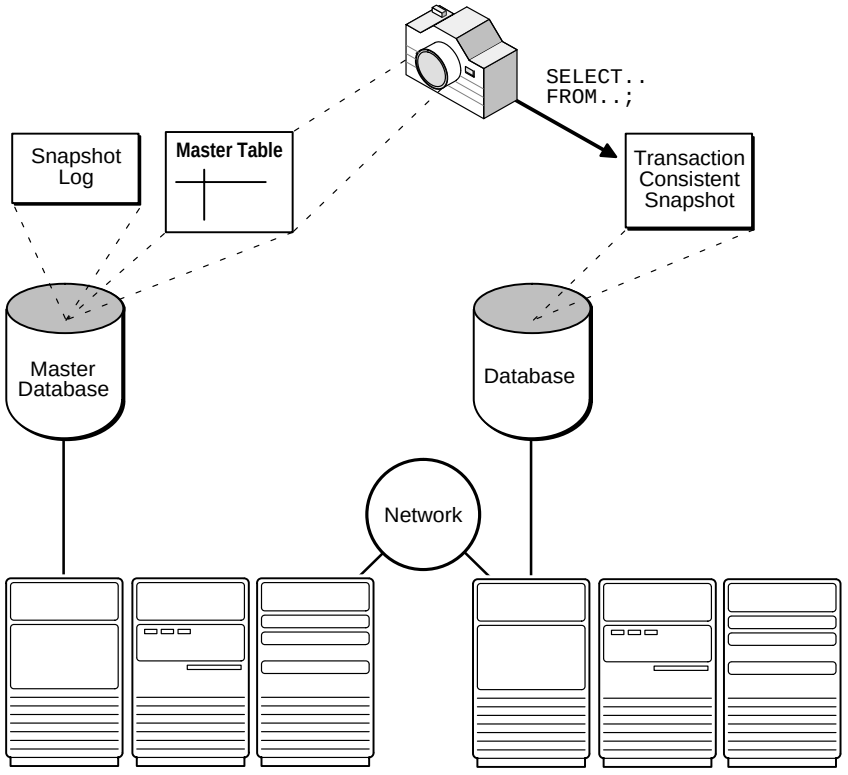

### Information Transport

Basic replication can be useful as an information transport mechanism. For example, basic replication can periodically move data from a production transaction processing database to a data warehouse.

## Read-Only Table Snapshots

A *read-only table snapshot* is a local copy of table data originating from one or more remote master tables. An application can query the data in a read-only table snapshot, but cannot insert, update, or delete rows in the snapshot. Figure 1–5 and the sections that follow explain more about read-only table snapshots and basic replication.

***Figure 1–5 Basic Replication Components: read-only snapshots, master tables, and snapshot logs***



### Read-Only Snapshot Architecture

Oracle supports basic data replication with its table snapshot mechanism. The following sections explain the architecture of simple read-only table snapshots.

**Note:** Oracle offers other basic replication features such as complex snapshots and ROWID snapshots for unique application requirements. To learn more about these special configurations, read "Other Basic Replication Options" on page 1-11.

**A Snapshot's Defining Query** The logical data structure of table snapshots is defined by a query that references data in one or more remote master tables. A snapshot's *defining query* determines what data the snapshot will contain.

A snapshot's defining query should be such that each row in the snapshot corresponds directly to a row or a part of a row in a single master table. Specifically, the defining query of a snapshot should not contain a distinct or aggregate function, a GROUP BY or CONNECT BY clause, join, restricted types of subqueries, or a set operation. The following example shows a simple table snapshot definition.

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

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

A snapshot's defining query can include restricted types of subqueries that reference multiple tables to filter rows from the snapshot's master table. A *subquery snapshot* can be used to create snapshots that "walk up" the many-to-one references from child to parent tables that may involve multiple levels. The following example creates a simple subquery snapshot.

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

**Internal Snapshot Objects** When you create a new, read-only table snapshot, Oracle creates several underlying database objects to support the snapshot.

- Oracle always creates a base table to store snapshot data. The name of a snapshot's base table is *SNAP$_snapshotname*.

- Oracle creates an index for the primary key in the snapshot base table. The name of the index is the name of the index used to enforce the corresponding PRIMARY KEY constraint at the master site.
- Oracle always creates a view for each snapshot. The view takes the name of the snapshot and provides read-only access to the snapshot.
- Oracle might create additional indexes for a snapshot defined with a subquery.

## Snapshot Refreshes

A snapshot's data does not necessarily match the current data of its master tables. A table snapshot is a transaction-consistent reflection of its master data as that data existed at a specific point in time. To keep a snapshot's data relatively current with the data of its master, Oracle periodically refreshes the snapshot. A *snapshot refresh* is an efficient batch operation that makes that snapshot reflect a more current state of its master.

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

To refresh snapshots, Oracle supports different types of refreshes, "complete" and "fast" snapshot refresh groups, as well as "manual" and "automatic" refreshes.

### Complete and Fast Refreshes

Oracle can refresh a snapshot using either a complete refresh or a fast refresh.

**Complete Refreshes** To perform a *complete refresh* of a snapshot, the server that manages the snapshot executes the snapshot's defining query. The result set of the query replaces the existing snapshot data to refresh the snapshot. Oracle can perform a complete refresh for any snapshot.

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

### Snapshot Logs

When a master table corresponds to one or more snapshots, create a snapshot log for the table so that fast refreshes of the snapshots are an option. A master table's *snapshot log* tracks fast refresh data for all corresponding snapshots—only one snapshot log is possible per master table. When a server performs a fast refresh for a snapshot, it uses data in its master table's snapshot log to refresh the snapshot efficiently. Oracle automatically purges specific refresh data from a snapshot log after all snapshots perform refreshes such that the log data is no longer needed.

When you create a snapshot log for a master table, Oracle creates an underlying table to support the snapshot log. A snapshot log's table holds the primary keys, timestamps, and optionally the ROWIDs of rows that applications update in the master table. A snapshot log can also contain filter columns to support fast refreshes for snapshots with subqueries. The name of a snapshot log's table is MLOG$_*master_table_name*.

### Snapshot Refresh Groups

To preserve referential integrity and transaction consistency among the snapshots of several related master tables, Oracle organizes and refreshes each snapshot as part of a *refresh group*. Oracle refreshes all snapshots in a group as a single operation. After refreshing all snapshots in a refresh group, the snapshot data in the group corresponds to the same transaction-consistent point in time.

### Automatic Snapshot Refreshes

When creating a snapshot refresh group, administrators usually configure the group so Oracle automatically refreshes its snapshots. Otherwise, administrators would have to manually refresh the group whenever necessary.

When configuring a refresh group for automatic refreshes, you must:

- Specify a refresh interval for the group.
- Configure the server managing the snapshots with one or more SNP background processes to periodically refresh snapshots that are due for refreshing.

**Automatic Refresh Intervals** When you create a snapshot refresh group, you can specify an automatic refresh interval for the group. When setting a group's refresh interval, consider the following characteristics:

- The dates or date expressions specifying the refresh interval must evaluate to a future point in time.
- The refresh interval must be greater than the length of time necessary to perform a refresh.
- Relative date expressions evaluate to a point in time relative to the most recent refresh date. If a network or system failure interferes with a scheduled group refresh, the evaluation of a relative date expression could change accordingly.
- Explicit date expressions evaluate to specific points in time, regardless of the most recent refresh date.

**Refresh Types** By default, Oracle attempts to perform a fast refresh of each snapshot in a refresh group. If Oracle cannot perform a fast refresh for an individual snapshot, for example when a master table has no snapshot log, the server performs a complete refresh for the snapshot.

**SNP Background Processes** Oracle Server's automatic snapshot refresh facility functions by using job queues to schedule the periodic execution of internal system procedures. Job queues require that at least one SNP background process be running. An *SNP background process* periodically checks the job queue and executes any outstanding jobs.

### Manual Snapshot Refreshes

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

## Other Basic Replication Options

Oracle supports additional basic replication features that can be useful in certain situations:

- Complex Snapshots.
- ROWID Snapshots.

### Complex Snapshots

When the defining query of a snapshot contains a distinct or aggregate function, a GROUP BY or CONNECT BY clause, join, restricted types of subqueries, or a set operation, the snapshot is a *complex snapshot*. The following example is a complex table snapshot definition.

```
CREATE SNAPSHOT scott.emp AS
 SELECT ename, dname
  FROM scott.emp@hq.acme.com a, scott.dept@hq.acme.com b
   WHERE a.deptno = b.deptno
   SORT BY dname
```

The primary disadvantage of a complex snapshot is that Oracle *cannot* perform a fast refresh of the snapshot; Oracle only performs complete refreshes of a complex

snapshot. Consequently, use of complex snapshots can affect network performance during complete snapshot refreshes.

### ROWID Snapshots

*Primary key snapshots*, as discussed in earlier sections of this chapter, are the default for Oracle. Oracle bases a primary key snapshot on the primary key of its master table. Because of this structure, you can:

- Reorganize the master tables of a snapshot without completing a full refresh of the snapshot.
- Create a snapshot with a defining query that includes a restricted type of subquery.

For backward compatibility only, Oracle also supports *ROWID snapshots* based on the physical row identifiers or "ROWIDs" of rows in the master table. Only use ROWID snapshots for snapshots of master tables in an Oracle7.3 database, and not when creating new snapshots of master tables in Oracle8 databases.

**Note:** To support a ROWID snapshot, Oracle creates an additional index on the snapshot's base table with the name *I_SNAP$_snapshotname*.

## Advanced Replication Concepts

In advanced replication environments, data replicas anywhere in the system can provide both read and update access to a table data. The following sections explain the principal concepts of an advanced replication system.

- Uses for Advanced Replication.
- Advanced Replication Configurations.
- Replication Objects, Groups, Sites, and Catalogs.
- Oracle's Advanced Replication Architecture.
- Replication Administrators, Propagators, and Receivers.
- Replication Conflicts.
- Unique Advanced Replication Options.

## Uses for Advanced Replication

Advanced data replication is useful for many types of application systems with special requirements.

### Disconnected Environments

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

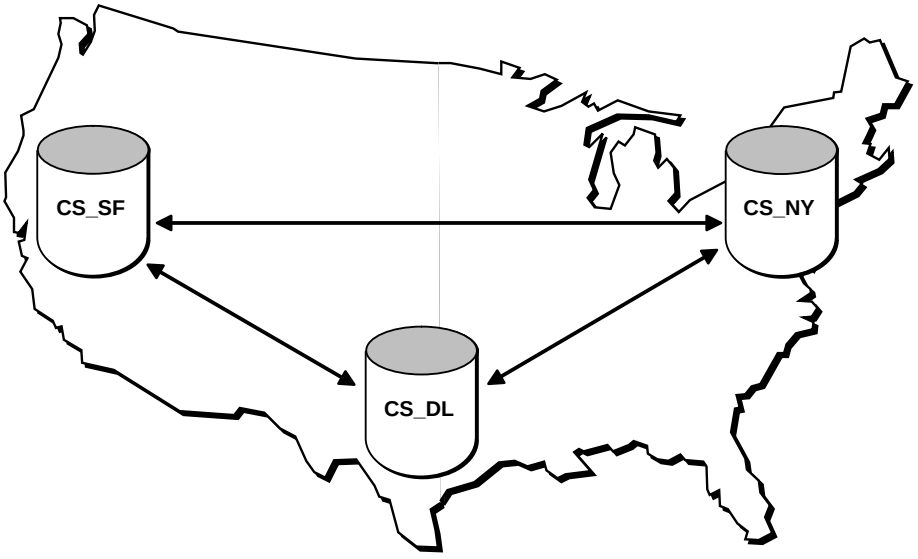
### Failover Site

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

### Distributing Application Loads

Advanced replication is useful for transaction processing applications that require multiple points of access to database information for the purposes of distributing a heavy application load, ensuring continuous availability, or providing more localized data access. Applications that have such requirements commonly include customer service oriented applications.

***Figure 1–6 Advanced Replication System Supporting Multiple Points of Update Access***




### Information Transport

Advanced replication can be useful as an information transport mechanism. For example, an advanced replication system can periodically off-load data from an update-intensive operational database to a data warehouse or data mart.

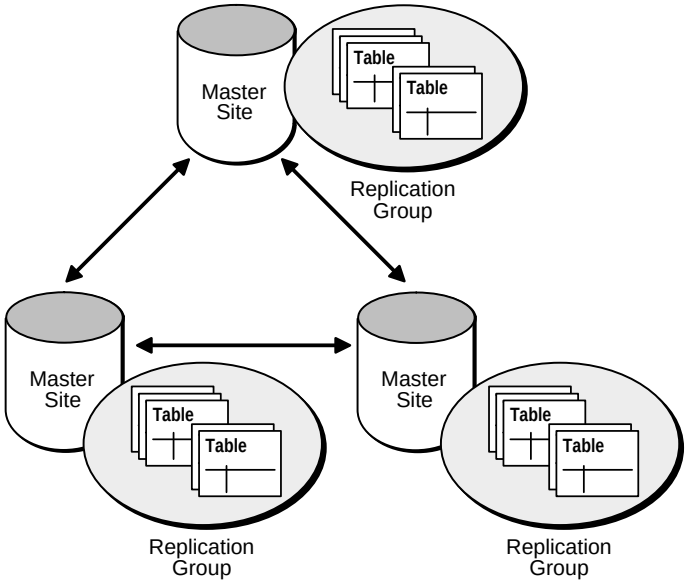
## Advanced Replication Configurations

Oracle supports the requirements of advanced replication environments using multimaster replication as well as snapshot sites.

### Multimaster Replication

Oracle's *multimaster replication* allows multiple sites, acting as equal peers, to manage groups of replicated database objects. Applications can update any replicated table at any site in a multimaster configuration. Figure 1–7 illustrates a multimaster advanced replication system.

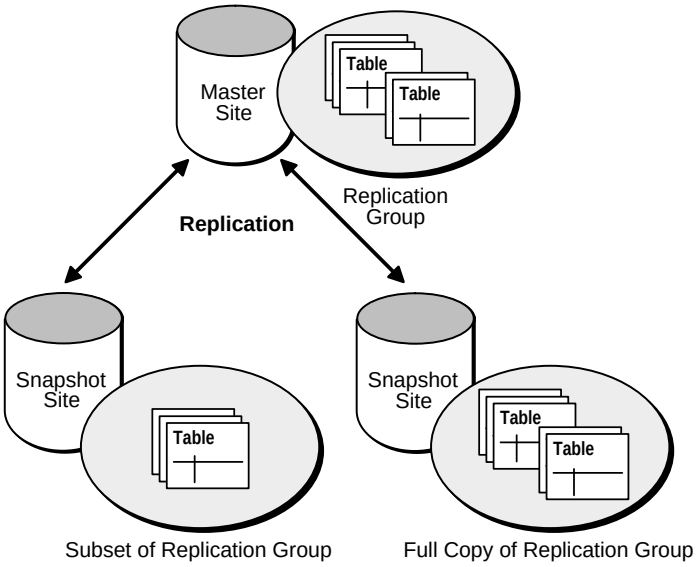
***Figure 1–7 Multimaster Replication System***




### Snapshot Sites and Updatable Snapshots

Master sites in an advanced replication system can consolidate information that applications update at remote snapshot sites. Oracle's advanced replication facility allows applications to insert, update, and delete table rows through *updatable snapshots*. Figure 1–8 illustrates an advanced replication environment with updatable snapshots.

***Figure 1–8 Advanced Replication System with Updatable Snapshots***




Updatable snapshots have the following properties.

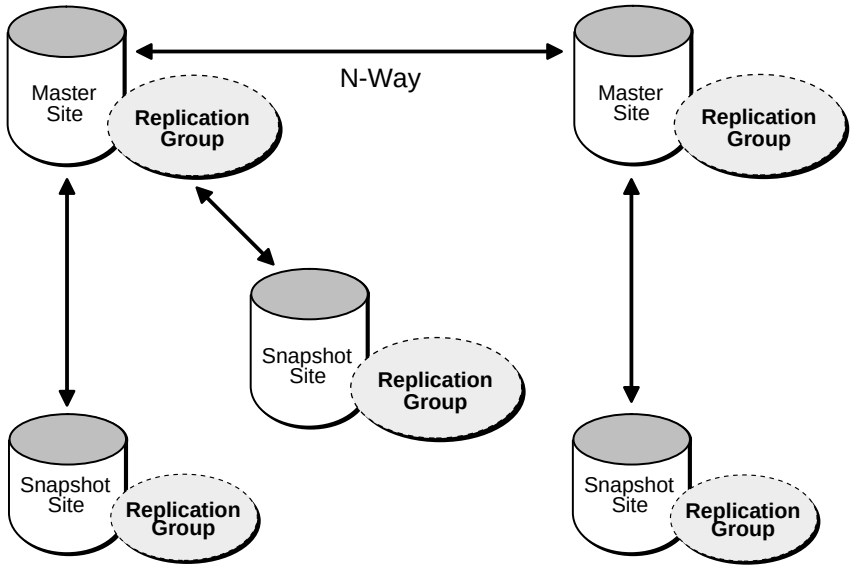
- Updatable snapshots are always simple, fast-refreshable table snapshots.
- Oracle propagates the changes made through an updatable snapshot to the snapshot's remote master table. If necessary, the updates then cascade to all other master sites.
- Oracle refreshes an updatable snapshot as part of a refresh group identical to read-only snapshots.
- Updatable snapshots have the same underlying objects, such as base table, indexes, and views, as read-only snapshots. Additionally, Oracle creates the table USLOG$_*snapshotname* to support updatable snapshots.

### Hybrid Configurations

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

For example, as shown in Figure 1–9, *n*-way replication between two masters can support full-table replication between the databases that support two geographic regions. Snapshots can be defined on the masters to replicate full tables or table subsets to sites within each region.

***Figure 1–9 Hybrid Configuration***




Key differences between updatable snapshots and replicated masters include the following:

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

## Advanced Replication and Oracle's Replication Manager

Advanced replication environments supporting an update-anywhere data model can be challenging to configure and manage. To help administer advanced replication environments, Oracle provides a sophisticated management tool, *Oracle Replication Manager*. Other sections in this book include information and examples for using Replication Manager.

## Replication Objects, Groups, Sites, and Catalogs

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

### Replication Objects

A *replication object* is a database object existing on multiple servers in a distributed database system. Oracle's advanced replication facility enables you to replicate tables and supporting objects such as views, database triggers, packages, indexes, and synonyms.

### Replication Groups

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

### Replication Sites

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

- A *master site* maintains a complete copy of all objects in a replication group. All master sites in a multi-master, advanced replication environment communicate directly with one another to propagate data and schema changes in the replication group. A replication group at a master site is more specifically referred to as a *master group*.

- Additionally, every replication group has one and only one *master definition site*. A replication group's master definition site is a master site serving as the control point for managing the replication group and objects in the group.
- A *snapshot site* supports simple read-only and updatable snapshots of the table data at an associated master site. A snapshot site's table snapshots can contain all or a subset of the table data within a replication group. However, these must be simple snapshots with a one-to-one correspondence to tables at the master site. For example, a snapshot site may contain snapshots for only selected tables in a replication group. And a particular snapshot might be just a selected portion of a certain replicated table. A replication group at a snapshot site is more specifically referred to as a *snapshot group*. A snapshot group can also contain other replication objects.

### Replication Catalog

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

### Replication Management API and Administration Requests

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

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

## Oracle's Advanced Replication Architecture

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

**Note:** Oracle offers other advanced replication features such as procedural replication and synchronous data propagation for unique application requirements. To learn more about these special configurations, read "Unique Advanced Replication Options" on page 1-28.

### Row-Level Replication

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

### Generated Replication Objects

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

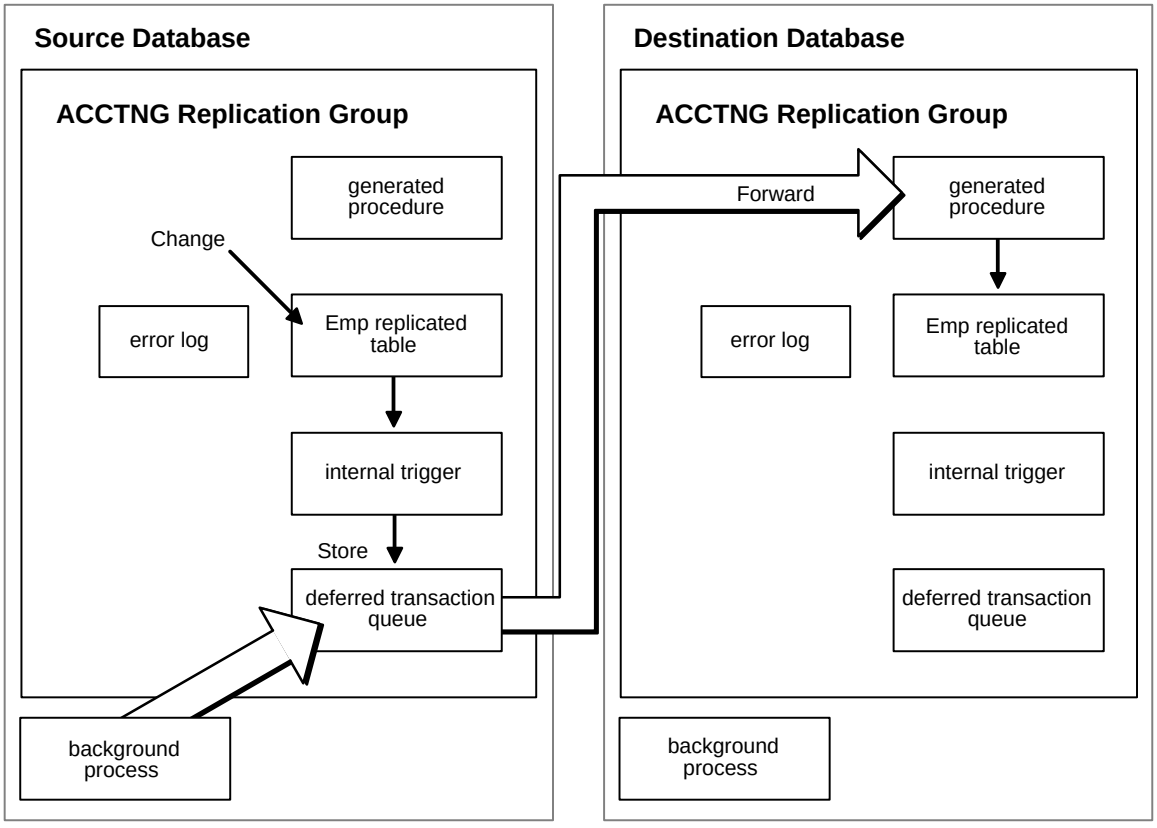
- When you replicate a table, you can generate two corresponding packages. Oracle uses a replicated table's *tablename$RP* package to replicate transactions that involve the table. Oracle uses a replicated table's *tablename$RR* package to resolve replication conflicts that involve the table.
- When you replicate a package specification and package body to support procedural replication, you can generate a corresponding *wrapper* package specification and package body. By default, Oracle names the wrapper for a package specification and package body using the name of the object with the prefix "defer_".

**Note:** With previous versions of Oracle, the server also generates PL/SQL triggers to support a replicated table. However, Oracle "activates" internal triggers when you choose to replicate a table.

### Asynchronous (Store-and-Forward) Data Propagation

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

***Figure 1–10 Asynchronous Data Replication Mechanisms***




As Figure 1–10 shows, Oracle uses its internal system of triggers, deferred transactions, deferred transaction queues, and job queues to propagate data-level changes asynchronously among master sites in an advanced replication system, as well as from an updatable snapshot to its master table.

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

### Serial Propagation

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

### Parallel Propagation

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

Parallel propagation uses the same execution mechanism Oracle uses for parallel query, load, recovery, and other parallel operations. Each server process propagates transactions through a single stream. A parallel coordinator process controls these

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

### Purging of the Deferred Transaction Queue

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

### Snapshot Propagation Mechanisms

Updatable snapshots in an advanced replication environment can both "push" and "pull" data to and from its master table, respectively.

**Master Table Updates** Updates to an updatable snapshot are asynchronously pushed to its master table using Oracle's row-level, asynchronous data propagation mechanisms (RPCs, deferred transactions, and job queues).

**Snapshot Refresh** Identical to basic replication environments, advanced replication systems use Oracle's snapshot refresh mechanism to pull changes asynchronously from a master table to associated updatable (and read-only) snapshots.

**Other Considerations** An updatable snapshot's push and pull tasks are independent operations that you can configure associatively or separately.

- Snapshot sites can configure refresh groups to automatically push changes made to the member snapshots to the master site, and then refresh the snapshots.
- Snapshot sites can configure updatable snapshots to push changes to the master site and refresh snapshots at different times and intervals.

For example, an advanced replication environment that consolidates information at a master site might configure updatable snapshots to push changes to the master site every hour but refresh updatable snapshots infrequently, if ever.

## Replication Administrators, Propagators, and Receivers

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

- Every site in an Oracle advanced replication system requires at least one *replication administrator*, a user responsible for configuring and maintaining replicated database objects.

- Each replication site in an Oracle advanced replication system requires special user accounts to propagate and apply changes to replicated data.

### Configuration Options

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

## Replication Conflicts

Advanced replication systems supporting an update-anywhere model of data replicas must address the possibility of replication conflicts. The following sections explain the different types of replication conflicts, when they can occur, and how Oracle detects and resolves replication conflicts.

### Types of Replication Conflicts

Three types of *conflicts* can occur in advanced replication environments: uniqueness conflicts, update conflicts, and delete conflicts.

**Uniqueness Conflicts** A *uniqueness conflict* occurs when the replication of a row attempts to violate entity integrity (a PRIMARY KEY or UNIQUE constraint). For example, consider what happens when two transactions originating from two different sites each insert a row into a respective table replica with the same primary key value. In this case, replication of the transactions will cause a uniqueness conflict.

**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 occur when two different transactions originating from different sites update the same row at nearly the same time.

**Delete Conflicts** A *delete conflict* occurs when two transactions originate from different sites, with one transaction deleting a row that the other transaction updates or deletes.

### Replicated Data Models and Conflicts

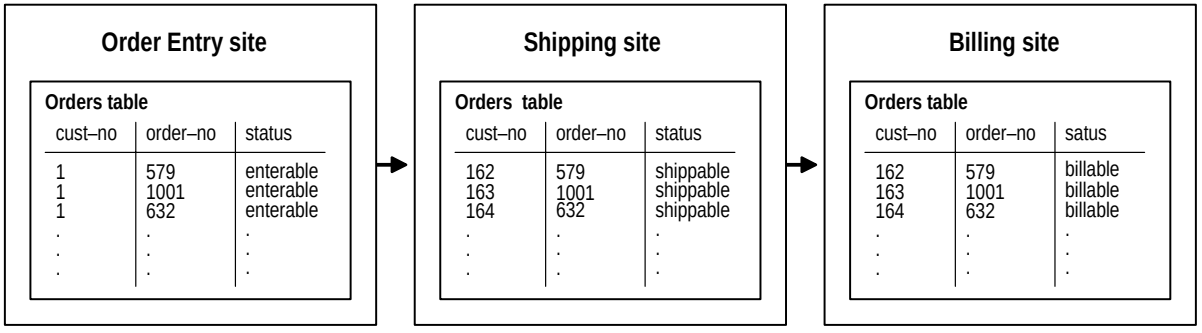
When designing applications to work on top of a database system using advanced replication, you must consider the possibility of replication conflicts. In such a case, your applications must use one of several different replicated *data ownership models* that ensure global database integrity by avoiding or resolving replication conflicts.

**Primary Site, Static Ownership** *Primary ownership*, also called *static ownership*, is the replicated data model that basic read-only replication environments support. Primary ownership prevents all replication conflicts, because only a single server permits update access to a set of replicated data.

Rather than control ownership of data at the table level, applications can use horizontal and vertical partitioning to establish more granular static ownership of data. For example, applications might have update access to specific columns or rows in a replicated table on a site-by-site basis.

**Dynamic Ownership** The *dynamic ownership* replicated data model is less restrictive than primary site ownership. With dynamic ownership, the capability to update a data replica moves from site to site, still ensuring that only one site provides update access to specific data at any given point in time. A workflow system clearly illustrates the concept of a dynamic ownership. For example, related departmental applications can read the status code of a product order to determine when they can and cannot update the order. Figure 1–11 illustrates an application that uses a dynamic ownership model.

***Figure 1–11 Dynamic Ownership in an Order Processing System***




**Shared Ownership** Primary site ownership and dynamic ownership replication data models that promote conflict avoidance are often too restrictive to implement for some database applications. Some applications must operate using a *shared ownership* replicated data model in which applications can update the data of any table replica at any time.

When a shared data ownership system replicates changes asynchronously (store-and-forward replication), corresponding applications must avoid or detect and resolve replication conflicts if and when they occur.

### Conflict Avoidance Techniques

Typically, you can design an advanced replication system to avoid all or a large percentage of replication conflicts, especially uniqueness and delete conflicts.

### Conflict Detection

Although conflict avoidance is preferable, it is not always possible. When an application uses a shared ownership data model with asynchronous row-level replication and replication conflicts are possible, Oracle automatically detects uniqueness, update, and delete conflicts. To detect conflicts during replication, Oracle compares a minimal amount of row data from the originating site with the corresponding row information at the receiving site. When there are differences, Oracle detects the conflict.

To detect replication conflicts accurately, Oracle must be able to uniquely identify and match corresponding rows at different sites during data replication. Typically, Oracle's advanced replication facility uses the primary key of a table to uniquely identify rows in the table. When a table does not have a primary key, you must designate an *alternate key*. This key serves as a column or set of columns that Oracle uses to identify rows in the table during data replication. In either case, you should not allow applications to update the identity columns of a table. This ensures that Oracle can identify rows and preserve the integrity of replicated data.

### Conflict Resolution

When a receiving site in an advanced replication system is using asynchronous row-level replication and it detects a conflict in a transaction, the default behavior is to log the conflict and the entire transaction and leave the local version of the data intact. In most cases, you should use Oracle's advanced replication facility to automate the resolution of replication conflicts.

**Column Groups** Oracle uses *column groups* to detect and resolve update conflicts during asynchronous, row-level advanced replication. A column group is a logical grouping of one or more columns in a table. Every column in a replicated table is part of a single column group. When configuring replicated tables, you can create column groups and then assign columns and corresponding update conflict resolution methods to each group.

Each column group in a replicated table can have a list of one or more update conflict resolution methods. Indicating multiple methods for a group allows Oracle to resolve an update conflict in different ways should other methods fail to resolve the conflict. When trying to resolve an update conflict for a group, Oracle executes the group's resolution methods in the order listed for the group.

By default, every replicated table has a *shadow column group*. A table's shadow column group contains all columns that are not within a specific column group. You cannot assign conflict resolution methods to a table's shadow group.

**Conflict Resolution Methods** When designing column groups you can choose from among many built-in *conflict resolution methods*. For example, to resolve update conflicts, you might choose to have Oracle overwrite the column values at the destination site with the column values from the originating site. Oracle offers many other update conflict resolution methods.

Oracle also allows you to assign uniqueness conflict resolution methods to PRIMARY KEY and UNIQUE constraints. However, Oracle offers no delete conflict resolution methods. Consequently, applications that operate within an asynchronous, shared ownership data model should avoid delete conflicts by not using DELETE statements to delete rows. Instead, applications can mark rows for deletion and configure the system to periodically purge deleted rows using procedural replication.

## Unique Advanced Replication Options

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

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

### Procedural Replication

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

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

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

**Additional Information:** See “Using Procedural Replication” on page 7-2.

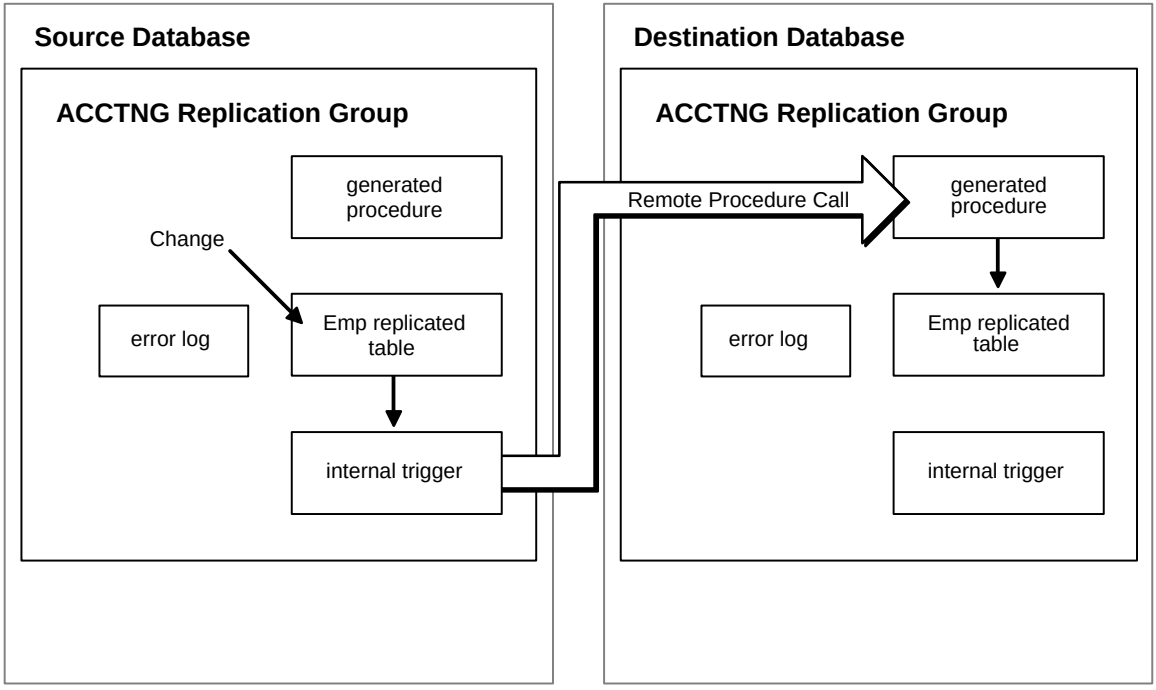
### Synchronous (Real-Time) Data Propagation

Asynchronous data propagation is the normal configuration for advanced replication environments. However, Oracle also supports synchronous data propagation for applications with special requirements. *Synchronous data propagation* occurs when an application updates a local replica of a table, and within

the same transaction also updates all other replicas of the same table. Consequently, synchronous data replication is also called *real-time data replication*. Use synchronous replication only when applications require that replicated sites remain continuously synchronized.

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

***Figure 1–12 Synchronous Data Replication Mechanisms***




As Figure 1–12 shows, Oracle uses the same system of internal database triggers to generate RPCs that replicate data-level changes to other replication sites to support synchronous, row-level data replication. However, Oracle does not defer the execution of such RPCs. Instead, data replication RPCs execute within the boundary of the same transaction that modifies the local replica. Consequently, a data-level change must be possible at all sites that manage a replicated table or else a transaction rollback occurs.

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

**Replication Conflicts and Synchronous Data Replication** When a shared ownership system replicates all changes synchronously (real-time replication), replication conflicts are cannot occur. With real-time replication, applications use distributed transactions to update all replicas of a table at the same time. As is the case in nondistributed database environments, Oracle automatically locks rows on behalf of each distributed transaction to prevent all types of destructive interference among transactions. Real-time replication systems can prevent replication conflicts. However, this type of system is highly dependent on system and network availability because it can function only when all system sites are available.

# 2

# Using Basic Replication

This chapter explains how to configure and manage a basic replication environment using read-only table snapshots. This chapter covers the following topics.

- Quick Start: Building a Basic Replication Environment.
- Preparing a Database for Snapshots.
- Creating Snapshot Logs.
- Creating Simple Snapshots.
- Creating Snapshots with Subqueries.
- Creating Refresh Groups.
- Managing a Basic Replication Environment.
- Tuning Performance for Snapshots.
- Other Basic Replication Options.
- Monitoring Basic Replication Environments.

**Note:** Examples in this chapter explain how to use both SQL commands and Oracle's Enterprise Manager to build and manage basic replication systems. For complete information about Enterprise Manager, see your Enterprise Manager documentation.

# Quick Start: Building a Basic Replication Environment

To create a basic replication environment that uses table snapshots to provide read-only access to a master site, complete the following steps:

1. Design the basic replication environment. Decide which master tables you want to replicate using read-only table snapshots, and which databases require such snapshots.
2. At each snapshot site, create the schemas and database links necessary to support snapshots.
3. At the master site, create the snapshot logs necessary to support fast refreshes of all snapshots.
4. Create the snapshots at each snapshot site.
5. At each snapshot site, create the refresh groups that the snapshots will use to refresh, and assign each snapshot to a refresh group.
6. Grant privileges necessary for application users to access snapshots.

For detailed information about each step, see the later sections of this chapter.

## A Simple Example

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

### Step 1: Design the Environment

The first step is to design the basic replication environment. This example demonstrates how to replicate the tables SCOTT.EMP and SCOTT.DEPT at the master site DBS1 using corresponding snapshots in DBS2.

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

### Step 2: Create Snapshot Site Schemas and Database Links

The master site, DBS1, already has the schema SCOTT with the tables EMP and DEPT, which have primary keys. The snapshot site must have a corresponding schema SCOTT to contain the proposed snapshots. Additionally, the snapshot schema SCOTT must have a private database link that establishes connections to the corresponding schema at the master site.

The following SQL command script completes the schema and database link setup at the snapshot site.

```
CONNECT system/manager@dbs2;
CREATE USER scott IDENTIFIED BY tiger QUOTA UNLIMITED ON data;
GRANT CONNECT TO scott;
CONNECT scott/tiger@dbs2;
CREATE DATABASE LINK dbs1 CONNECT TO scott IDENTIFIED BY tiger;
```

### Step 3: Create Necessary Master Site Snapshot Logs

Before creating snapshots, create the master site snapshot logs that will be necessary to support fast refreshes for the snapshots. The following SQL command script demonstrates how to create snapshot logs at the master site to support the snapshots.

```
CONNECT system/manager@dbs1;
CREATE SNAPSHOT LOG ON scott.emp;
CREATE SNAPSHOT LOG ON scott.dept;
```

### Step 4: Create Snapshots

Once the necessary snapshot logs are in place, you can create the snapshots. The following SQL command script demonstrates how to create the snapshots SCOTT.EMP and SCOTT.DEPT at the snapshot site.

```
CONNECT system/manager@dbs2;
CREATE SNAPSHOT scott.emp AS SELECT * FROM scott.emp@dbs1.acme.com;
CREATE SNAPSHOT scott.dept AS SELECT * FROM scott.dept@dbs1.acme.com;
```

### Step 5: Create Snapshot Site Refresh Groups

After creating the snapshots, make sure to assign all related snapshots to a refresh group that the snapshots will use to refresh. The following SQL command script demonstrates how to create and schedule the refresh group SCOTT.REFGRP1 at the snapshot site and assign to it the new EMP and DEPT snapshots.

```
CONNECT system/manager@dbs2;
DBMS_REFRESH.MAKE(
  name => 'scott.refgrp1',
  list => 'scott.dept,scott.emp',
  next_date => SYSDATE,
  interval => 'SYSDATE+1/24');
COMMIT;
```

### Step 6: Grant Access to Snapshots

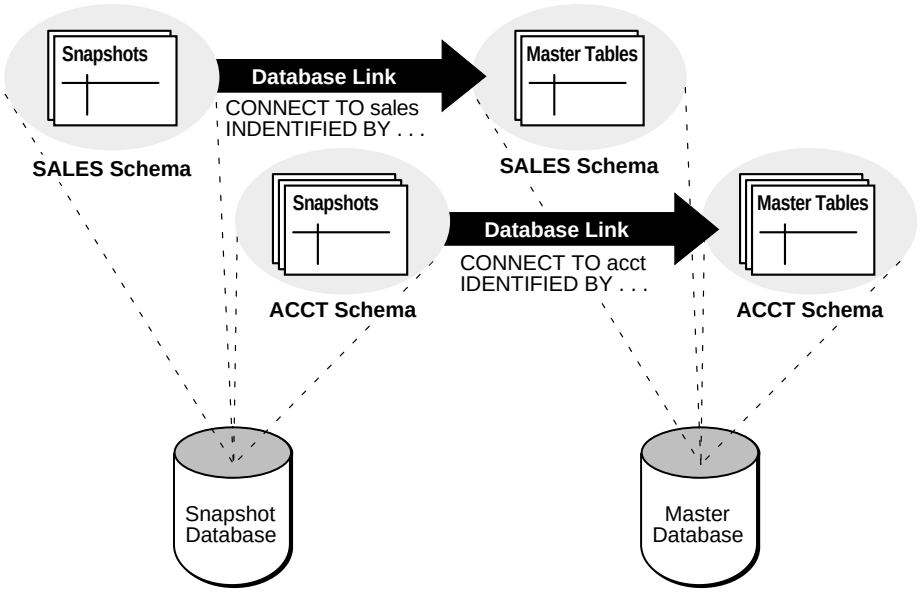
After configuring snapshots, grant access to users.

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

# Preparing a Database for Snapshots

Before building snapshots in a basic replication environment, you must prepare for the snapshots at each snapshot site. Specifically, each snapshot site must have the schemas and database links to support the proposed snapshots. To simplify the configuration of a basic replication environment, implement the schema and database link design that Figure 2–1 and the following sections describe.

***Figure 2–1 Recommended Schema and Database Link Configuration For Basic Replication Environments***




## Necessary Schemas

A schema containing a snapshot in a snapshot database should correspond to the schema that contains the master table in the master database. Therefore, identify the tables and encompassing schemas in the master database that you want to replicate using read-only table snapshots. Then in the snapshot database, create accounts with the same names as the schemas that contain master tables in the master database. For example, if all master tables are in the SALES schema of the DB1 database, create a corresponding SALES account in the snapshot database DB2.

## Necessary Database Links

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

To ensure proper behavior, a snapshot's defining query must use a database link that includes an embedded user name and password in its definition. This type of database link always establishes connections to the remote database using the specified account. Additionally, the remote account that the link uses must have the privileges necessary to access the data referenced in the snapshot's defining query.

To simplify the setup of a basic replication environment, create a private database link from each snapshot schema in the snapshot database to the corresponding schema in the master database. Be sure to embed the associated master database account information in each private database link at the snapshot database. For example, the SALES schema at a snapshot database DB2 should have a private database link DB1 that connects using the SALES username and password.

**Note:** For additional information about snapshot queries among distributed systems, see the book, *Oracle8 Distributed Database Systems.*

## Planning for Snapshot Refreshes

Before you create snapshots, you should have an idea of how you would like to refresh them using snapshot refresh groups. Some planning will help make the configuration of refresh groups much easier.

### Granting Required Privileges

Privilege management can be challenging in a basic replicated environment. To create and refresh a snapshot, both the creator and snapshot owner must be able to issue the defining query of the snapshot. This capability depends directly on the database link that the snapshot's defining query uses to access the master table of the snapshot. To simplify privilege management for snapshot refreshes in a basic replication environment, use the recommended schema / database link setup described in the previous sections.

### Designing Refresh Groups

Refresh groups not only provide a mechanism to refresh a number of snapshots efficiently as a group, but also allow you to preserve the referential integrity and transaction consistency among the table snapshots of several related master tables.

After refreshing all snapshots in a refresh group, the data of all snapshots in the group corresponds to the same transaction consistent point in time.

Before building a basic replication environment, identify the snapshots that require referential and transaction integrity. Then group related snapshots accordingly into the same refresh group.

**Note:** Do not arbitrarily group snapshots to form large refresh groups. Snapshot refreshes of an unnecessarily large refresh group can generate a significant amount of rollback data requiring the use of a large rollback segment.

Additionally, it's important to decide how often applications will require a refresh of their snapshots. If the master tables receive predictable updates, automatically refresh the associated snapshots at the appropriate interval.

### Starting SNP Background Processes

To simplify administration, most basic replication environments configure refresh groups to refresh all snapshots automatically. Each snapshot database in a basic replication environment must start one or more SNP background processes to support the automatic refresh of snapshot refresh groups. The following initialization parameters control the SNP background process setting for each server.

- JOB_QUEUE_PROCESSES specifies the number of SNP*n* background processes per server, where *n* is 0 to 9 followed by A to Z. When you configure refresh groups to update snapshots automatically, set this parameter to a value of one or higher. One snapshot refresh process will usually be sufficient unless you have many refresh groups that refresh simultaneously.
- JOB_QUEUE_INTERVAL specifies the interval, in seconds, between wake-ups for the SNP background processes of a server. The default of 60 seconds is adequate for typical replication environments.

## Creating Snapshot Logs

Every master table must have an associated snapshot log to support efficient, fast refreshes of corresponding snapshots. If a master table does not have an associated snapshot log, then complete refreshes of corresponding snapshots are the only option.

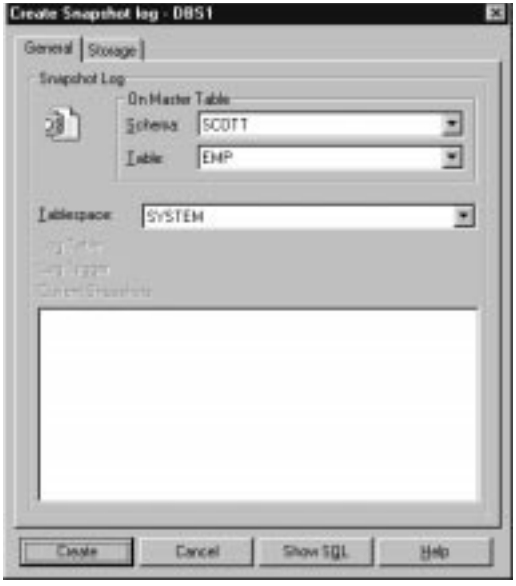
Oracle creates the snapshot log for a master table in the same database as the table. A master table's snapshot log is a table itself. When a transaction changes information in the master table, an internal trigger on the master table automatically inserts rows into the corresponding snapshot log. Rows in a snapshot

log list changes made to the master table, as well as information about which snapshots have and have not been updated to reflect the changes at the master table.

A snapshot log is associated with a single master table. Likewise, a master table can have only one snapshot log. When a master table is the data source for several different snapshots, perhaps in different databases, all snapshots of the master table use the same snapshot log.

**Note:** For complete information about managing snapshot logs, see "Managing Snapshot Logs" on page 2-28.

To create a snapshot log for a master table, use Enterprise Manager's Schema Manager application or the SQL command CREATE SNAPSHOT LOG. The following example creates a snapshot log.




The equivalent CREATE SNAPSHOT LOG statement is:

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

Before creating a snapshot log for a master table, you should consider several issues, as described in the sections that follow.

## Snapshot Log Names

When you create a snapshot log for a master table, Oracle automatically creates the log as a table MLOG$_*master_table_name* in the schema that contains the master table. If a master table name is longer than 20 bytes, Oracle truncates the *master_table_name* portion at 20 bytes and appends the name with a four-digit number to ensure uniqueness. This guarantees that the objects comply with the naming rules for schema objects.

## Required Privileges

The privileges required to create a snapshot log directly relate to the privileges necessary to create the underlying objects associated with a snapshot log.

- If you own the master table, you can create an associated snapshot log if you have the CREATE TABLE privilege.
- If you are creating a snapshot log for a table in another user's schema, you must have the CREATE ANY TABLE and COMMENT ANY TABLE privileges, as well as the SELECT privilege for the master table.

In either case, the owner of the snapshot log must have sufficient quota in the tablespace intended to hold the snapshot log.

## Timing of Snapshot Log Creation

If you plan to perform a fast refresh for snapshots, be sure to create a corresponding snapshot log for the snapshot's master table *before* creating the snapshot. If you create the snapshot log after the snapshot, Oracle will perform the first refresh of the snapshot as a complete refresh rather than as a fast refresh.

## Special Requirements for Primary Key Snapshots

By default, Oracle creates a snapshot log to support primary key snapshots. Therefore, the master table must contain a valid PRIMARY KEY constraint before you can create a snapshot log.

## Snapshot Log Storage Parameters

You can set the storage options for a snapshot log during creation using the Storage page of the Create Snapshot Log property sheet in Schema Manager.

In general, it is best to set a snapshot log's storage options as follows:

- Set PCTFREE to 0 and PCTUSED to 99.
- Set extent storage parameters according to the update activity (the number of INSERT, UPDATE, and DELETE statements) on the master table.

## Special Requirements for Subquery Snapshots

To support a simple snapshot that uses a subquery in its defining query, the following requirements must be met:

- For every master table that appears in a subquery, there must be a corresponding snapshot log.
- Each snapshot log must specify filter columns for each column referenced in the subquery of the snapshot, excluding primary key columns.

**Note:** Columns using a LOB datatype cannot be filter columns.

See "Advanced Subsetting with Subqueries" on page 2-16 for several examples of snapshot logs with filter columns that are necessary to support snapshots with subqueries.

### Internal Mechanisms of Snapshot Log Creation

When you create a snapshot log, Oracle performs several operations internally:

- Oracle creates the base table MLOG$_*master_table_name* to store the primary keys (and / or ROWIDs), filter columns, and timestamps of rows updated in the master table. Oracle does not update the timestamp column until the log is first used by a snapshot refresh.
- Oracle activates an internal trigger on the master table to insert the primary keys (and / or ROWIDs), filter columns, and timestamps of inserted, updated, and deleted rows into the master snapshot log.

The schema containing a master table also contains the base table for its snapshot log.

**Caution:** Do not alter or change data in the underlying table that supports a snapshot log.

## Creating Simple Snapshots

To create a read-only snapshot, you can use the Create Snapshot property sheet of Enterprise Manager's Schema Manager application or the SQL command CREATE SNAPSHOT.

**Note:** For complete information about managing snapshots, see "Managing Read-Only Snapshots" on page 2-34.

The following example creates a simple, read-only snapshot.

The equivalent CREATE SNAPSHOT statement is:

```
CREATE SNAPSHOT scott.emp
 AS SELECT * FROM scott.emp@dbs1.acme.com;
```

Before creating a read-only table snapshot, consider the issues discussed in the sections that follow.

## Snapshot Names

A snapshot name must be unique within the encompassing schema. A snapshot name can be up to 30 bytes in length; however, keep snapshot names to 19 or fewer bytes when possible. When a snapshot name contains more than 19 bytes, Oracle automatically truncates the prefixed names of the underlying table, and appends them with a four-digit number to ensure new object names are unique and that they comply with the naming rules for schema objects.

## Required Privileges

Several privileges are necessary to create a fully functional snapshot.

- To create a snapshot in your own schema, you must have the CREATE SNAPSHOT, CREATE TABLE, and CREATE VIEW system privileges.
- To create a snapshot in another user's schema, you must have the CREATE ANY SNAPSHOT system privilege.
- The schema that contains the snapshot must have sufficient quota in the target tablespace to store the snapshot's base table and index.
- To create and refresh a snapshot, both the creator and snapshot owner must be able to issue the defining query of the snapshot. This capability depends directly on the database link that the snapshot's defining query uses.

## Special Requirements for Primary Key Snapshots

By default, Oracle creates all new snapshots as primary key snapshots. To create a snapshot:

- The master table for a snapshot must contain an enabled PRIMARY KEY constraint before you create the new snapshot.
- The defining query of the snapshot must reference all columns in the master table's primary key.

## Snapshot Refresh Settings

Although Schema Manager and the CREATE SNAPSHOT command allow you to specify refresh settings for individual snapshots, you should always refresh a snapshot as part of a refresh group. See "Creating Refresh Groups" on page 2-24 for more information about configuring snapshot refreshes with refresh groups. Additionally, see "Individual Snapshot Refreshes" on page 2-44 for more information about configuring an individual snapshot for refresh.

## A Snapshot's Defining Query

When creating a new snapshot, consider the following issues relating to the query that defines the snapshot's structure.

### Referenced Master Table Column Datatypes

A snapshot's defining query can reference master table columns that use the following Oracle datatypes: NUMBER, DATE, CHAR, VARCHAR2, NCHAR, NVARCHAR2, RAW, ROWID, BLOB, CLOB, and NCLOB. Snapshots **cannot** include columns that use the LONG, LONG RAW, or BFILE datatypes. Additionally, Oracle does **not** support user-defined object types within snapshots.

Oracle propagates BLOBs, CLOBs, and NCLOBs during fast refresh only when:

- A LOB has been updated (including piece-wise updates and appends).
- New rows exist that satisfy the selection criteria of the snapshot.

A snapshot's defining query cannot reference a LOB column in its WHERE clause.

### Restrictions for Simple Snapshots

To create a simple snapshot, the snapshot's defining query **cannot** contain the following SQL attributes:

- Distinct or aggregate functions.
- GROUP BY or CONNECT BY clauses.
- Joins (other than the allowed types of subqueries).
- Set operations.

When a snapshot definition uses any of the above attributes, the snapshot is a complex snapshot. Oracle cannot use a snapshot log to perform fast refreshes for a complex snapshot.

### Explicit Table References

Always design the defining query of a snapshot so it explicitly references its remote data. Otherwise, the snapshot might reference different remote data during snapshot creation and subsequent refreshes. For example, consider when SCOTT creates a snapshot using the following commands:

```
CONNECT scott/tiger
CREATE DATABASE LINK sales.hq.com USING 'hq.sales.com';
CREATE SNAPSHOT emp AS SELECT * FROM emp@sales.hq.com;
```

In this example, the snapshot definition implicitly references the remote table SCOTT.EMP because the database link establishes a connection to the remote database as SCOTT/TIGER during snapshot creation. Instead, simply make the remote table reference explicit as the following example shows.

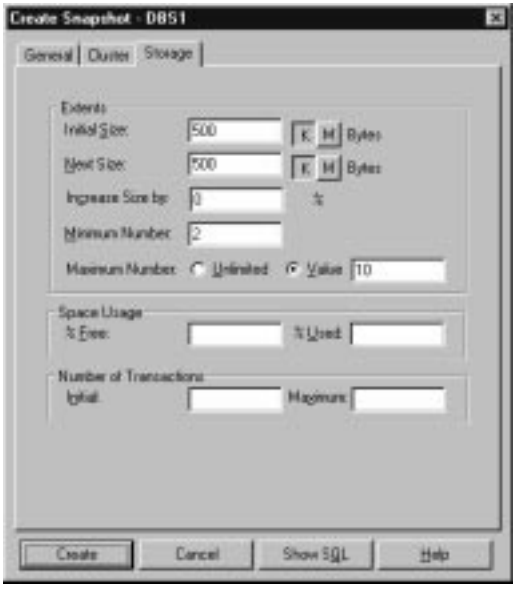
```
CREATE SNAPSHOT emp AS SELECT * FROM scott.emp@sales.hq.com;
```

To clearly identify the remote data to which a snapshot corresponds, the snapshot's defining query should:

- Reference fully qualified table names rather than partial table names.
- Reference only remote tables, not remote master views or synonyms
- Not generate context-sensitive data. For example, do not create a simple snapshot with a query that uses the SQL functions SYSDATE, UID, or USER.

## Snapshot Storage Settings

You can set the storage options for a snapshot during creation using the Storage page of the Create Snapshot property sheet in Schema Manager.




In general, a simple snapshot's storage options should mimic the storage options for its master table because they share the same characteristics. However, if a simple snapshot does not duplicate all columns of its master table, modify the snapshot storage items accordingly.

## Clustered Snapshots

When a number of master tables are clustered in the master database, consider clustering the corresponding snapshots in the remote database. To create a snapshot as part of a data cluster, use the Cluster page of the Create Snapshot property sheet in Schema Manager.

When you create a snapshot in a data cluster, Oracle always uses the storage parameters of the cluster's data segment.

## Data Load Options

By default, when you create a snapshot, Oracle immediately executes the defining query of the snapshot to populate the snapshot's base table with the rows of the corresponding master table. In large replicated environments, you should consider using snapshot cloning to reduce the network overhead necessary to create snapshot databases. For more information about snapshot cloning, see "Snapshot Cloning and Offline Instantiation" on page 7-14.

# Creating Snapshots with Subqueries

In many cases, a snapshot must correspond to a subset of the rows in its master table. Often, you can use a simple WHERE clause in the defining query of a snapshot to identify the subset of master table rows. For example, consider a corporate salesforce automation system that defines the sales territory of each salesperson by the zip codes of customers. Each salesperson's personal computer database must maintain only the information about the customers in his or her sales territory, as well as the corresponding orders. For instance, a salesperson who is responsible for customers with the zip code 19555 can create a snapshot to provide read-only access to the customers in the corresponding sales territory.

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

This simple example shows how to create basic subsetting for simple snapshots. More advanced subsetting often requires the use of a subquery in a snapshot's defining query. The following sections extend the example above to explain the benefits of subquery snapshots. Each related example explains how to create a snapshot with a subquery.

## Advanced Subsetting with Subqueries

A snapshot that uses a subquery can be a flexible solution for advanced subsetting requirements. The following examples demonstrate the creation and advantages of snapshots with subqueries in the fictional salesforce automation system described above. In the following examples, the name of the central company database is *hq.acme.com*. The following CREATE table commands describe some of the important master tables of the salesforce automation system in the SALES schema at the central company database:

**CUSTOMERS Table**

```
CREATE TABLE sales.customers
 ( c_id  INTEGER PRIMARY KEY,
   zip   INTEGER
 -- other columns defined here
 );
```

**ORDERS Table**

```
CREATE TABLE sales.orders
 ( o_id  INTEGER PRIMARY KEY,
   c_id  INTEGER
 -- other columns defined here
     );
```

**ORDER_LINES Table**

```
CREATE TABLE sales.order_lines
 ( ol_id INTEGER,
   o_id INTEGER,
  PRIMARY KEY (ol_id, o_id)
 -- other columns defined here );
```

**ASSIGNMENTS Table**

```
CREATE TABLE sales.assignments
 ( c_id INTEGER,
   s_id INTEGER,
  PRIMARY KEY (c_id, s_id)
);
```

**SALESPERSONS Table**

```
CREATE TABLE sales.salespersons
 ( s_id INTEGER PRIMARY KEY,
  s_name VARCHAR2(30) UNIQUE
 -- other columns defined here
);
```

To create snapshots that can refresh with fast refreshes, snapshot logs must also exist for the corresponding master tables. For each master table that will be referenced by a snapshot that uses a subquery, the associated snapshot log must include the appropriate filter columns. The following SQL commands build the necessary snapshot logs at the central database, *hq.acme.com*.

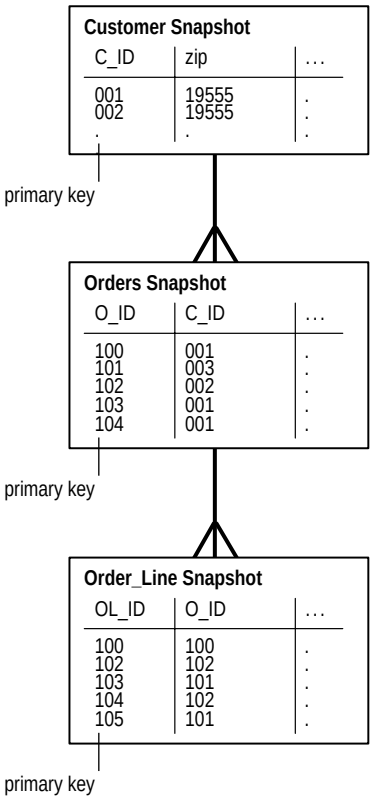
```
CREATE SNAPSHOT LOG ON sales.customers
 WITH PRIMARY KEY (zip);
CREATE SNAPSHOT LOG ON sales.orders
 WITH PRIMARY KEY (c_id);
CREATE SNAPSHOT LOG ON sales.order_lines
 WITH PRIMARY KEY;
CREATE SNAPSHOT LOG ON sales.assignments
 WITH PRIMARY KEY;
```

```
CREATE SNAPSHOT LOG ON sales.salespersons
 WITH PRIMARY KEY (s_name);
```

The following examples also assume that the corresponding SALES schema and private database link that are necessary to support the snapshots already exist in the *spdb1.acme.com* database.

### Example 1

***Figure 2–2 Advanced subsetting with a subquery.***




The CUSTOMERS snapshot in the previous section provides read-only access to the salesperson's customer records. The salesperson also requires snapshots that provide read-only access to the order information that corresponds to the

salesperson's customers. One technique would be to add a zip code column to the master ORDERS and ORDER_LINES tables and then define simple snapshots similar to the CUSTOMERS snapshot. However, such denormalization of the master tables is not typically desirable and can be difficult to maintain. A better solution is to define snapshots on the ORDERS and ORDER_LINES master tables that reference the zip code column in the CUSTOMERS table using a subquery.

The following SQL commands show how to create snapshots of the ORDERS and ORDER_LINES master tables that contain the orders and line items that correspond to the salesperson's sales territory:

**ORDERS Snapshot**

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

**ORDER_LINES Snapshot**

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

The subqueries in the example snapshots walk up the many-to-one references from the child to parent tables that may involve one or multiple levels. When these snapshots are created, Oracle fills the snapshot base tables with all orders or order line rows that belong to customers whose zip code column values match the snapshot selection criterion. Subsequent fast refreshes return only the rows that have changed since snapshot creation or since the last refresh.

If a zip code column is updated so that a customer no longer satisfies the selection criterion of the snapshot, the orders and order line rows for the customer will be removed from the snapshots during the next refresh. If a zip code column in another customer row is updated so that it satisfies the selection criterion of the snapshot, the customer's orders and order line rows will be added during the next refresh.

### Example 2: A Better Approach

The snapshots that the previous section define derive their subsets of the CUSTOMERS, ORDERS, and ORDER_LINES tables by embedding a zip code into each snapshot's defining query. Consequently, the snapshots cannot accommodate a simple change in the salesperson's territory—for example, the snapshots would have to be rebuilt if the salesperson's sales territory zip code changed from 19555 to 19500. For greater flexibility, the snapshots in the following example show how subqueries can also traverse many-to-many references between tables in selected cases.

This example makes use of the SALESPERSONS and ASSIGMENTS tables. Each salesperson has a row in the SALESPERSONS table that maps their S_ID to their name. The ASSIGNMENTS table is an intersection table that maps customers to salespersons, for example, C_ID to S_ID. The ASSIGNMENTS table implements a many-to-many relationship between CUSTOMER and SALESPERSON: A salesperson can have multiple customers; a customer can have multiple salespersons.

The following SQL commands create new CUSTOMERS, ORDERS, and ORDER_LINES snapshots that can be fast refreshed.

#### CUSTOMERS Snapshot

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

#### ORDERS Snapshot

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

```
    ( SELECT * FROM sales.salespersons@hq.acme.com s
      WHERE s.s_id = a.s_id AND s_name = 'gsmith')));
```

### ORDER_LINES Snapshot

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

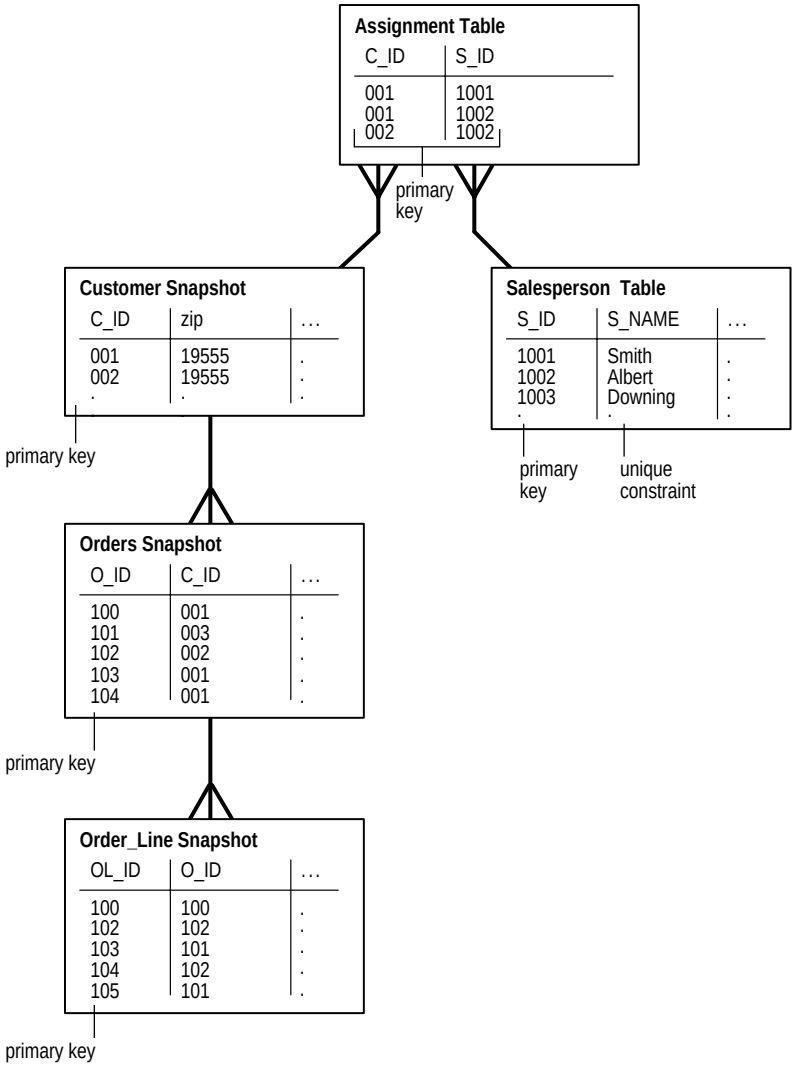
These subqueries walk up the many-to-one references from ORDER_LINES to ORDERS to CUSTOMERS and traverse the many-to-many references between CUSTOMERS and SALESPERSONS using the ASSIGNMENTS intersection table. Again, when the CUSTOMERS, ORDERS, and ORDER_LINES snapshots are created, Oracle fills the snapshot base tables with all customer, order, and order line rows belonging to customers who have been assigned to the salesperson. Subsequent fast refreshes return only the rows that have changed since snapshot creation or since the last refresh. If an ASSIGNMENTS record is deleted or updated so a customer is no longer assigned to the salesperson, the customer, order, and order line rows for the salesperson will be removed from the snapshots during the next refresh. If an ASSIGNMENTS record is inserted or updated so a customer is now assigned to the salesperson, the appropriate customer, order, and order line rows will be added during the next refresh.

To avoid the join to the SALESPERSONS table, the previous snapshot definitions with subqueries could also be defined using only the ASSIGNMENTS intersection table. For example, you could create the CUSTOMERS snapshot using only a reference to the S_ID column the ASSIGNMENTS table as follows:

```
CREATE SNAPSHOT sales.customers AS
 SELECT * FROM sales.customers@dbs1.acme.com c
 -- conditions for customers
 WHERE EXISTS
 ( SELECT * FROM sales.assignments@dbs1.acme.com a
```

```
WHERE a.c_id = c.c_id AND a.s_id = 1001);
```

***Figure 2–3 Advanced subsetting with a subquery.***




Using subqueries that traverse many-to-many relationships can provide greater flexibility in many cases. Note that in Example 2, salespeople can be assigned to

different territories and hence different customers by performing the appropriate changes to the ASSIGNMENTS table. In the earlier examples, if a salesperson were assigned different zip codes, the snapshots would need to be dropped and re-created to include the new zip code territory assignment in the snapshot definition.

Similarly in Example 2, the company could change the way territories are defined and allocated. Note that the company could move from a geographic territory scheme based on zip codes to a more elaborate scheme involving vertical industry specializations or national accounts by just rederiving and performing the appropriate updates to the ASSIGNMENTS table. In the first example, additional columns would need to be added to the CUSTOMERS table to contain the vertical industry and/or national account information, and the snapshots would again need to be dropped and re-created to use these new columns.

## Restrictions for Snapshots for Subquery

All issues relevant to simple snapshots are also relevant to snapshots with subqueries. See the section "Creating Simple Snapshots" on page 2-10 for complete information. Additionally, the defining query of a snapshot with a subquery is subject to several other restrictions to preserve the simple snapshot's fast refresh capability.

**Note:** To determine whether a simple subquery snapshot satisfies the many restrictions below, create the snapshot with "fast refresh". Oracle will return errors if the snapshot violates any restrictions for simple subquery snapshots.

- Subqueries can "walk up" many-to-one references from child to parent tables when:
  - Prior to creating the snapshot, there is a PRIMARY KEY or UNIQUE constraint on the join column(s) referenced in each parent table.
  - The join expression uses exact match or equality comparisons, in other words, "equi-joins".
- Subqueries can also traverse many-to-many references provided that:
  - Prior to creating the snapshot, there is a PRIMARY KEY or UNIQUE constraint that includes both sets of join columns in the intersection table, and a PRIMARY KEY or UNIQUE constraint on the join column(s) and a separate UNIQUE constraint on the filter column(s) of the table the intersection table joins to. If there is only an intersection table, then there must be a PRIMARY KEY or UNIQUE constraint that includes both the join column(s) and filter column(s) of the intersection table.

  - The subquery specifies an exact match or equality comparison.

    **Note:** The combination of these two properties means that only one row is returned from the many-to-many reference. To illustrate this, see "Example 2: A Better Approach" on page 2-20. The PRIMARY KEY constraint of the ASSIGNMENT table contains both the C_ID and S_ID join columns. The S_ID join column of the SALESPERSON table has a PRIMARY KEY constraint and the S_NAME filter column has a UNIQUE constraint. The subquery specifies that the value in S_NAME equals a constant, in this case, 'gsmith'.

- Snapshots must be primary key snapshots.
- The master table's snapshot log must include all filter columns referenced in the subquery. **Note:** A filter column is a non-primary-key column referenced in the subquery predicate. For more information, see "Special Requirements for Subquery Snapshots" on page 2-9.
- The subquery must be a positive subquery. For example, you can use EXISTS, but not NOT EXISTS.
- Subqueries can include AND expressions, but each OR expression may only reference columns contained within one row. Multiple OR expressions within a subquery can be ANDed.
- Each table can be joined only once within the subquery, that is, each table can be in only a single EXISTS expression.
- Each table referenced in the subquery must be located in the same master database.
- The sum of the number of selected columns plus the number of primary key columns for each table used in the subquery must be less than the maximum number of columns allowed in a table (1,000 columns).

## Creating Refresh Groups

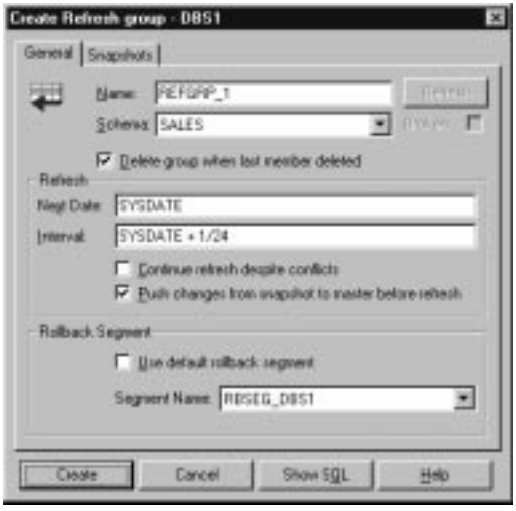
A table snapshot is a representation of its master data as that data existed at a specific moment in time. To keep a snapshot's data relatively current with the data of its master, Oracle must periodically refresh the snapshot. A snapshot refresh is an efficient batch operation that makes that snapshot reflect a more current state of its master.

To preserve referential integrity and transaction consistency among the table snapshots of several related master tables, Oracle organizes and refreshes each snapshot as part of a refresh group. After refreshing all snapshots in a refresh

group, the data of all snapshots in the group corresponds to the same transaction consistent point in time.

**Note:** For complete information about managing refresh groups, see "Managing Snapshot Refreshes and Refresh Groups" on page 2-37.

To create a refresh group, you can use Schema Manager or an equivalent call to the DBMS_REFRESH.MAKE procedure.




- On the General page of the Create Refresh Group property sheet, specify the new group's name, schema, refresh settings, and associated rollback segment.
- On the Snapshots page of the Create Refresh Group property sheet, specify a list of member snapshots for the new group. The maximum number of snapshots in a refresh group is 100.

The equivalent call to the DBMS_REFRESH.MAKE procedure is:

```
DBMS_REFRESH.MAKE(
 name             => 'sales.refgrp_1',
 list             => 'sales.customers,sales.orders,sales.order_lines',
 next_date        => SYSDATE,
 interval         => 'SYSDATE + 1/24',
 implicit_destroy => TRUE,
 rollback_seg     => 'trs_1'
);
```

Before creating refresh groups, there are several issues to consider.

## Refresh Settings

When you create a refresh group, you can determine when to first refresh the group and then how often to refresh the group thereafter.

### Next Date

A refresh group's initial refresh is determined by the setting of the Next Date field in the Create Refresh Group property sheet of Schema Manager (the NEXT_DATE parameter of the DBMS_REFRESH.MAKE procedure).

### Interval

A refresh group's refresh interval determines the automatic refresh interval for all snapshots in the group. When setting a group's automatic refresh interval, understand the following behaviors:

- The dates or date expressions that specify 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 time point that is relative to the most recent refresh date. If a network or system failure should interfere with a scheduled group refresh, the evaluation of a relative date expression could change accordingly.
- Explicit date expressions evaluate to a specific time point, regardless of the more recent refresh date.

### Example Date Expressions

The following examples are typical date expressions you can use for a refresh group's refresh interval.

- SYSDATE + 7
  Relative date expression that evaluates to exactly seven days from the most recent refresh.
- SYSDATE + 1/48
  Relative date expression that evaluates to exactly a half hour from the most recent refresh.

- NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 3/24
  Explicit date expression that evaluates to every Monday at 3 AM.
- NEXT_DAY(ADD_MONTHS(TRUNC(SYSDATE,'Q'),3),'THURSDAY')
  Explicit date expression that evaluates to the first Thursday of each quarter.

#### Refresh Settings for Updatable Snapshots

When you create a refresh group, you can specify other settings that are designed specifically for refresh groups that include updatable snapshots in an advanced replication environment. For basic, read-only replication environments, simply use the default settings. See "Managing Refresh Groups" on page 4-22 for more information about configuring these special refresh group settings with advanced replication systems.

### Refresh Types

By default, Oracle attempts to refresh each snapshot in a group using a fast refresh. Oracle performs a complete refresh of an individual snapshot in a refresh group only when one of the following situations is true:

- A snapshot log is not available for a snapshot.
- A snapshot is complex—Oracle cannot fast refresh a complex snapshot.
- A snapshot was created with an individual refresh setting (not recommended).
- A snapshot's master table has been dropped and subsequently re-created.

### Rollback Segment Setting

When Oracle refreshes the snapshots in a refresh group, the server can generate a significant amount of rollback data. When you create a refresh group, be sure to target a sufficiently large rollback segment for the group's refreshes.

## Managing a Basic Replication Environment

The following sections explain how to manage the various components of a basic replication environment, including snapshot logs, snapshots, and snapshot refresh groups.

## Managing Snapshot Logs

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

- Altering Snapshot Logs.
- Managing Snapshot Log Space.
- Reorganizing Master Tables that Have Snapshot Logs.
- Dropping Snapshot Logs.

### Altering Snapshot Logs

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

**Required Privileges** Only the owner of the master table or a user with the SELECT privilege for the master table can alter a snapshot log.

**Altering Snapshot Log Storage Parameters** To alter a snapshot log's storage parameters, use the Storage and Options pages of the Snapshot Log property sheet or an equivalent ALTER SNAPSHOT LOG statement. For example:

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

**Altering a Snapshot Log to Add Filter Columns** To add new filter columns to a snapshot log, use the SQL command ALTER SNAPSHOT LOG. For example:

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

### Managing Snapshot Log Space

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

For example, Oracle refreshes the CUSTOMERS snapshot at the SPDB1 database. However, the server that manages the master table and associated snapshot log

does not purge the snapshot log rows used during the refresh of this snapshot until the CUSTOMERS snapshot at the SPDB2 database also refreshes using these rows.

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

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

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

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

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

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

**Additional Information**: The parameters for the DBMS_SNAPSHOT. PURGE_LOG procedure are described in Table 9–209.

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

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

1. Acquire an exclusive lock on the master table to prevent updates from occurring during the following process.

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

2. Using a second database session, copy the rows in the snapshot log (in other words, the MLOG$ base table) to a temporary table.

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

3. Using the second session, truncate the log using the SQL command TRUNCATE.

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

4. Using the second session, reinsert the old rows so that you do not have to perform a complete refresh of the dependent snapshots.

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

5. Using the first session, release the exclusive lock on the master table.

   ```
   ROLLBACK;
   ```

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

**Required Privileges** The owner of a snapshot log or a user with the DELETE ANY TABLE system privilege can truncate a snapshot log.

### Reorganizing Master Tables that Have Snapshot Logs

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

**Reorganization Notification** When you reorganize a table, any ROWID information of the snapshot log must be invalidated. Oracle detects a table reorganization automatically only if the table is *truncated* as part of the reorg. See "Method 2 for Reorganizing Table t" on page 2-32.

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

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

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

**Note:** To ensure that any previously fast refreshable snapshot is still refreshable, follow the guidelines in "Methods of Reorganizing a Database Table" on page 2-31.

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

**Examples** The following two statements preserve snapshot log information when the master table is truncated:

```
TRUNCATE TABLE tablename PRESERVE SNAPSHOT LOG;
TRUNCATE TABLE tablename;
```

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

```
TRUNCATE TABLE tablename PURGE SNAPSHOT LOG
```

**Methods of Reorganizing a Database Table** Oracle provides four table reorganization methods that preserve the capability for fast refresh; these appear under the following headings. Other reorg methods require an initial complete refresh to enable subsequent fast refreshes.

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

**Method 1 for Reorganizing Table t**

1. Call dbms_snapshot.begin_table_reorganization for table t.
2. Rename table t to t_old.
3. Create table t as select * from t_old.
4. Call dbms_snapshot.end_table_reorganization for new table t.

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

• Ensure that no transaction is issued against the reorganized table between calling dbms_snapshot.begin_table_reorganization and dbms_snapshot.end_table_reorganization.

**Method 2 for Reorganizing Table t**

1. Call dbms_snapshot.begin_table_reorganization for table t.
2. Export table t.
3. Truncate table t with PRESERVE SNAPSHOT LOG option.
4. Import table t using conventional path.
5. Call dbms_snapshot.end_table_reorganization for new table t.

• **Warning**: When you truncate master tables as part of a reorg, you must use the PRESERVE SNAPSHOT LOG clause of the truncate table DDL.

• Ensure that no transaction is issued against the reorganized table between calling dbms_snapshot.begin_table_reorganization and dbms_snapshot.end_table_reorganization.

**Method 3 for Reorganizing Table t**

1. Call dbms_snapshot.begin_table_reorganization for table t.
2. Export table t.

3. Rename table t to t_old.
4. Import table t using conventional path.
5. Call dbms_snapshot.end_table_reorganization for new table t.

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

• Ensure that no transaction is issued against the reorganized table between calling dbms_snapshot.begin_table_reorganization and dbms_snapshot.end_table_reorganization.

**Method 4 for Reorganizing Table t**

1. Call dbms_snapshot.begin_table_reorganization for table t.
2. Select contents of table t to a flat file.
3. Rename table t to t_old.
4. Create table t with the same shape as t_old.
5. Run SQL*Loader using conventional path.
6. Call dbms_snapshot.end_table_reorganization for new table t.

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

• Ensure that no transaction is issued against the reorganized table between calling dbms_snapshot.begin_table_reorganization and dbms_snapshot.end_table_reorganization.

### Dropping Snapshot Logs

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

- All snapshots of a master table have been dropped.
- All snapshots of a master table are to be completely refreshed, not fast refreshed.

To drop a snapshot log, you can use Schema Manager to remove the log or an equivalent DROP SNAPSHOT LOG statement.

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

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

## Managing Read-Only Snapshots

The following sections describe how to create and manage read-only snapshots in a basic replication environment. Topics include:

- Using Read-Only Snapshots.
- Registering a Snapshot at its Master Site.
- Altering Read-Only Snapshot Storage Parameters.
- Dropping Read-Only Snapshots.
- Recovering Read-Only Snapshots from Media Failure.

### Using Read-Only Snapshots

Applications can query snapshots just like a table or view. For example, the following query references a local snapshot named CUSTOMERS.

```
SELECT * FROM sales.customers;
```

Applications cannot issue any INSERT, UPDATE, or DELETE statements when using a read-only snapshot; if they do, Oracle returns an error.

**Caution:** Although applications can issue INSERT, UPDATE, and DELETE statements against the base table of a snapshot, such operations will corrupt the snapshot. Applications should update a master table only, and allow Oracle to refresh the snapshot to make it a more current version of its master table. When applications must update a local snapshot rather than a remote master table, you must configure an advanced replication system using updatable snapshots. See Chapter 4, “Using Snapshot Site Replication” for more information.

**Caution**: Administrators should never modify the structure of a read-only snapshot in any way. For example, never add triggers or integrity constraints to the base table of a snapshot.

**Required Privileges** To query a snapshot, a user must own the snapshot, have the SELECT privilege for the snapshot, or have the SELECT ANY TABLE system privilege.

Grant the SELECT privilege for a snapshot to all users who require access to the snapshot. **Do not** allow users to connect to the snapshot database using the account

that owns any snapshots. Otherwise, a malicious user could use the private database link to access the remote master table in any way.

**Creating Views and Synonyms Based on Snapshots** Optionally, you can create views and synonyms based on snapshots. For example, the following statement creates a view based on the EMP snapshot.

```
CREATE VIEW sales_dept AS
 SELECT ename, empno
   FROM scott.emp
    WHERE deptno = 10;
```

### Registering a Snapshot at its Master Site

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

**Viewing Information about Registered Snapshots** You can use the General page of the Snapshot Log property sheet in Schema Manager to list the snapshots associated with a snapshot log. For additional information, you can query the DBA_REGISTERED_SNAPSHOTS data dictionary view to list the following information about a remote snapshot:

- The owner, name, and database that contains the snapshot.
- The snapshot's defining query.
- Other snapshot characteristics, such as its refresh method (fast or complete).

You can also join the DBA_REGISTERED_SNAPSHOT view with the DBA_SNAPSHOT_LOGS view at the master site to obtain the last refresh times for each snapshot. Administrators can use this information to monitor snapshot activity from master sites and coordinate changes to snapshot sites if a master table needs to be dropped, altered, or relocated.

**Internal Mechanisms** Oracle automatically registers a read-only snapshot at its master database when you create the snapshot, and unregisters the snapshot when you drop it.

**Note:** Oracle7 master sites cannot register snapshots.

**Caution:** Oracle cannot guarantee the registration or unregistration of a snapshot at its master site during the creation or drop of the snapshot, respectively. If Oracle cannot successfully register a snapshot during creation, Oracle completes snapshot

registration during a subsequent refresh of the snapshot. If Oracle cannot successfully unregister a snapshot when you drop the snapshot, the registration information for the snapshot persists in the master database until it is manually unregistered.

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

**Additional Information**: The REGISTER_SNAPSHOT and UNREGISTER_SNAPSHOT procedures are described on Table 9–193 and Table 9–196, respectively.

### Altering Read-Only Snapshot Storage Parameters

To alter the storage parameter of a snapshot that is not in a data cluster, you can use Schema Manager or the SQL command ALTER SNAPSHOT.

```
ALTER SNAPSHOT sales.customers
  STORAGE (NEXT 500K);
```

**Required Privileges** To alter a snapshot's storage parameters, the snapshot must be contained in your schema. Otherwise, you must have the ALTER ANY SNAPSHOT system privilege.

### Dropping Read-Only Snapshots

You can drop a snapshot independently of its master tables or its snapshot log. To drop a local snapshot, use Schema Manager or the SQL command DROP SNAPSHOT.

```
DROP SNAPSHOT sales.customers;
```

When you drop the only snapshot of a master table, you should also drop the snapshot log of the master table, if present.

**Dropping a Snapshot's Master Table** When you drop a master table, Oracle automatically drops the associated snapshot log, if present. Alternatively, all associated snapshots remain and continue to be accessible. However, when Oracle attempts to refresh a snapshot based on a nonexistent master table, Oracle returns an error.

If you later re-create the master table, a dependent snapshot can again successfully **complete** refresh, as long as Oracle can successfully issue the defining query of the

snapshot against the new master table. However, Oracle cannot perform a *fast* refresh of the snapshot until *after* you create a snapshot log for the new master table. If Oracle cannot successfully refresh a snapshot after dropping and re-creating its master table, drop and re-create the snapshot as well.

**Privileges Required to Drop a Snapshot** Only the owner of a snapshot or a user with the DROP ANY SNAPSHOT system privilege can drop a snapshot.

### Recovering Read-Only Snapshots from Media Failure

When a media failure occurs, it may be necessary to recover either a database that contains a master table of a snapshot or a database with a snapshot. If a master database is independently recovered to a past point in time (that is, coordinated time-based distributed database recovery is not performed), any dependent remote snapshot that refreshed in the interval of lost time will be inconsistent with its master table. In this case, the administrator of the master database should instruct the remote administrator to perform a complete refresh of any inconsistent snapshot.

**Additional information**: See the book *Oracle8 Backup and Recovery Guide* to learn about recovering from media failure.

## Managing Snapshot Refreshes and Refresh Groups

The following sections describe how to manage refresh groups for the snapshots in an Oracle database. Topics include:

- Altering Refresh Groups.
- Deleting a Refresh Group.
- Manually Refreshing Snapshot Refresh Groups.
- Troubleshooting Refresh Problems.

### Altering Refresh Groups

After you create a refresh group, you can alter it in several ways. The following sections explain how to add new member snapshots to a refresh group, remove member snapshots from a refresh group, and alter the automatic refresh interval for a refresh group.

**Adding Members to a Refresh Group** To add member snapshots to a refresh group, use the Snapshots page of the Refresh Group property sheet in Schema Manager or call the DBMS_REFRESH.ADD procedure.

```
DBMS_REFRESH.ADD
  name   => 'sales_refgrp_1',
   list   => 'sales.order_lines'
);
```

**Note:** The maximum number of snapshots in a refresh group is 100.

**Removing Members from a Refresh Group** To remove member snapshots from a refresh group, use the Snapshots page of the Refresh Group property sheet in Schema Manager or call the DBMS_REFRESH.SUBTRACT procedure.

```
DBMS_REFRESH.SUBTRACT
 name  => 'sales_refgrp_1',
 list   => 'sales.assignments'
);
```

After removing a snapshot from a refresh group, Oracle will not refresh the snapshot unless you add it to another refresh group or refresh it manually.

**Altering a Group's Refresh Settings** To change a refresh group's next refresh date or automatic refresh interval, use the General page of the Refresh Group property sheet in Schema Manager or call the DBMS_REFRESH.CHANGE procedure.

```
DBMS_REFRESH.CHANGE
 name       => 'sales_refgrp_1',
 next_date  => SYSDATE,
 interval   => 'SYSDATE + 1'
);
```

### Deleting a Refresh Group

To drop a refresh group and implicitly remove all member snapshots from the group, use Schema Manager or call the DBMS_REFRESH.DESTROY procedure.

```
DBMS_REFRESH.DESTROY( name => 'sales_refgrp_1');
```

When you drop a refresh group, Oracle will not refresh the former member snapshots unless you add them to another refresh group or refresh them manually.

### Manually Refreshing Snapshot Refresh Groups

Refresh groups enable you to configure automatic refresh settings that ease the administration of snapshots in a replication environment. However, some circumstances justify the need to refresh a snapshot manually. For example, immediately following a bulk data load into a master table, dependent snapshots will not represent the master table's data. Rather than wait for the next scheduled automatic group refreshes, you might want to manually refresh dependent snapshot groups to propagate the new rows of the master table immediately to associated snapshots.

To refresh a group of snapshots manually, use the Refresh button of the General page in the Refresh Group property sheet of Schema Manager, or call the DBMS_REFRESH.REFRESH procedure.

```
DBMS_REFRESH.REFRESH('sales_refgrp_1');
```

Forcing a manual refresh of a refresh group does not affect the next automatic refresh of the group.

### Troubleshooting Refresh Problems

The following sections explain several common snapshot refresh problems.

**Common Problems** Several common factors can prevent the automatic refresh of a group of snapshots:

- The lack of an SNP background process at the snapshot database.
- An intervening network or server failure.
- An intervening server shutdown.

When a snapshot refresh group is experiencing problems, ensure that none of the above situations is preventing Oracle from completing group refreshes.

**Automatic Refresh Retries** When Oracle fails to refresh a group automatically, the group remains due for its refresh to complete. Oracle will retry an automatic refresh of a group with the following behavior:

- Oracle retries the group refresh first one minute later, then two minutes later, four minutes later, and so on, with the retry interval doubling with each failed attempt to refresh the group.
- Oracle does not allow the retry interval to exceed the refresh interval itself.
- Oracle retries the automatic refresh up to sixteen times.

If after sixteen attempts to refresh a refresh group Oracle continues to encounter errors, Oracle considers the group broken. The General page of the Refresh Group property sheet in Schema Manager indicates when a refresh group is broken. You can also query the BROKEN column of the USER_REFRESH and USER_REFRESH_CHILDREN data dictionary views to see the current status of a refresh group.

The errors causing Oracle to consider a snapshot refresh group broken are recorded in a trace file. After you correct the problems preventing a refresh group from refreshing successfully, you must refresh the group manually. Oracle then resets the broken flag so that automatic refreshes can happen again.

**Additional Information**: The name of the snapshot trace file is of the form SNP*n*, where *n* is platform specific. See your platform-specific Oracle documentation for the name on your system.

**Snapshots Continually Refreshing** If you encounter a situation where Oracle continually refreshes a group of snapshots, check the group's refresh interval. Oracle evaluates a group's automatic refresh interval before starting the refresh. If a group's refresh interval is less than the amount of time it takes to refresh all snapshots in the group, Oracle continually starts a group refresh each time the SNP background process checks the queue of outstanding jobs.

**Snapshot Logs Growing Without Bounds** If a snapshot log is growing without bounds, check to see whether a network or site failure has prevented a master from becoming aware that a snapshot has been dropped. You may need to purge part of the log as described in "Purging Rows from a Snapshot Log" on page 2-29 and unregister the snapshot.

# Tuning Performance for Snapshots

The following sections explain how you can tune the performance of snapshots.

## Indexing Snapshots

To maximize the query performance of applications that use a snapshot, you can create indexes for the snapshot. To index a column (or columns) of a snapshot, create an index on the underlying base table of the snapshot.

**Caution**: Do not declare non-deferrable UNIQUE constraints for a snapshot.

### Tuning Subquery Snapshots

You can improve the performance of fast refreshes for a snapshot with a subquery by defining referential integrity constraints (whenever possible) and non-unique indexes on the equi-join columns referenced in the snapshot's defining query. You must declare the necessary FOREIGN KEY constraints before you create the snapshot. Furthermore, you should not modify or drop these constraints as long as the snapshot exists.

If you use the cost-based optimizer, be sure to analyze the master table, snapshot base table, snapshot log (at the master), and the updatable snapshot log (at the snapshot site, if updatable).

**Recommendation**: In general, it is best to analyze the snapshot logs when they are empty or nearly empty.

A variable number of indexes are automatically added on the underlying snapshot tables used by simple snapshot with a subquery. These indexes prevent full-table scans during refresh when a change to a row in a master table affects that snapshot's partitioning criteria.

See "Example 2: A Better Approach" on page 2-20. Using the example ORDER_LINES snapshot, the SNAP$_ORDER_LINE table would have a nonunique index on each of the following columns: C_ID, O_ID, and OL_ID.

**Note:** Frequent modification of columns used by snapshots with restricted subqueries to perform equi-joins or filtering can cause performance degradation during refresh.

## Other Basic Replication Options

Oracle supports some additional basic replication features that can be useful in certain situations.

- Complex Snapshots.
- ROWID Snapshots.
- Individual Snapshot Refreshes.

### Complex Snapshots

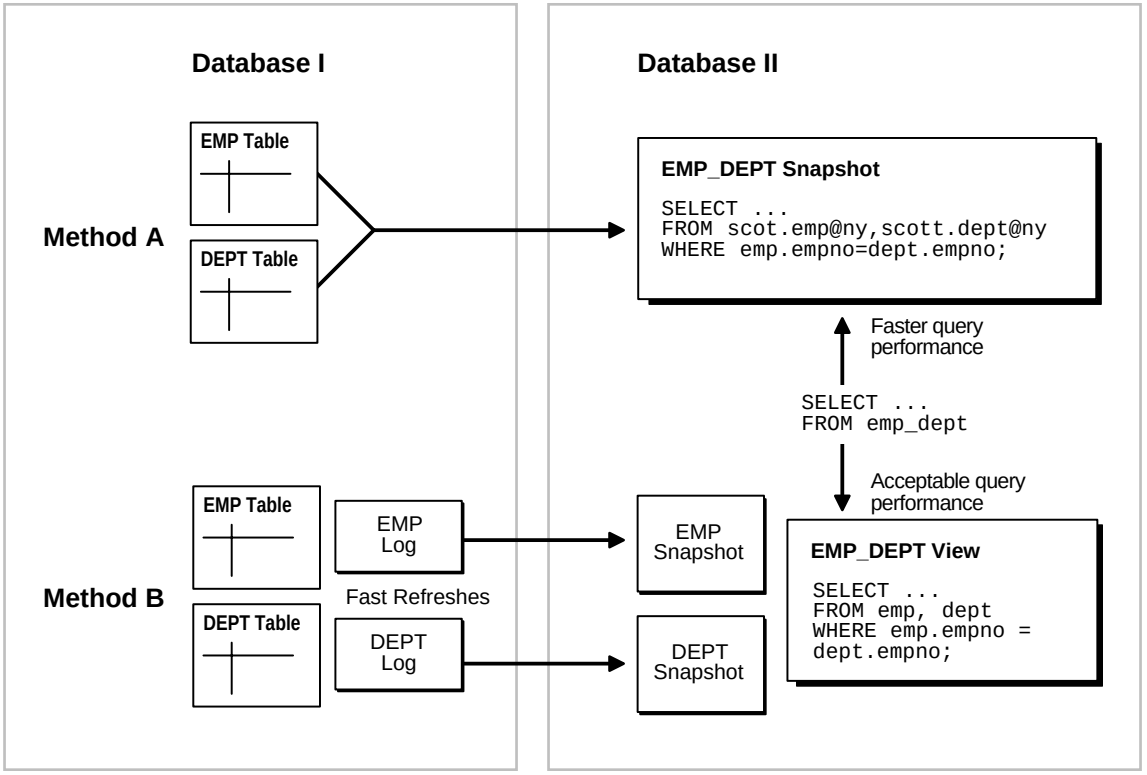
When the defining query of a snapshot contains a distinct or aggregate function, a GROUP BY or CONNECT BY clause, join, restricted types of subqueries, or a set operation, the snapshot is a complex snapshot.

**Note:** In most cases, you should avoid using complex snapshots, because Oracle *cannot* perform a fast refresh of a complex snapshot. Consequently, the use of complex snapshots can degrade network performance during complete snapshot refreshes.

### A Comparison of Simple and Complex Snapshots

For certain applications, you might want to consider the use of a complex snapshot. Figure 3-1 and the following sections summarize some issues to consider.

***Figure 2–4 Comparison of Simple and Complex Snapshots***




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

**Simple Snapshots with a Joined View** Method B shows two simple snapshots in Database II, as well as a view that performs the join in the snapshots' database. Query performance against the view would not be as good as the query performance against the complex snapshot in Method A. However, the simple snapshots can be more efficiently refreshed using snapshot logs.

In summary, to decide which method to use:

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

### Storage Settings for Complex Snapshots

Because a complex snapshot is always completely refreshed, set its PCTFREE to 0 and PCTUSED to 100 for maximum efficiency.

## ROWID Snapshots

For backward compatibility, Oracle supports ROWID snapshots in addition to the default, primary key snapshots. Oracle bases a ROWID snapshot on the physical row identifiers (ROWIDs) of rows in the master table. ROWID snapshots should be used only for snapshots of master tables in an Oracle7 database, and should not be used when creating new snapshots of master tables in Oracle databases.

### Creating a ROWID Snapshot

To explicitly create a ROWID snapshot for an Oracle7 Release 7.3 master table, you must use a CREATE SNAPSHOT statement with the WITH ROWID clause.

```
CREATE SNAPSHOT ...
 WITH ROWID
  AS ...
```

### Snapshot Logs and ROWID Snapshots

To support fast refreshes of ROWID snapshots, you must create a snapshot log of a master table with a CREATE SNAPSHOT LOG statement that includes the WITH ROWID clause.

### Master Table Reorganization and ROWID Snapshots

All ROWID snapshots of a master table require a full refresh after you reorganize the table. Subsequently, Oracle can fast refresh all ROWID snapshots.

### Individual Snapshot Refreshes

In typical environments, all snapshots are refreshed as part of a refresh group. However, certain environments do not have concern for referential integrity among snapshots, and require or prefer that snapshots be refreshed individually. For example, you might want to refresh certain snapshots more frequently than others.

To configure a snapshot for individual refreshes, create a refresh group for only the single snapshot. Oracle automatically creates a refresh group for an individual snapshot when you:

- Create or alter a snapshot with Schema Manager and specify refresh settings.
- Create a snapshot with the SQL command CREATE SNAPSHOT and specify a REFRESH clause.
- Alter a snapshot with the SQL command ALTER SNAPSHOT and specify a REFRESH clause.

## Monitoring Basic Replication Environments

The Schema Manager component of Enterprise Manager allows you to view the properties of snapshot logs, snapshots, and refresh groups quickly in a basic replication environment. Alternatively, you can query a database's data dictionary to view information about snapshot logs, snapshots, and refresh groups. The following data dictionary views provide information about the components of a basic replication environment:

- DBA_SNAPSHOTS
- DBA_SNAPSHOT_LOGS
- DBA_REFRESH, DBA_RGROUP, DBA_REFRESH_CHILDREN
- DBA_REGISTERED_SNAPSHOTS

**Note:** Most data dictionary views have three versions, which have different prefixes: USER_*, ALL_*, and SYS.DBA_*.

# 3

# Using Multimaster Replication

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

This chapter covers the following topics.

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

**Note:** This chapter explains how to manage a multimaster replication system that uses the default replication architecture—row-level replication using asynchronous propagation. For information about configuring procedural replication and synchronous data propagation, see Chapter 7, "Advanced Techniques". Also, examples appear throughout this chapter of how to use the Oracle Replication Manager tool to manage a multimaster replication system. Each section lists equivalent replication management API procedures for your reference. For

complete information about Oracle's replication management API, see Chapter 9, "Replication Management API Reference".

# Quick Start: Building a Multimaster Replication Environment

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

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

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

## A Simple Example

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

### Step 1: Design the Environment

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

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

### Step 2: Use the Replication Manager Setup Wizard

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

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

### Step 3: Create a Master Group

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

### Step 4: Grant Access to Replication Objects

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

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

### Other Steps

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

# Preparing for Multimaster Replication

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

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

## The Replication Setup Wizard

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

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

To start the Replication Manager setup wizard:

1. Click **File**.
2. Click **Setup Wizard**.

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

### Create Master Sites

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

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

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

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

### Create Replication Administrator, Propagator, and Receiver Accounts

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

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

- By default, the wizard creates a single account that is capable of performing administrator, propagator, and receiver functions. Simply enter an account name and password and then press **Next** to continue.
- Alternatively, when you want to create a separate account for replication administration apart from the propagator/receiver account, click **Different Propagator/Receiver** and enter different account information for each user account. Then press **Next** to continue.

### Create Schemas to Organize Replication Objects

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

To add new schemas to the list

1. Click **New**.
2. In the **New Replicated Object Schema** dialog that appears, enter the name of a schema that you want to use to contain replication objects, as well as the password for the schema. When you finish, click **Add** to add the schema to the list of schemas in the setup wizard.
3. Repeat Steps 1 and 2 for each schema. After you enter the final schema's information, click **OK** to add and dismiss the **New Replicated Object Schema** dialog.
4. After reviewing the list of schemas, click **Next** to continue.

### Create Scheduled Links

The next page of the setup wizard lets you indicate default propagation characteristics for all master sites in the system. The setup wizard uses this information to create corresponding scheduled links from each master site to all other master sites. For explanations of each setting in this page of the wizard, see "Creating a Scheduled Link" on page 3-9.

After reviewing the default scheduling settings, click **Next** to continue.

### Specify Purge Scheduling

The next page of the setup wizard lets you configure the default purge schedule for the deferred transaction queue at each master site in the system. For explanations of each setting in this page of the wizard, see "Purging a Site's Deferred Transaction Queue" on page 3-12.

After reviewing the default purge settings, click **Next** to continue.

### Customize Each Master Site

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

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

To customize a master site's settings:

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

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

- Replication administrator and / or propagator accounts.
- USING CLAUSE for administrator / propagator / receiver database links.
- Scheduled propagation settings to another master site.

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

### Reviewing and Building the Configuration

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

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

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

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

### What's Next?

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

1. Create the master groups at the site that serves as each group's master definition site. See "Managing Master Groups" on page 3-15 for more information about creating and managing master groups.
2. Create or identify the replication objects for each master group at its master definition site. See "Adding Objects to a Master Group" on page 3-19 for more information about creating and managing replicated objects.
3. Configure conflict resolution for all replicated tables in each master group. See Chapter 5, "Conflict Resolution" for more information about configuring conflict resolution for replication objects.
4. Resume replication activity for each master group. See "Resuming Replication Activity for a Master Group" on page 3-19 for more information about managing the replication activity for master groups.
5. Monitor the multimaster environment to ensure that the system is operating properly. See "Monitoring an Advanced Replication System" on page 6-4 for more information about monitoring the activity of master groups.

## Starting SNP Background Processes

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

- JOB_QUEUE_PROCESSES specifies the number of SNP*n* background processes per server, where *n* is 0 to 9 followed by A to Z. Set this parameter to a value of one or higher. Two or three background processes will usually be sufficient unless you have a large system.
- JOB_QUEUE_INTERVAL specifies the interval, in seconds, between wake-ups for the SNP background processes of a server. The default of 60 seconds is adequate for typical replication environments.

# Managing Scheduled Links

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

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

## Creating a Scheduled Link

To create a new scheduled link:

1. Click anywhere in the open database connection that connects to the site where you want to create the link.
2. Click the **Create New** toolbar button.
3. Click **New Scheduled Link**.

Use the **Create New Scheduled Link** property sheet to create the new link. The following sections explain the settings that are available for the **General** and **Options** pages of this property sheet.

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

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

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

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

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

**Parallel Propagation** Whether to use parallel propagation (or serial propagation) for the scheduled link. When checked, you can set parallel propagation settings for the link. When unchecked, the new scheduled link uses serial propagation. See "Planning for Parallel Propagation" on page 3-31 for more information.

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

**Delay Seconds** The amount of time to continue polling the queue, even if the queue is empty. See "Guidelines for Scheduled Links" on page 3-10 for more information about this setting.

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

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

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

### Guidelines for Scheduled Links

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

For example, to simulate near real-time replication, you might want to have each scheduled link constantly push a master site's deferred transaction queue to its

destination. Alternatively, when you choose to propagate deferred transactions at regular intervals, you must decide how often and when to schedule pushes. You might want schedule links at a time of the day when connectivity is guaranteed or when communications costs are lowest, such as during evening hours. Furthermore, you might want to stagger the scheduling for links among all master sites to distribute the load that replication places on network resources.

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

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

**Deciding between Serial and Parallel Propagation** A scheduled link can push a site's deferred transaction queue using either serial or parallel propagation. For more information about issues related to serial and parallel propagation, see "Planning for Parallel Propagation" on page 3-31.

## Editing a Scheduled Link

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

1. Click on the target scheduled link.
2. Click the **Properties** toolbar button.

Use the **Edit Scheduled Link** property sheet to modify the properties of the scheduled link and apply your changes. See "Creating a Scheduled Link" on page 3-9 for more information about the properties that you can adjust for a scheduled link.

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

### Viewing the Status of a Scheduled Link

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

1. Click anywhere in the open database connection that connects to the site where you want to view links.
2. Click the **Scheduling** folder.
3. Click the **Links** folder.

The detail panel of Replication Manager displays a list of all scheduled links for the site, including the current status (enabled or disabled) of each link.

**API Equivalent:** DBMS_DEFER_SYS.DISABLED

### Deleting a Scheduled Link

To delete a scheduled link

1. Click on the target scheduled link.
2. Click the **Delete** toolbar button.

**API Equivalent:** DBMS_DEFER_SYS.UNSCHEDULE_PUSH

## Purging a Site's Deferred Transaction Queue

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

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

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

**Rollback Segment** The rollback segment to target when performing a purge of the local deferred transaction queue. Click **Browse** to display the **Select a Rollback**

**Segment** dialog and pick a rollback segment in the database. A null value for this setting allows Oracle to pick the rollback segment when purging the deferred transaction queue.

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

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

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

### Guidelines for Scheduled Purges of a Deferred Transaction Queue

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

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

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

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

## Specifying a Site's Purge Schedule

If you manually configured a master or snapshot site or want to modify a site's purge schedule, use the **Edit DB Connection** property sheet. To edit the purge schedule for a site:

1. Click on a database connection to the site.
2. Click the **Properties** toolbar button.

Use the **Purge Scheduling** page of the **Edit DB Connection** property sheet to modify the purge schedule for the site and apply your changes.

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

## Manually Purging a Site's Deferred Transaction Queue

To manually purge a master or snapshot site's deferred transaction queue, use the **Edit DB Connection** property sheet. To edit the purge schedule for a site:

1. Click on a database connection to the site.
2. Click the **Properties** toolbar button.
3. Click on the **Purge Scheduling** page.
4. Click **Purge Now**.

**API Equivalents:** DBMS_DEFER_SYS.PURGE

# Managing Master Groups

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

The following sections explain more about managing master groups.

## Creating a Master Group

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

1. Click anywhere in the open database connection that connects to the new master group's master definition site. When selecting a master definition site for a new master group, choose the site that is most likely to be available at all times. You can change a group's master definition site later, if necessary.
2. Click the **Create New** toolbar button.
3. Click **New Master Group**.

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

- To create a master group without any replication objects for the group and without registering other master sites that replicate the master group, complete the settings of the **General** page only and ignore the optional **Objects** and **Destinations** pages.
- Use the **Objects** page only when you want to add objects to the new master group during creation. You should consider several issues when adding objects to a master group. See "Adding Objects to a Master Group" on page 3-19 for more information about adding replication objects to a master group.
- Use the **Destinations** page only when you want to add destination master sites to the new master group during creation. You should consider several issues when adding master sites to a master group. See "Adding a Master Site to a Master Group" on page 3-24 for more information about adding master sites to a replication group.

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

**Warning:** If you decide to add one or more tables to a master group during creation of the group, make sure not to resume replication activity. First consider the possibility of replication conflicts, and configure conflict resolution for the replicated tables in the group. See Chapter 5, "Conflict Resolution" for more information about configuring conflict resolution for master group objects.

**API Equivalents:** DBMS_REPCAT.CREATE_MASTER_REPGROUP, DBMS_REPCAT.SET_COLUMNS

### Using Connection Qualifiers for a Master Group

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

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

**Additional Information:** See Chapter 2 of *Oracle8 Distributed Database Systems* to learn about defining connection qualifiers for a database link.

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

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

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

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

**Attention:** If you plan to use connection qualifiers, you will probably need to increase the value of the initialization parameter OPEN_LINKS at all master sites. The default is four open links per process. You will need to estimate the required value based on your usage. See the *Oracle8 Reference* for more information about the parameter OPEN_LINKS.

## Deleting a Master Group

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

1. Click on the target master group at its master definition site.
2. Click the **Delete** toolbar button.

**API Equivalent:** DBMS_REPCAT.DROP_MASTER_REPGROUP

## Suspending Replication Activity for a Master Group

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

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

- Adding an object to a master group.
- Altering the definition of an object in a master group.
- Regenerating replication support for a replication object.

- Adding or modifying conflict resolution for a replicated table.
- Adding a master site destination to a master group.
- Changing the propagation method from one site to another.

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

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

To suspend replication activity for a master group:

1. Click the target master group in the database connection the connects to the group's master definition site.
2. Click the **Properties** toolbar button.
3. Click the **Operations** page of the **Edit Master Group** property sheet.
4. Click **Suspend Replication**.

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

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

**API Equivalent:** DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY

## Resuming Replication Activity for a Master Group

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

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

To resume replication activity for a master group:

1. Click the target master group in the database connection that connects to the group's master definition site.
2. Click the **Properties** toolbar button.
3. Click the **Operations** page of the **Edit Master Group** property sheet.
4. Click **Resume Replication**.

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

**API Equivalent**: DBMS_REPCAT.RESUME_MASTER_ACTIVITY

## Adding Objects to a Master Group

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

1. Open the **Database Objects** folder at the master definition site.
2. Drag and drop selected database objects into the target master group at the same master definition site.

You can also use the **Add Objects to Group** dialog and **Edit Master Group** property sheet to add objects to a master group.

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

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

- How to replicate the object definition (and data) at all master sites. See "Replicating Object Definitions to Master Sites" on page 3-21 for more information about this topic.
- When adding a table, how to replicate table data at all master sites. See "Replicating Table Data to Master Sites" on page 3-22 for more information about this topic.
- When adding a table, how to configure conflict resolution for the table. See Chapter 5, "Conflict Resolution" for more information about this topic.

**API Equivalent:** DBMS_REPCAT.CREATE_MASTER_REPOBJECT

### Designating an Alternate Key for a Replicated Table

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

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

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

**API Equivalent:** DBMS_REPCAT.SET_COLUMNS

### Datatype Considerations for Replicated Tables

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

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

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

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

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

### Replicating Object Definitions to Master Sites

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

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

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

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

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

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

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

### Replicating Table Data to Master Sites

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

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

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

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

**Offline Instantiation** If you are currently replicating a large amount of data and want to add a new site to the system, you should consider offline instantiation. For complete information about offline instantiation, see "Snapshot Cloning and Offline Instantiation" on page 7-14.

## Altering Objects in a Master Group

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

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

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

1. Click on the target object in the master group at its master definition site.
2. Click the **Properties** toolbar button.
3. Click the **General** page of the **Edit Replication Object** property sheet.
4. Click **Alter Replication Object**.
5. Use the **Alter Replication Object** dialog to alter the definition of the replication object. For example, you might want to add a new column to a replicated table. Make sure to fully qualify all object references.

**API Equivalent:** DBMS_REPCAT.ALTER_MASTER_REPOBJECT

### Considerations

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

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

## Removing Objects from a Master Group

To remove objects from a master group:

1. Click the target master group at the master definition site.
2. Select the target objects to remove from the group.
3. Click the **Delete** toolbar button.

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

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

**API Equivalent:** DBMS_REPCAT.DROP_MASTER_REPOBJECT

## Adding a Master Site to a Master Group

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

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

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

- You must specify all master sites in the system, including all new master sites as well as the sites that the system already supports. To specify master sites that already exist in the system, use the **Browse** button on the **Create Master Sites** page of the wizard. To specify new master sites, use the **New** button of the same page.
- The wizard will ask you to enter the passwords for the SYSTEM accounts and existing propagators in the system.
- Optionally, you can choose to change system settings in the multimaster environment, including the administrator and / or propagator / receiver accounts, and default and site-specific propagation settings.

**Note:** See “The Replication Setup Wizard” on page 3-4 for more information about using the setup wizard for multimaster configuration.

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

1. Click the target master group at its master definition site.
2. Click the **Properties** toolbar button.
3. Click the **Destinations** page of the **Edit Master Group** property sheet.
4. Add one or more destination master sites to the group. A scheduled link to the new master site must already exist before you attempt to address the new master site.

**Note:** When adding a master site to a master group that contains tables with circular dependencies or a specific table that contains a self-referential constraint, you must precreate the tables at the master site and manually load data at the new site. See "Replicating Object Definitions to Master Sites" on page 3-21 for more information.

**API Equivalent:** DBMS_REPCAT.ADD_MASTER_DATABASE

## Removing a Master Site from a Master Group

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

1. Click the target master group at its master definition site.
2. Click the **Properties** toolbar button.
3. Click the **Destinations** page of the **Edit Master Group** property sheet.
4. Remove one or more destination master sites from the group.

**API Equivalent:** DBMS_REPCAT.REMOVE_MASTER_DATABASES

### Special Circumstances

The sites being removed from a master group do not have to be accessible. When a master site will not be available for an extended period of time due to a system or network failure, you might decide to drop the master site from the master group. However, because the site is unavailable, you most likely will not be able to suspend replication activity for the master group. If this is the case, you are responsible for:

- Cleaning the deferred transaction queue.
- Removing any data inconsistencies.

Specifically, the next time that you suspend replication activity for a master group, you must complete the following steps in the following order as soon as possible after the unavailable master sites are removed:

1. Suspend replication activity for the master group.
2. Remove all deferred transactions from each master site where the destination for the transaction is a removed master site.
3. Delete all deferred transactions from removed master sites.
4. Re-execute or delete all error transactions at each remaining master site.
5. Ensure that no deferred or error transactions exist at each remaining master. If you cannot remove one or more deferred transactions from a remaining master, execute the DBMS_DEFER_SYS.DELETE_TRAN procedure at the master site.
6. Ensure that all replicated data is consistent. Use the DBMS_RECTIFIER_DIFF package to determine and fix differences.
7. Resume replication activity for the master group.

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

## Generating Replication Support for Master Group Objects

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

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

Oracle generates replication support for an object using two phases:

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

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

### Generating Replication Support for Individual Objects

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

1. Click on the target object in the master group at its master definition site.
2. Click the **Properties** toolbar button.
3. Click the **General** page of the **Edit Replication Object** property sheet.
4. Click **Generate Support** to regenerate replication support for the target object.

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

### Generating Replication Support for All Master Group Tables

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

1. Click on the target master group at its master definition site.
2. Click the **Properties** toolbar button.
3. Click the **Operations** page of the **Edit Master Group** property sheet.
4. Click **Generate** to regenerate replication support for all tables in the target master group.

**API Equivalents:** DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT

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

### Minimizing Data Propagation

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

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

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

**Additional Information:** To learn about additional techniques that minimize data propagation, see "Minimizing Data Propagation for Update Conflict Resolution" on page 5-40.

## Viewing Information About Master Groups

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

### Listing Master Groups

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

1. Open the site's **Configuration** folder.
2. Click the subordinate **Master Groups** folder.

For each master group at the site, the detail panel lists the name of the master group, whether the site is the master definition site for the group, and the status of the group (for example, normal or quiesced).

### Listing Objects for a Master Group

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

1. Open the site's **Configuration** folder.
2. Open the subordinate **Master Groups** folder.
3. Click a subordinate master group.

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

### Displaying a Destination Map for a Master Group

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

1. Open the site's **Configuration** folder.
2. Open the subordinate **Master Groups** folder.
3. Open the target master group.
4. Click the **Destination Map** folder of the target master group.

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

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

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

Use the **Edit Database Destination** property sheet to

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

### Displaying Generated Objects Associated with a Master Group

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

1. Open the site's **Configuration** folder.
2. Open the subordinate **Master Groups** folder.
3. Open the target master group.
4. Click the **Generated Objects** folder of the target master group.

### Data Dictionary Views

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

- REPGROUP
- REPSITES
- REPOBJECT
- REPCATLOG

### Other Master Site Administration Issues

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

## Advanced Multimaster Replication Options

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

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

### Planning for Parallel Propagation

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

- With serial propagation, Oracle propagates replicated transactions one at a time in the same order that they are committed on the source system. To configure a schedule link with serial propagation, use the **Create Scheduled Link** or **Edit Scheduled Link** property sheet, and disable the **Parallel Propagation** setting of the **Options** page.
- With parallel propagation, Oracle propagates replicated transactions using multiple parallel streams for higher throughput. When necessary, Oracle orders the execution of dependent transactions to preserve data integrity. To configure a scheduled link with parallel propagation, use the **Create Scheduled Link** or **Edit Scheduled Link** property sheet, and enable the **Parallel Propagation** setting of the **Options** page. Set **Processes** to the desired degree of parallelism (1 or greater).

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

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

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

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

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

- PARALLEL_MAX_SERVERS.
- PARALLEL_MIN_SERVERS.
- PARALLEL_SERVER_IDLE_TIME.

**Additional Information:** See the book *Oracle8 Concepts*.

## Understanding Replication Protection Mechanisms

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

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

**Note:** Successful propagation does not necessarily imply successful application of the transaction at the remote site. Errors such as unresolvable conflicts or running out of storage space can cause the transaction to result in an error, which the remote site keeps track of. See "Displaying Error Transactions" on page 6-11 for more information about viewing and managing error transactions.

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

- In the case of serial propagation, the advanced replication facility uses two-phase commit.
- In the case of parallel propagation, the advanced replication facility uses a special-purpose distributed transaction protocol optimized for parallel operations. The remote site keeps track of the transactions that have been propagated successfully and periodically sends this information back to the local site. The local site records this information and purges the entries in its local queue that have been propagated to all destination sites. In case of failures, the local site asks the remote site for information on the transactions that have been propagated successfully so that propagation can continue at the appropriate point.

### Data Propagation Dependency Maintenance

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

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

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

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

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

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

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

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

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

**Additional Information:** See "Using Connection Qualifiers for a Master Group" on page 3-16.

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

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

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

# 4

# Using Snapshot Site Replication

This chapter explains how to configure and manage an advanced replication environment that uses snapshot sites. This chapter covers the following topics:

- Quick Start: Adding a Snapshot Site to an Advanced Replication System.
- Preparing for Snapshot Site Replication.
- Managing Snapshot Logs.
- Managing Snapshot Groups.
- Managing Snapshots.
- Managing Refresh Groups.
- Other Snapshot Site Administration Issues.

**Note:** This chapter explains how to manage an advanced replication system that uses the default replication architecture—row-level replication using asynchronous propagation. For information about configuring procedural replication and synchronous propagation, see Chapter 7, "Advanced Techniques". Also, examples appear throughout this chapter of how to use the Oracle Replication Manager tool to build and manage an advanced replication system with updatable snapshot sites. Corresponding replication management API calls are listed for your reference.

# Quick Start: Adding a Snapshot Site to an Advanced Replication System

To add snapshot sites to an advanced replication environment, complete the following steps:

1. Design the snapshot sites for the advanced replication environment. Decide what tables and supporting objects to replicate at each snapshot site, and which master site in the system will support each snapshot site.
2. Use Replication Manager's setup wizard to configure a number of databases as snapshot sites. The Replication Manager setup wizard quickly configures all components necessary to support each snapshot site.
3. At each master site, create the snapshot logs to support the proposed snapshots.
4. At each snapshot site, create the refresh groups to support the proposed snapshots.
5. At each snapshot site, create one or more snapshot groups to replicate corresponding master group objects.
6. Grant privileges so application users can access data at each snapshot site.

Detailed information about each step and other optional configuration steps are described in later sections of this chapter.

## A Simple Example

The following simple example demonstrates the steps necessary to add a snapshot site in an advanced replication environment. After configuring all master sites in an advanced replication environment, perform the following steps.

### Step 1: Design the Snapshot Site

The first step is to design each snapshot site. A snapshot site supports simple read-only and updatable snapshots of the table data at an associated master site. A snapshot site's table snapshots can contain all or just a subset of the table data within a master group. In either case, they must be simple snapshots that have a one-to-one correspondence to tables at the master site. A snapshot group can also contain other replication objects such as procedures, packages, functions, synonyms, and views. This example demonstrates how to create updatable snapshots of the master tables SCOTT.EMP and SCOTT.DEPT at the master site DBS1.

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

### Step 2: Use the Replication Manager Setup Wizard

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

- Identify the master site DBS1.
- Identify the snapshot site DBS3.
- Create the default SNAPADMIN account at the snapshot site to serve as global replication administrator and propagator account.
- Create the schema SCOTT at the snapshot site to support the proposed snapshots.
- Configure the default scheduling of data propagation from the snapshot site to its master site and purging of the deferred transaction queue.

### Step 3: Create Necessary Snapshot Logs at the Master Site

Before creating snapshot groups and snapshots for a snapshot site, make sure to create the necessary snapshot logs at the master site. A snapshot log is necessary for every master table that supports at least one simple snapshot with fast refreshes. Using Replication Manager's **Create Snapshot Log** property sheet, create a snapshot log for the master tables SCOTT.EMP and SCOTT.DEPT.

**Note:** To create a snapshot log using Replication Manager, you must establish a database connection to the master site as a database administrator with the necessary privileges (for example, as SYSTEM).

### Step 4: Create Necessary Refresh Groups at the Snapshot Site

Before creating snapshot groups and snapshots for a snapshot site, make sure to create the refresh groups that you will use to refresh snapshots at the snapshot site. Oracle organizes and refreshes individual snapshots as part of a refresh group. Using Replication Manager's **Create Refresh Group** property sheet, create the refresh group SCOTT.REFGRP_1. Simply use the default scheduling settings for the new refresh group.

### Step 5: Use the Replication Manager Snapshot Group Wizard

When you use snapshot sites in an advanced replication environment, Oracle replicates tables and related replication objects as part of a snapshot group. Using the database connection to the snapshot site DBS3, open Replication Manager's snapshot group wizard, which then guides you through the creation of a new snapshot group, including the following tasks:

- Choosing a master site for the new snapshot group.
- Picking a master group for the new snapshot group.
- Choosing the master group objects to replicate in the new snapshot group.
- Adjusting group settings and individual snapshot settings.

Use the wizard to create a new snapshot group that corresponds with the EMPLOYEE master group of the master site DBS1. When using the wizard, create full snapshots of the master tables SCOTT.EMP and SCOTT.DEPT. For both snapshots, use the default propagation and storage settings, and the refresh group SCOTT.REFGRP_1.

### Step 6: Grant Access to Replication Objects

After configuring a new snapshot group, grant access to the snapshots so that users that connect to the snapshot site can use them.

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

### Other Steps

This simple example does not mention other steps that might be necessary to configure some advanced replication systems. For example, when an advanced replication system operates using a shared ownership data model, you'll want to configure conflict resolution for all master tables before allowing applications to work with updatable snapshots. Refer to Chapter 5, "Conflict Resolution" for more detailed information about configuring conflict resolution.

# Preparing for Snapshot Site Replication

Before starting to build snapshot sites as part of an advanced replication environment, you must prepare each snapshot site database with the following:

- A replication administrator.
- A replication propagator.
- Database links to provide for snapshot site-to-master site database communication.

The Replication Manager setup wizard can build a snapshot site quickly with the above components. After building a snapshot site with the Replication Manager setup wizard, you must complete some additional preparation steps:

- Decide how to propagate changes to the master site.
- Decide how to refresh snapshots.
- Start one or more SNP background processes at the snapshot site.

The following sections explain how to prepare a database for snapshot site replication.

## The Replication Setup Wizard

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

- The wizard creates a database account to serve as a snapshot site replication administrator. By default, the wizard creates this account to serve also as the replication propagator.
- The wizard grants the necessary privileges to the replication administrator/propagator account.
- The wizard creates a scheduled link from the snapshot site to its corresponding master site.

To start the Replication Manager setup wizard:

1. Click **File**.
2. Click **Setup Wizard**.

The following sections explain how to use the Replication Manager setup wizard to prepare the snapshot sites in an advanced replication system.

### Create Snapshot Sites

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

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

### Specify the Master Site

The next page of the wizard asks you to identify the master site for the new snapshot site(s) that you will be creating.

1. Type the name of the master site (for example, DBS1).
2. Type the password for the SYSTEM user at the master site.

### Create Snapshot Sites

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

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

### Create Replication Administrator and Propagator Accounts

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

The setup wizard allows two different types of account configurations at each snapshot site:

- By default, the wizard creates a single account that is capable of performing both administrator and propagator functions. Simply enter an account name and password and then press **Next** to continue.
- Alternatively, to create a separate account for replication administration apart from the propagator account, click **Different Propagator** and enter different account information for the propagator user account. Then press **Next** to continue.

Typically, the default wizard setup option that creates one account for both functions is acceptable. However, some systems with unique security concerns require distinct administrator and propagator accounts. For more information about specialized security configurations for snapshot sites in an advanced replication environment, see "Alternative Security Setup for an Advanced Replication Snapshot Site" on page 7-24.

### Create Schemas to Organize Replication Objects

The same page of the setup wizard lets you indicate what schemas to create as schemas that will contain replication objects at the new snapshot sites. The wizard creates schemas to match those that contain replication objects at the master site, as well as private database links from snapshot site schemas to corresponding master site schemas.

To add new schemas to the list:

1. Click **Select Object Schemas**.
2. From the **Select from Master Schemas** list that appears, select all master site schemas that contain replication objects that you will be replicating to the new snapshot sites. After reviewing the list of schemas, click **Next** to continue. Replication Manager then prompts for the password of each master site schema that you specified. Enter each password and click **OK** to continue.

### Create Scheduled Links

The next page of the setup wizard lets you indicate default propagation characteristics for all snapshot sites. The setup wizard uses this information to create corresponding scheduled links from each snapshot site to the master site. For explanations of each setting in this page of the wizard, see "Managing Scheduled Links" on page 3-9.

After reviewing the default scheduling settings, click **Next** to continue.

### Specify Purge Scheduling

The next page of the setup wizard lets you configure the default purge schedule for the deferred transaction queue at each snapshot site in the system. For explanations of each setting in this page of the wizard, see "Purging a Site's Deferred Transaction Queue" on page 3-12.

After reviewing the default purge settings, click **Next** to continue.

### Customize Each Snapshot Site

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

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

To customize a snapshot site's settings:

1. Select the target snapshot site to customize.
2. Click **Customize**.

Next, use the pages of the **Customize Snapshot Site** property sheet to customize the target snapshot site's:

- Replication administrator and/or propagator accounts.
- USING CLAUSE for administrator/propagator database links.
- Scheduled propagation and purge settings.
- Supporting replication schemas.

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

### Reviewing and Building the Configuration

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

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

After you click **Finish**, Replication Manager builds the snapshot sites.

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

**Additional Information:** See "Alternative Security Setup for an Advanced Replication Snapshot Site" on page 7-24.

### What's Next?

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

1. Create snapshot groups for each snapshot site. See "Managing Snapshot Groups" on page 4-12 for more information about creating and managing snapshot groups.
2. Configure conflict resolution for the master tables of all updatable snapshots in each replication group. See Chapter 5, "Conflict Resolution" for more information about configuring conflict resolution for replication objects.
3. Resume replication activity for each master group. See "Resuming Replication Activity for a Master Group" on page 3-19 for more information about managing the replication activity for master groups.
4. Monitor the replication environment to ensure that the system is operating properly. See "Monitoring an Advanced Replication System" on page 6-4 and "Other Snapshot Site Administration Issues" on page 4-24 for more information about monitoring the activity of master and snapshot groups, respectively.

## Planning Scheduled Links and Snapshot Refreshes

An updatable snapshot in an advanced replication environment both "pushes" and "pulls" data to and from its master table, respectively. Advanced replication systems use Oracle's snapshot refresh mechanism to pull changes asynchronously

from a master table to associated updatable snapshots. In contrast, updates to an updatable snapshot are asynchronously (or synchronously) pushed to its master table using Oracle's row-level data propagation mechanisms. Because an updatable snapshot's push and pull tasks are independent operations, you configure them associatively or separately.

- A snapshot site can configure a refresh group to automatically push all changes made to the member snapshots to the master site, and then refresh the snapshots.
- A snapshot site can configure updatable snapshots to push changes to the master site and refresh snapshots at different times and intervals.

For example, an advanced replication environment that consolidates information at a master site might configure updatable snapshots to push changes to the master site every hour but refresh updatable snapshots infrequently, if ever.

Before you create a snapshot site, decide how you would like to refresh snapshots using snapshot refresh groups, and also how you would like to configure each snapshot group's scheduled link to propagate changes to its master site. Some planning will help make subsequent configuration steps much easier.

**Additional Information:** See "Managing Scheduled Links" on page 3-9 for more information about creating and managing scheduled links.

**Additional Information:** See "Managing Refresh Groups" on page 4-22 for more information about configuring and managing refresh groups.

## Starting SNP Background Processes

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

- JOB_QUEUE_PROCESSES specifies the number of SNP*n* background processes per server, where *n* is 0 to 9 followed by A to Z. Set this parameter to a value of one or higher. Two or three background processes will usually be sufficient unless you have a large system.
- JOB_QUEUE_INTERVAL specifies the interval, in seconds, between wake-ups for the SNP background processes of a server. The default of 60 seconds is adequate for typical replication environments.

# Managing Snapshot Logs

A master table's snapshot log keeps track of fast refresh data for all corresponding simple snapshots. When a server performs a fast refresh for a snapshot, it uses the data in its master table's snapshot log to refresh the snapshot very efficiently. Replication Manager has many features that help you to create and manage snapshot logs.

The following sections explain more about managing snapshot logs.

## Creating a Snapshot Log

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

To create a snapshot log at the master site of a snapshot site:

1. Click anywhere in the open database connection that connects to the master site.
2. Click the **Create New** toolbar button.
3. Click **New Snapshot Log**.

The **Create Snapshot Log** property sheet has three pages: **General**, **Tablespace and Extents**, and **Filter Columns**.

- Use the settings of the **General** page to identify the schema and name of the master table, and indicate whether the log supports primary key snapshots, ROWID snapshots, or both.
- Use the settings of the **Tablespace and Extents** page to specify storage specifications for the log.
- Use the settings of the **Filter Columns** page to identify filter columns for snapshots that contain subqueries.

## Altering a Snapshot Log

To edit the storage settings or filter columns of a snapshot log:

1. Click on the target snapshot log at the master site.
2. Click the **Properties** toolbar button.

The **Edit Snapshot Log** property sheet has three pages: **General**, **Tablespace and Extents**, and **Filter Columns**.

- Use the settings of the **General** page to supplement whether the log supports primary key snapshots, ROWID snapshots, or both.
- Use the settings of the **Tablespace and Extents** page to modify the storage specifications for the log.
- Use the settings of the **Filter Columns** page to modify the filter columns for snapshots that contain subqueries.

### Deleting a Snapshot Log

When a master table no longer supports snapshots that require fast refreshes, you can delete a corresponding snapshot log. To delete a snapshot log:

1. Click on the target snapshot log at the master site.
2. Click the **Delete** toolbar button.

## Managing Snapshot Groups

A snapshot group in an advanced replication system maintains a partial or complete copy of all or some of the objects in a corresponding master group. Replication Manager has many features that help you create and manage snapshot groups. The following sections explain more about managing snapshot groups.

### Creating a Snapshot Group

Before you create a new snapshot group, make sure that the following preliminary tasks have been completed.

- Create the scheduled link that the new snapshot group can use to propagate updates to the master site.
- If you plan to create a simple snapshot with fast refresh as part of creating a new snapshot group, make sure to create the necessary snapshot log before creating the group.

To create a new snapshot group:

1. Click anywhere in the open database connection that connects to the snapshot site.
2. Click the **Create New** toolbar button.
3. Click **New Snapshot Group**.

Replication Manager's snapshot group wizard then guides you through the creation of a new snapshot group. The following sections explain more about the settings that each page of the snapshot group wizard lets you indicate.

**API Equivalents:** DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP, DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT, DBMS_REFRESH.MAKE

### Designing a Snapshot Group

The first three pages of the snapshot group wizard allow you to design the basic structure of a snapshot group.

- First, you must specify the master site for the new group.
- Next, you must choose the corresponding master group for the new snapshot group.
- Finally, you must select which objects in the master group to replicate as part of the new snapshot group; you can choose all objects in the master group, some, or none.

### Setting Snapshot Group Settings and Defaults

The snapshot group wizard lets you specify group and default settings for all snapshots in a new snapshot group. The following sections explain each setting.

**Group Propagation Mode** When a snapshot group contains updatable snapshots, you must indicate how you want the DML changes made to the snapshots at the snapshot site to be propagated to the snapshot site's master site. The default setting is for asynchronous propagation. For more information about synchronous propagation in an advanced replication environment, see "Using Synchronous Data Propagation" on page 7-6.

**Individual Snapshot Settings** After selecting a snapshot from the **Snapshots in the group** list, you can adjust many of the snapshot's properties.

- Click **Updatable** to create the selected snapshot as an updatable snapshot.
- Click **Fast Refresh** to use fast refreshes for the selected snapshot.
- Click **Where clause** and then **Edit** to specify a WHERE clause for the selected snapshot. See "Creating Updatable Snapshots with a WHERE Clause" on page 4-18 for more information.
- Click **Tablespace & Extent Specs** to specify storage settings for the selected snapshot.

- Click **Minimum Communication** to minimize the communication necessary to perform conflict detection for the selected updatable snapshot. See "Minimizing Data Propagation" on page 4-19 for more information about this setting.
- Select a **Refresh Group** to choose a refresh group for the selected snapshot. Click **Edit** to edit the properties of a selected refresh group; click **New** to create a new refresh group.

**Note:** By default, the snapshot group wizard will create a refresh group, with the same name as the snapshot group, for all snapshots in the snapshot group.

**Default Snapshot Settings** After selecting **Default Snapshot Settings** from the **Snapshots in the group** list, you can adjust the default settings for all snapshots in the group that do not have specific settings. The wizard lets you set default storage settings, a default minimum communication setting, and a default refresh group for all snapshots in a new snapshot group.

## Registering a Snapshot Group at its Master Site

When you create a snapshot group, Oracle automatically attempts to register the group at the master site. Should registration fail for any reason, you can manually register a snapshot group at its master site using the replication management API.

**API Equivalent:** DBMS_REPCAT.REGISTER_SNAPSHOT_REPGROUP

## Unregistering a Snapshot Group at its Master Site

When you delete a snapshot group or switch a snapshot group's master site, Oracle automatically attempts to unregister the group at its former master site. Should this process fail for any reason, you can manually unregister a snapshot group at a master site using the replication management API.

**API Equivalent:** DBMS_REPCAT. UNREGISTER_SNAPSHOT_REPGROUP

## Adding Objects to a Snapshot Group

To add objects to a snapshot group:

1. Click the target snapshot group at the snapshot site.
2. Click the **Add to group** toolbar button.
3. Click **Add Objects to a Snapshot Group**.
4. Use the snapshot group wizard to add selected objects from the master group.

5. Click **Next**.
6. If you selected one or more master tables in Step 5, you can use the wizard to specify individual settings for corresponding snapshots.
7. Click **Finish** to have the wizard add the new objects to the snapshot group.

**API Equivalent:** DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT

## Altering Objects in a Snapshot Group

To alter the definition of a replication object in a snapshot group, you should *always* use Replication Manager (or an equivalent API call). Use of Enterprise Manager or a SQL DDL command (for example, ALTER TABLE) to alter an object in a replicated environment can create inconsistencies.

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

You cannot alter the definition of nonsnapshot objects in a snapshot group. For more information about altering individual snapshots in a snapshot group, see "Altering a Snapshot" on page 4-20.

## Deleting Objects from a Snapshot Group

To remove objects from a snapshot group:

1. Click the target snapshot group at the snapshot site.
2. Select the target objects to remove from the group.
3. Click the **Delete** toolbar button.

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

**API Equivalent:** DBMS_REPCAT.DROP_SNAPSHOT_REPOBJECT

## Editing a Snapshot Group

To edit a snapshot group:

1. Click the target snapshot group at the snapshot site.
2. Click the **Properties** toolbar button.

The **Edit Snapshot Group** property sheet has the following pages: **General** and **Objects**.

- Use the **General** page to edit the target snapshot group's master site and propagation mode to the master site.
- Use the **Objects** page to add and remove objects in the snapshot group.

**API Equivalent:** DBMS_REPCAT.SWITCH_SNAPSHOT_MASTER, DBMS_REPCAT.ALTER_SNAPSHOT_PROPAGATION

### Deleting a Snapshot Group

To delete a snapshot group:

1. Click the target snapshot group at the snapshot site.
2. Click the **Delete** toolbar button.

**Note:** When you drop a snapshot group, you can also decide whether to drop the group's objects from the database. If you choose not to drop a group's objects when you drop the group, the objects remain in the database at the snapshot site. Snapshots that remain after dropping the corresponding snapshot group persist as ungrouped snapshots and will be refreshed as long as they remain in a refresh group. However, Oracle does not forward to the master site changes made to an ungrouped updatable snapshot.

**API Equivalent:** DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP

## Managing Snapshots

A table snapshot is a local copy of table data that originates from one or more remote master tables. Applications can query the data in a read-only table snapshot, but cannot insert, update, or delete rows in the snapshot. However, applications can query, insert, update, or delete the rows in an updatable snapshot.

Oracle's data replication facility supports independent table snapshots as well as simple snapshots that are part of a snapshot group. Consider the following issues when deciding whether to create a new snapshot as part of a snapshot group:

- A snapshot group can contain simple, read-only and updatable snapshots.
- A snapshot group does not support complex read-only snapshots. Oracle creates all complex read-only snapshots independent of any snapshot group.

Replication Manager has many features that help you create and manage snapshots in an advanced replication environment. The following sections explain more about managing snapshots.

**Note:** Replication Manager is not designed to manage read-only snapshots in a basic replication environment. See Chapter 2, "Using Basic Replication" for more information about creating and managing a basic replication environment that uses read-only snapshots.

## Creating a Snapshot

Before you create a new simple snapshot, make sure that the following preliminary tasks have been completed.

- If you are creating a simple snapshot with fast refresh, create the necessary snapshot log for the master table before creating the snapshot.
- Create a refresh group that snapshots can use for automatic refreshes.

To create a read-only snapshot independent of a snapshot group:

1. Click anywhere in the open database connection that connects to the snapshot site.
2. Click the **Create New** toolbar button.
3. Click **New Snapshot**.

Replication Manager's snapshot wizard then guides you through the creation of a new snapshot.

**Note:** When you create an updatable snapshot using Replication Manager, the snapshot wizard always creates the new snapshot as part of the appropriate snapshot group.

To create a read-only snapshot or updatable snapshot as part of a snapshot group

1. Click the target snapshot group at the snapshot site.
2. Click the **Add to Group** toolbar button.
3. Click **Add Objects to a Snapshot Group**.

Replication Manager's snapshot group wizard then guides you through the process of creating new snapshot group objects, including snapshots. See "Creating a Snapshot Group" on page 4-12 for information on settings for the snapshot group wizard.

**API Equivalent:** DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT

### Datatype Considerations for Snapshots

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

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

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

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

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

### Creating Updatable Snapshots with a WHERE Clause

You can create an updatable snapshot with a WHERE clause to make the snapshot reflect a subset of the rows in its master table. For example, consider the following workflow environment where the updatable snapshots of the ORDERS master table reflect different sets of orders depending on the STATUS value of orders at the master site.

**At the ORDER Database:**

```
CREATE SNAPSHOT sales.orders FOR UPDATE
 AS SELECT * FROM sales.orders@dbs1.acme.com
  WHERE status = 'SHIPPABLE';
```

**At SHIP Database:**

```
CREATE SNAPSHOT sales.orders FOR UPDATE
  AS SELECT * FROM sales.orders@dbs1.acme.com
  WHERE status = 'BILLABLE';
```

**At ACCT Database:**

```
CREATE SNAPSHOT sales.orders FOR UPDATE
  AS SELECT * FROM sales.orders@dbs1.acme.com
  WHERE status = 'COMPLETE';
```

Oracle does not restrict applications from updating the columns that define the selection criteria for the result set of an updatable snapshot. For example, an application connected to the ORDER database could change the status of an order to "BILLABLE". Because of this behavior, it is possible for an updatable snapshot to contain rows that no longer match the selection criteria of the snapshot, at least until the snapshot is refreshed. For many applications, this behavior is acceptable.

Alternatively, when you want the rows of an updatable snapshot always to match the selection criteria for the snapshot, define a view on the snapshot or snapshot's base table with a CHECK constraint.

### Minimizing Data Propagation

The **Minimize Communication** setting found in some Replication Manager wizard pages and property sheets lets you determine how much data snapshot sites must transfer to perform conflict detection for an updatable snapshot (primary key snapshots only). When you use the default setting, to minimize communication, Oracle propagates only the new values for updated columns plus the old values of the primary key and the columns in each updated column group. The following Replication Manager components provide access to the **Minimize Communication** setting:

- Snapshot group wizard.
- Snapshot wizard.
- Edit Replication Object property sheet.

**Note:** The default setting, to minimize communication, is valid only for Oracle8 databases. When you base an updatable snapshot on a master table in an Oracle7 release 7.3 database, you must disable the Minimize Communication setting. When disabled, Oracle propagates the old and new values of all columns in a row when any column in the row is updated. This is the behavior expected by Oracle7 release 7.3.

**Additional Information:** See "Minimizing Data Propagation for Update Conflict Resolution" on page 5-40.

## Creating Snapshots with Subqueries

You can create a new snapshot as a subquery snapshot when using either the snapshot group wizard or the snapshot wizard.

To create a subquery snapshot when using the snapshot group wizard

1. Click the snapshot in the **Snapshots in the group** list of the **Individual Snapshot Settings** page of the snapshot group wizard.
2. Click the **Where clause** option, then click the corresponding **Edit** button.
3. Type the subquery for the new snapshot.

To create a subquery snapshot when using the snapshot wizard, type the subquery for the new snapshot in the **Where Clause** page of the snapshot wizard.

**Additional Information:** See "Creating Snapshots with Subqueries" on page 2-16 for more information about subquery snapshots.

**Additional Information:** See "Creating Updatable Snapshots with a WHERE Clause" on page 4-18 for more information about specifying a WHERE clause for a snapshot.

## Regenerating Replication Support for an Updatable Snapshot

To generate support for an updatable snapshot:

1. Click the target snapshot.
2. Click the **Properties** toolbar button.
3. Click **Generate Support** on the **General** page of the **Edit Replication Object** property sheet.

**API Equivalent:** DBMS_REPCAT.GENERATE_SNAPSHOT_SUPPORT

## Altering a Snapshot

The following sections explain how to alter a snapshot in an advanced replication environment.

### Editing a Snapshot's Storage Settings

To edit a snapshot's storage settings:

1. Click the target snapshot.
2. Click the **Properties** toolbar button.

3. Modify the settings of the **Tablespace and Extents** page of the **Edit Snapshot** property sheet (for snapshots independent of a group) or **Edit Replication Object** property sheet (for snapshots within a group).

### Manipulating a Snapshot's Base Table

Do not manipulate, modify, add to, or subtract from, the data in the base table of a snapshot. You can declare integrity constraints, such as referential or uniqueness constraints, for the base table of a snapshot. However, such constraints must be configured for deferred constraint checking.

You can also define PL/SQL triggers on the base table of a snapshot. However, such triggers must be coded so that they do **not** fire during snapshot refresh. Triggers that fire during snapshot refresh may generate unexpected results.

**Additional Information:** See "Triggers and Replication" on page 7-34.

### Altering a Snapshot Definition

Never use Enterprise Manager or a SQL DDL command (for example, ALTER TABLE) to alter a snapshot definition. To alter the definition of a snapshot, drop the snapshot and then re-create it.

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

## Deleting a Snapshot

To remove a snapshot from a snapshot group:

1. Click the target snapshot group at the snapshot site.
2. Select the target snapshots to remove from the group.
3. Click the **Delete** toolbar button.

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

**API Equivalent:** DBMS_REPCAT.DROP_SNAPSHOT_REPOBJECT

# Managing Refresh Groups

To preserve referential integrity and transaction consistency among the table snapshots of several related master tables, Oracle organizes and refreshes individual snapshots as part of a refresh group. After refreshing all of the snapshots in a refresh group, the data of all snapshots in the group corresponds to the same transaction-consistent point in time. Replication Manager has many features that help you create and manage refresh groups in an advanced replication environment. The following sections explain more about managing refresh groups.

**Additional Information:** See "Creating Refresh Groups" on page 2-24 for more information about creating and managing refresh groups.

## Creating a Refresh Group

To create a new refresh group for a snapshot site:

1. Click anywhere in the database connection that connects to the snapshot site.
2. Click the **Create New** toolbar button.
3. Click **New Refresh Group**.

The **Create Refresh Group** property sheet has three pages: **General**, **Snapshots**, and **Scheduling**.

- Use the **General** page to specify the name and owner for the group.
- Use the **Snapshots** page to add snapshots to the refresh group.
- Use the **Scheduling** page to configure the refresh group's refresh settings.

**Note:** To refresh a snapshot, the user account of the database link used by the snapshot must have SELECT privileges on the master base table and, for fast refreshes, on the corresponding snapshot log.

**API Equivalent:** DBMS_REFRESH.MAKE

## Adding Snapshots to a Refresh Group

To add one or more snapshots to a refresh group:

1. Click on a snapshot group or the **Ungrouped Snapshots** folder at the snapshot site.
2. Drag and drop selected snapshots into the target refresh group at the same snapshot site.

**API Equivalent:** DBMS_REFRESH.ADD

## Deleting Snapshots from a Refresh Group

To remove one or more snapshots from a refresh group:

1. Click the target refresh group at the snapshot site.
2. Select the target objects to remove from the group.
3. Click the **Delete** toolbar button.

**API Equivalent:** DBMS_REFRESH.SUBTRACT

## Changing Refresh Settings for a Snapshot Group

To edit a snapshot group's refresh settings:

1. Click the target refresh group at the snapshot site.
2. Click the **Properties** toolbar button.
3. Click the **Scheduling** page of the **Edit Refresh Group** property sheet.
4. Specify the new refresh settings for the refresh group.

**API Equivalent:** DBMS_REFRESH.CHANGE

## Manually Refreshing a Group of Snapshots

To force the immediate refresh of a refresh group:

1. Click the target refresh group at the snapshot site.
2. Click the **Properties** toolbar button.
3. Click the **General** page of the **Edit Refresh Group** property sheet.
4. Click **Refresh Now**.

**API Equivalent:** DBMS_REFRESH.REFRESH

## Deleting a Refresh Group

To delete a refresh group:

1. Click the target refresh group at the snapshot site.
2. Click the **Delete** toolbar button.

**Note:** After you drop a refresh group, Oracle no longer refreshes the group's orphaned snapshots automatically. To refresh the snapshots, you must add them to another snapshot refresh group or refresh them manually.

**API Equivalent:** DBMS_REFRESH.DESTROY

## Other Snapshot Site Administration Issues

The preceding sections of this chapter explained the most commonly performed administrative procedures that involve snapshot sites. For additional information on less commonly performed administrative procedures for snapshot sites, see "Advanced Management of Master and Snapshot Groups" on page 6-2.

## Data Dictionary Views

In addition to using Replication Manager to view information about an snapshot site in an advanced replication environment, you can also query the following data dictionary views:

**At the Snapshot Site:**

- DBA_SNAPSHOTS
- DBA_REFRESH
- DBA_REFRESH_CHILDREN

**At the Master Site:**

- DBA_REGISTERED_SNAPSHOT
- DBA_SNAPSHOT_LOGS

# 5

# Conflict Resolution

This chapter covers the following topics:

- Introduction to Replication Conflicts.
- Overview of Conflict Resolution Configuration.
- Configuring Update Conflict Resolution.
- Configuring Uniqueness Conflict Resolution.
- Configuring Delete Conflict Resolution.
- Guaranteeing Data Convergence.
- User-Defined Conflict Resolution Methods.
- User-Defined Conflict Notification Methods.
- Viewing Conflict Resolution Information.

**Note:** This chapter has examples of how to use the Oracle Replication Manager tool to manage conflict resolution in an advanced replication system. Each section also lists equivalent replication management API procedures for your reference. For complete information about Oracle's replication management API, see Chapter 9, “Replication Management API Reference”.

# Introduction to Replication Conflicts

Replication conflicts can occur in an advanced replication environment that permits concurrent updates to the same data at multiple sites. For example, when two transactions originating from different sites update the same row at nearly the same time, a conflict can occur. When you configure an advanced replication environment, you must consider whether replication conflicts can occur. If your system design permits replication conflicts and a conflict occurs, the system data does not converge until the conflict is resolved in some way.

In general, your first choice should always be to design a replicated environment that avoids the possibility of conflicts. Using several techniques, most system designs can avoid conflicts in all or a large percentage of the data that you replicate. However, many applications require that some percentage of data be updatable at multiple sites. If this is the case, you must then address the possibility of replication conflicts.

The next few sections introduce general information about replication conflicts, how to design an advanced replication system with replication conflicts in mind, how you can avoid replication conflicts in your replicated system design, and how Oracle can detect and resolve conflicts in designs where conflict avoidance is not possible.

## Understanding Your Data and Application Requirements

When you design any type of database application and supporting database, it is critical that you understand the requirements of the application before you begin to build the database or the application itself. For example, each application should be modular, with clearly defined functional boundaries and dependencies (for example, order-entry, shipping, billing). Furthermore, you should normalize supporting database data to reduce the amount of hidden dependencies between modules in the application system.

In addition to basic database design practices, there are additional requirements that you must investigate when building a database that operates in an advanced replication environment. Start by considering the general requirements of applications that will work with replicated data. For example, some applications might work fine with basic read-only table snapshots, and as a result, can avoid the possibility of replication conflicts altogether. Other applications might require that most of the replicated data be read-only and a small fraction of the data (for example, one or two tables or even one or two columns in a specific table) be 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.

### Some Examples

To better understand how to design a replicated database system with conflicts in mind, consider the following environments where conflict detection and resolution is feasible in some cases but not possible in others:

- Conflict resolution is often not possible in reservation systems where multiple bookings for the same item are not allowed. For example, when reserving specific seats for a concert, different agents accessing different replicas of the reservation system cannot book the same seat for multiple customers because there is no way to resolve such a conflict.
- Conflict resolution is often possible in customer management systems. For example, salespeople can maintain customer address information at different databases in a replicated environment. Should a conflict arise, the system can resolve the conflicting updates by applying the most recent update to a record.

## Types of Replication Conflicts

Advanced Replication includes facilities for detecting and resolving three types of conflicts: update conflicts, uniqueness conflicts, and delete conflicts.

### Update Conflicts

An *update conflict* occurs when the replication of an update to a row conflicts with another update to the same row. Update conflicts can happen when two transactions, originating from different sites, update the same row at nearly the same time.

### Uniqueness Conflicts

A *uniqueness conflict* occurs when the replication of a row attempts to violate entity integrity (a PRIMARY KEY or UNIQUE constraint). For example, consider what happens when two transactions originate from two different sites each inserting a row into a respective table replica with the same primary key value. In this case, replication of the transactions causes a uniqueness conflict.

### Delete Conflicts

A *delete conflict* occurs when two transactions originate from different sites, with one transaction deleting a row that the other transaction updates or deletes.

## Avoiding Conflicts

If application requirements permit, you should first design an advanced replication system that avoids the possibility of replication conflicts altogether. The next few sections briefly suggest several techniques that you can use to avoid some or all replication conflicts.

### Primary Site and Dynamic Site Ownership Data Models

One way you can avoid the possibility of replication conflicts is to limit the number of sites in the system with simultaneous update access to the replicated data. Two replicated data ownership models support this approach: primary site ownership and dynamic site ownership.

**Primary Site Ownership** Primary ownership is the replicated data model that basic read-only replication environments support. Primary ownership prevents all replication conflicts, because only a single server permits update access to a set of replicated data.

Rather than control the ownership of data at the table level, applications can employ horizontal and vertical partitioning to establish more granular static ownership of data. For example, applications might have update access to specific columns or rows in a replicated table on a site-by-site basis.

**Additional Information:** For more information about Oracle's basic, read-only replication features, see Chapter 2.

**Dynamic Site Ownership** The dynamic ownership replicated data model is less restrictive than primary site ownership. With dynamic ownership, capability to update a data replica moves from site to site, still ensuring that only one site provides update access to specific data at any given point in time. A workflow system clearly illustrates the concept of a dynamic ownership. For example, related departmental applications can read the status code of a product order, for example, ENTERABLE, SHIPPABLE, BILLABLE, to determine when they can and cannot update the order.

**Additional Information:** For more information about using dynamic ownership data models, see "Using Dynamic Ownership Conflict Avoidance" on page 7-27.

### Avoiding Specific Types of Conflicts

When both primary site ownership and dynamic ownership data models are too restrictive for your application requirements, you must use a shared ownership data model. Even so, typically you can use some simple strategies to avoid specific types of conflicts.

**Avoiding Uniqueness Conflicts** It is quite easy to configure an advanced replication environment to prevent the possibility of uniqueness conflicts. For example, you can create replica sequences at each site so that each sequence generates a mutually exclusive set of sequence numbers; however, this solution can become problematic as the number of sites increase or the number of entries in the replicated table grows. Alternatively, you can allow each site's replica sequences to use the full range of sequence values and include a unique site identifier as part of a composite primary key.

**Avoiding Delete Conflicts** Delete conflicts should always be avoided in all replicated data environments. In general, applications that operate within an asynchronous, shared ownership data model should not delete rows using DELETE statements. Instead, applications can mark rows for deletion and then configure the system to periodically purge logically deleted rows using procedural replication.

**Avoiding Update Conflicts** After trying to eliminate the possibility of uniqueness and delete conflicts in an advanced replication system, you should also try to limit the number of update conflicts that are possible. However, in a shared ownership data model, update conflicts cannot be avoided in all cases. If you cannot avoid all update conflicts, you must understand exactly what types of replication conflicts are possible and then configure the system to resolve conflicts when they occur.

## Conflict Detection at Master Sites

Each master site in an advanced replication system automatically detects and resolves replication conflicts when they occur. For example, when a master site pushes its deferred transaction queue to another master site in the system, the remote procedures being called at the receiving site detect replication conflicts automatically, if any.

When a snapshot site pushes deferred transactions to its corresponding master site, the receiving master site performs conflict detection and resolution. A snapshot site refreshes its data by performing snapshot refreshes. The refresh mechanism ensures that, upon completion, the data at a snapshot is the same as the data at the corresponding master, including the results of any conflict resolution; therefore, it

is not necessary for a snapshot site to perform work to detect or resolve replication conflicts.

### How Oracle Detects Different Types of Conflicts

The receiving master site in an advanced replication system detects update, uniqueness, and delete conflicts as follows:

- The receiving site detects an update conflict if there is any difference between the old values of the replicated row (the value before the modification) and the current values of the same row at the receiving site.
- The receiving site detects a uniqueness conflict if a uniqueness constraint violation occurs during an INSERT or UPDATE of a replicated row.
- The receiving site detects a delete conflict if it cannot find a row for an UPDATE or DELETE statement because the primary key of the row does not exist.

**Note:** To detect and resolve an update conflict for a row, the propagating site must send a certain amount of data about the new and old versions of the row to the receiving site. For maximum performance, tune the amount of data that Oracle uses to support update conflict detection and resolution. For more information, see "Minimizing Data Propagation for Update Conflict Resolution" on page 5-40.

### Identifying Rows During Conflict Detection

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

**Warning:** Do not permit applications to update the identity columns of a table. This ensures that Oracle can identify rows and preserve the integrity of replicated data.

## Conflict Resolution

When replication conflicts occur at a receiving master site, you must resolve them to ensure that the data throughout the system eventually converges. *Data convergence* means that all sites managing replicated data will ultimately agree on a set of matching information. If replication conflicts happen and you neglect to resolve them, the replicated data at various sites remains inconsistent. Furthermore,

there can be undesirable cascading affects. An inconsistency can create additional conflicts, which create additional inconsistencies, and so on.

If you cannot avoid all types of replication conflicts in your system, you can configure the system to use Oracle's automatic conflict resolution features. The following sections explain more about Oracle's conflict resolution features for each type of replication conflict.

### Automatic versus Manual Conflict Resolution

You should always use Oracle's automatic conflict resolution features to resolve conflicts when they occur. When you do not configure automatic conflict resolution for replicated tables, Oracle simply logs conflicts at each site. In this case, you are forced to resolve conflicts manually to preserve the integrity of replicated data. Manual conflict resolution can be challenging to perform. Furthermore, delays in performing manual conflict resolution can leave inconsistencies in the data that can create cascading effects mentioned in the previous section.

### Update Conflict Resolution and Column Groups

Oracle uses column groups to detect and resolve update conflicts. A column group is a logical grouping of one or more columns in a replicated table. Every column in a replicated table is part of a single column group. When configuring replicated tables at the master definition site, you can create column groups and then assign columns and corresponding conflict resolution methods to each group.

**Ensuring Data Integrity with Multiple Column Groups** Having column groups allows you to designate different methods of resolving conflicts for different types of data. For example, numeric data is often suited for an arithmetical resolution method, and character data is often suited for a timestamp resolution method. However, when selecting columns for a column group, it is important to group columns wisely. If two or more columns in a table must remain consistent with respect to each other, place the columns within the same column group to ensure data integrity. For example, if the zip code column in a customer table uses one resolution method while the city column uses a different resolution method, the sites could converge on a zip code that does match the city. Therefore, all components of an address should typically be within a single column group so that conflict resolution is applied to the address as a unit.

**Shadow Column Groups** By default, every replicated table has a shadow column group. A table's shadow column group contains all columns that are not within a specific column group. You *cannot* assign conflict resolution methods to a table's

shadow group. Therefore, make sure to include a column in a column group when conflict resolution is necessary for the column.

### Uniqueness Conflict Resolution

In most cases, you should build an advanced replication system and corresponding applications so that uniqueness conflicts are not possible. However, if you cannot avoid uniqueness conflicts, you can assign one or more conflict resolution methods to a PRIMARY KEY or UNIQUE constraint in a replicated table to resolve uniqueness conflicts when they occur. Oracle provides a few prebuilt uniqueness conflict resolution methods. However, you will typically want to use these methods with conflict notification so that you can validate the accuracy of resolved uniqueness conflicts.

### Delete Conflict Resolution

You should always design advanced replication environments to avoid delete conflicts. If avoiding delete conflicts is too restrictive for an application design, you can write custom delete conflict resolution methods and assign them to replicated tables. Oracle does not offer any prebuilt delete conflict resolution methods. See "User-Defined Conflict Resolution Methods" on page 5-46 for more information about writing your own conflict resolution methods.

## Conflict Resolution Methods

To resolve replication conflicts automatically, you can assign one or more conflict resolution methods. Oracle has many prebuilt conflict resolution methods that you can use to resolve conflicts. If necessary, you can build your own methods to resolve conflicts. The following sections explain more about prebuilt and custom conflict resolution methods.

### Prebuilt Update Conflict Resolution Methods

Oracle offers the following prebuilt methods for update conflicts that you can assign to a column group.

- Overwrite and discard value.
- Minimum and maximum value.
- Earliest and latest timestamp value.
- Additive and average value.
- Priority groups and site priority.

**Additional Information:** For complete information about each prebuilt update conflict resolution method, “Prebuilt Update Conflict Resolution Methods” on page 5-16.

Oracle’s prebuilt update conflict resolution methods have varying characteristics in their ability to converge replicated data. For example, the additive conflict resolution method can converge replicated data managed by more than two master sites, but the earliest timestamp method cannot.

**Additional Information:** For specific information about the data convergence property of each prebuilt conflict resolution method, “Guaranteeing Data Convergence” on page 5-37.

### Prebuilt Uniqueness Conflict Resolution Methods

Oracle offers the following prebuilt uniqueness conflict resolution methods that you can assign to PRIMARY KEY and UNIQUE constraints.

- Append site name to duplicate value.
- Append sequence to duplicate value.
- Discard duplicate value.

Oracle’s prebuilt uniqueness conflict resolution methods do not converge the data in a replicated environment; they simply provide techniques for resolving PRIMARY KEY and UNIQUE constraint violations. Therefore, when you use one of Oracle’s uniqueness conflict resolution methods to resolve conflicts, you should also employ a notification mechanism to alert you to uniqueness conflicts when they happen and then manually converge replicated data, if necessary.

**Additional Information:** For complete information about Oracle’s prebuilt uniqueness conflict resolution methods, “Prebuilt Uniqueness Resolution Methods” on page 5-34.

### Restrictions of Prebuilt Conflict Resolution Methods

Oracle’s prebuilt conflict resolution methods do not support the following situations.

- Delete conflicts.
- Changes to primary key (or identity key) columns.
- NULLs in the columns that you designate to resolve the conflict.
- Referential integrity constraint violations.

For these situations, you must either provide your own conflict resolution method or determine a method of resolving error transactions manually.

### Custom Conflict Resolution and Notification Methods

In addition to using Oracle's prebuilt conflict resolution methods, you can also consider using conflict logging and conflict notification to compliment conflict resolution. Oracle lets you configure a replicated table to call user-defined methods that record conflict information or notify you when Oracle cannot resolve a conflict. You can configure column groups, constraints, and replicated tables to notify you for all conflicts, or for only those conflicts that Oracle cannot resolve.

**Additional Information:** For more information about writing your own conflict resolution and notification methods, see "User-Defined Conflict Resolution Methods" on page 5-46.

### Using Multiple Conflict Resolution Methods

Indicating multiple conflict resolution methods for a column group allows Oracle to resolve a conflict in different ways should others fail to resolve the conflict. When trying to resolve a conflict, Oracle executes each group's methods in the order that you list. The algorithm that Oracle uses to resolve update conflicts is as follows.

1. Starting with the first column group, the receiving master site examines each field in the group to determine if it has changed and, if so, if there is a conflict between the old, new, and current values.
2. If no conflict occurred, Oracle can continue to the next column group. If a conflict occurred, Oracle calls the conflict resolution method with the lowest assigned sequence number for the column group.
3. If the conflict resolution method successfully resolves the conflict, Oracle holds the appropriate values for the columns pending determination of status.
4. If the method cannot resolve the conflict, Oracle continues with the next method until it can resolve the conflict or when no more methods are available.
5. After evaluating all column groups (including the shadow column group) and successfully resolving any conflicts, Oracle stores the new values for the columns.
6. If Oracle is unable to resolve any conflict using assigned methods, the receiving site logs the entire transaction as an error transaction in the site's replication catalog and does not change the values in the local row.

You should use multiple conflict resolutions methods for the following reasons:

- To employ alternative conflict resolution methods when the preferred method cannot resolve a conflict.
- To receive automatic notification of replication conflicts when they occur.

The following sections explain more about each issue.

**Note:** You can also assign multiple conflict resolution methods to a PRIMARY KEY or UNIQUE constraint to resolve uniqueness conflicts, and to a replicated table to resolve delete conflicts.

**Using Multiple Conflict Resolution Methods for Backup** In certain situations, the preferred conflict resolution method that you set for a column group, constraint, or table might not always succeed. If this is at all possible, you should specify a sequence of one or more alternative methods to increase the possibility that Oracle can perform conflict resolution without the need for manual resolution.

Some system-defined conflict resolution methods cannot guarantee successful resolution of conflicts in all circumstances. For example, the latest timestamp update conflict resolution method uses a special timestamp column to determine and apply the most recent change. In the unlikely event that the row at the originating site and the row at another site change at precisely the same second, the latest timestamp method cannot resolve the conflict because Oracle stores time related information at the granularity of a second. If you declare a backup update conflict resolution method such as site priority, Oracle might be able to resolve the conflict automatically.

**Using Multiple Conflict Resolution Methods for Notification** Another reason to use multiple conflict resolution routines is to call a user-defined method that records conflict information or notifies you when Oracle cannot resolve a conflict. For example, you might decide to configure a PRIMARY KEY constraint to call a custom conflict notification method first and then resolve conflicts using the append site name uniqueness conflict resolution method.

**Additional Information: F**or more information about conflict notification, “User-Defined Conflict Notification Methods” on page 5-50

# Overview of Conflict Resolution Configuration

If you decide that conflict resolution is necessary in your advanced replication system, you must first design your conflict resolution strategy and then implement it when you create replicated tables. The following sections provide you with an overview of the steps necessary to complete each stage of conflict resolution configuration.

## Design and Preparation Guidelines for Conflict Resolution

Use the following guidelines to design and prepare for a conflict resolution strategy.

- Analyze your data to determine which column groups are appropriate for update conflict resolution, and which conflict resolution methods are appropriate for each column group. If you cannot avoid uniqueness and delete conflicts, you must also plan for these types of conflict resolution.
- Some of Oracle's prebuilt update conflict resolution methods require unique preparatory steps before use. For example:
  - When you use the earliest or latest timestamp update conflict resolution methods, the replicated table must have a DATE column that Oracle can use for timestamp comparison. Additionally, you might want to add a timestamp maintenance trigger to the table to address time synchronization issues. For more information about timestamp resolution, see "Earliest and Latest Timestamp" on page 5-18.
  - If any column groups in a replication table will use site priority or priority groups for conflict resolution, define the priority levels for each site or value. For more information about configuring priority groups and site priority groups, see "Using Priority Groups for Update Conflict Resolution" on page 5-22 and "Using Site Priority for Update Conflict Resolution" on page 5-28, respectively.
- If necessary, prepare for conflict notification methods.
  - Create a table to hold conflict notification information at each master site.
  - Create the PL/SQL procedure to record conflict notification in the table.
  - Add the custom defined conflict resolution methods to the package or add methods to automate e-mail notification (optional).

  For more information about configuring conflict notification, see "User-Defined Conflict Notification Methods" on page 5-50.

### Implementing Conflict Resolution

After planning, use Oracle Replication Manager and Oracle's replication management API to configure conflict resolution for the replicated tables in a master group. In general, these steps include the following:

1. Suspend replication activity for the master group.
2. Configure conflict resolution for the replicated tables in the master group at the master definition site. For example, when configuring update conflict resolution for a table, use Replication Manager to create necessary column groups and assign update conflict resolution methods to the groups.
3. Regenerate replication support for the replicated tables or for all objects in the master group after you finish configuring conflict resolution.
4. Resume replication activity for the master group once you are finished with all modifications and test your conflict resolution configuration.

The following sections explain how to configure update, uniqueness, and delete conflict resolution.

## Configuring Update Conflict Resolution

In a typical advanced replication environment, uniqueness and delete conflicts are not possible, and update conflicts, therefore, require the most attention during system design and configuration. Oracle's advanced replication facility uses *column groups* to detect and resolve update conflicts. The following sections explain how to configure column groups for a replicated table and associate update conflict resolution methods for column groups.

### Creating a Column Group

After adding a table to a master group at the master definition site and while replication activity is suspended for the group, you can configure column groups for the table and establish conflict resolution.

1. Click on the target table in the master group at the master definition site.
2. Click the **Properties** toolbar button.
3. Click the **Conflict Resolution** page of the **Edit Replication Object** dialog.
4. Click **Add** (the upper button) to display the **Create Column Group** dialog and create a new column group for the target table. When you create a column

group, specify a name and an optional comment for the column group, as well as one or more columns as members for the group.

5. Click **OK** to create the new column group.
6. Choose the update conflict resolution methods for the column group. See "Prebuilt Update Conflict Resolution Methods" on page 5-16 for more information.

**API Equivalent:** DBMS_REPCAT.MAKE_COLUMN_GROUP

## Adding and Removing Columns in a Column Group

While replication activity is suspended for a master group, you can add or remove the columns in a column group for a table.

1. Click on the target table in the master group at the master definition site.
2. Click the **Properties** toolbar button.
3. Click the **Conflict Resolution** page of the **Edit Replication Object** dialog.
4. Click on the column group to edit.
5. Click **Edit** (the upper button) to display the **Edit Column Group** dialog.
6. To add columns to the group, select the columns from the list of available columns and click **Add**. To remove columns from the group, select the columns from the list of columns in the group and click **Remove.**

**API Equivalent:** DBMS_REPCAT.ADD_GROUPED_COLUMN, DBMS_REPCAT.DROP_GROUPED_COLUMN

## Dropping a Column Group

While replication activity is suspended for a master group, you can drop a column group for a table.

1. Click on the target table in the master group at the master definition site.
2. Click the **Properties** toolbar button.
3. Click the **Conflict Resolution** page of the **Edit Replication Object** dialog.
4. Click on the column group to drop.
5. Click **Remove** (the upper button) to drop the target column group from the table.

**API Equivalent:** DBMS_REPCAT.DROP_COLUMN_GROUP

## Managing a Group's Update Conflict Resolution Methods

While replication activity is suspended for a master group, you can use Oracle Replication Manager to assign, remove, and order the update conflict resolution methods for a column group of a replicated table.

1. Click on the target table in the master group at the master definition site.
2. Click the **Properties** toolbar button.
3. Click the **Conflict Resolution** page of the **Edit Replication Object** dialog.
4. Click on the target column group that you want to manage.

At this point, you can assign, remove, or order the update conflict resolution methods for the selected column group. The following sections explain each procedure.

**Note:** You must suspend replication activity before creating or editing conflict resolution for a table. Furthermore, all changes that you make using the **Conflict Resolution** page are immediately committed to the replication environment.

### Assigning an Update Conflict Resolution Method

To assign a new update conflict resolution method to the selected column group, click **Add (the** lower button) to display the **Add Update Resolution Method** dialog and add a new update conflict resolution method for the target column group.

**Note:** Certain update conflict resolution methods require that you complete some preparatory work before assigning them to a column group (for example, priority groups). See the sections later in this chapter that discuss each type of conflict resolution method and any special requirements necessary to use them.

**API Equivalent:** DBMS_REPCAT.ADD_UPDATE_RESOLUTION

### Removing an Update Conflict Resolution Method

To remove an update conflict resolution method from the selected column group, click the method to remove, then click **Remove** (the lower button) to remove the selected conflict resolution method from the column group.

**API Equivalent:** DBMS_REPCAT.DROP_UPDATE_RESOLUTION

### Ordering a Column Group's Update Conflict Resolution Methods

To order or reorder the application of conflict resolution methods for the selected column group, promote or demote the selected resolution method using the up and down arrow buttons, then click **Reorder** to apply the new order.

## Prebuilt Update Conflict Resolution Methods

The following sections explain Oracle's prebuilt methods that you can use to resolve update conflicts, including:

- Additive and average.
- Minimum and maximum.
- Earliest and latest timestamp.
- Overwrite and discard.
- Priority groups and site priority.

The following sections explain each prebuilt update conflict resolution method in detail.

**Note:** The conflict resolution methods that you assign need to ensure data convergence and provide results that are appropriate for how your business uses the data. For complete information about data convergence and Oracle's prebuilt conflict resolution methods, see "Guaranteeing Data Convergence" on page 5-37.

### Additive and Average

The *additive* and *average* methods work with column groups consisting of a single numeric column only.

- The additive method adds the difference between the old and new values at the originating site to the current value at the destination site.

  ```
  current value = current value + (new value - old value)
  ```

  The additive conflict resolution method provides convergence for any number of master sites.

- The average conflict resolution method averages the new column value from the originating site with the current value at the destination site.

  ```
  current value = (current value + new value)/2
  ```

  The average method cannot guarantee convergence if your replicated environment has more than one master. This method is useful for an environment with a single master site and multiple updatable snapshots.

### Minimum and Maximum

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

If the new value of the designated column is *less than* the current value, the column group values from the originating site are applied at the destination site (assuming that all other errors were successfully resolved for the row). If the new value of the designated column is greater than the current value, the conflict is resolved by leaving the current values of the column group unchanged.

**Note:** If the two values for the designated column are the same (for example, if the designated column was not the column causing the conflict), the conflict is not resolved, and the values of the columns in the column group remain unchanged. Designate a backup conflict resolution method to be used for this case.

The *maximum* value method is the same as the minimum value method, except that the values from the originating site are only applied if the value of the designated column at the originating site is *greater than* the value of the designated column at the destination site.

There are no restrictions on the datatypes of the columns in the column group. Convergence for more than two master sites is only guaranteed if:

- For the maximum method, the column value is always increasing.
- For the minimum method, the column value is always decreasing.

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

### Earliest and Latest Timestamp

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

The following example demonstrates an appropriate application of the latest timestamp update conflict resolution method:

1. A customer in Phoenix calls the local salesperson and updates her address information.
2. After hanging up the phone, the customer realizes that she gave the local salesperson the wrong postal code.
3. The customer tries to call the local salesperson with the correct postal code, but the salesperson cannot be reached.
4. The customer calls the headquarters, which is located in New York. The New York site, rather than the Phoenix site, correctly updates the address information.
5. The network connecting New York headquarters with the local Phoenix sales site goes down temporarily.
6. When the New York / Phoenix network connection comes back up, Oracle sees two updates for the same address, and detects a conflict at each site.
7. Using the *latest timestamp* method, Oracle selects the most recent update, and applies the address with the correct postal code.

The earliest timestamp method applies the changes from the site with the earliest timestamp, and the latest timestamp method applies the changes from the site with the latest timestamp.

**Note:** When you use a timestamp conflict resolution method, you should designate a backup method, such as *site priority*, to be called if two sites have the same timestamp.

**Other Configuration Issues** When you use timestamp resolution, you must carefully consider how time is measured on the different sites managing replicated data. For example, if a replicated environment crosses time zones, applications that use the system should convert all timestamps to a common time zone such as Greenwich

Mean Time (GMT). Furthermore, if two sites in a system do not have their system clocks synchronized reasonably well, timestamp comparisons might not be accurate enough to satisfy application requirements.

There are two ways to maintain timestamp columns if you use the EARLIEST or LATEST timestamp update conflict resolution methods.

- Each application can include logic to synchronize timestamps.
- You can create a trigger for a replicated table to synchronize timestamps automatically for all applications.

A clock counts seconds as an increasing value. Assuming that you have properly designed your timestamping mechanism and established a backup method in case two sites have the same timestamp, the *latest* timestamp method (like the maximum value method) guarantees convergence. The *earliest* timestamp method, however, *cannot* guarantee convergence for more than two masters.

**Additional Information:** For an example of a timestamp and site maintenance trigger, "Sample Timestamp and Site Maintenance Trigger" on page 5-31

### Overwrite and Discard

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

For example, if you have a single master site that you expect to be used primarily for queries, with all updates being performed at the snapshot sites, you might select the overwrite method. The overwrite and discard methods are also useful if:

- Your primary concern is data convergence.
- You have a single master site.
- There is no particular business rule for selecting one update over the other.
- You have multiple master sites and you supply a notification facility to notify the person who ensures that data is correctly applied, instead of logging the conflict in the DEFERROR view and leaving the resolution to your local database administrator.

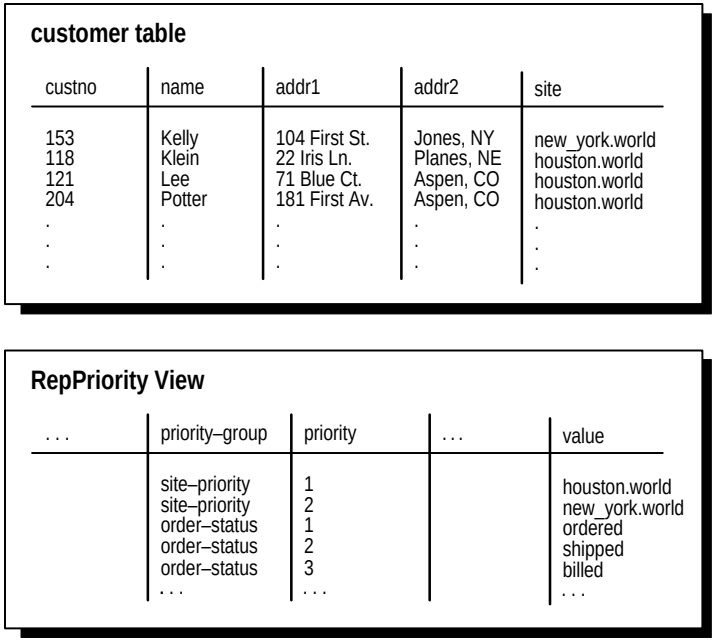
The overwrite method replaces the current value at the destination site with the new value from the originating site. Conversely, the discard method ignores the new value from the originating site.

### Priority Groups and Site Priority

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

As shown in Figure 5–1, the REPPRIORITY view displays the priority level assigned to each priority group member (value that the "priority" column can contain). You must specify a priority for all possible values of the "priority" column.

***Figure 5–1 Using Priority Groups***




The REPPRIORITY view displays the values of all priority groups defined at the current location. In the example shown in Figure 5–1, there are two different priority groups, site-priority and order-status. The CUSTOMER table is using the site-priority priority group.

Before you use Replication Manager to select the priority group method of update conflict resolution, you must designate which column in your table is the "priority" column.

**Additional Information:** To learn how to configure priority groups for update conflict resolution, “Using Priority Groups for Update Conflict Resolution” on page 5-22.

Site priority is a special kind of priority group. With site priority, the “priority” column you designate is automatically updated with the global database name of the site where the update originated. The REPPRIORITY view displays the priority level assigned to each database site. Site priority can be useful if one site is considered to be more likely to have the most accurate information. For example, in Figure 5–1, the New York site (priority value = 2) is corporate headquarters, while the Houston site (priority value = 1) is a sales office. Therefore, the headquarters office is considered more likely than the sales office to have the most accurate information about the credit that can be extended to each customer.

**Note:** The priority-group column of the REPPRIORITY view shows both the site-priority group and the order-status group.

When you are using site priority, convergence with more than two masters is not guaranteed. However, you can guarantee convergence with more than two masters when you are using priority groups if the value of the “priority” column is always increasing. That is, the values in the priority column correspond to an ordered sequence of events; for example: ordered, shipped, billed.

Similar to priority groups, you must complete several preparatory steps before using Replication Manager to select site priority conflict resolution for a column group.

Additional Information: To learn how to configure site priority, “Using Site Priority for Update Conflict Resolution” on page 5-28.

## Using Priority Groups for Update Conflict Resolution

To use the priority group method to resolve update conflicts, you must complete some special steps using Oracle's replication management API before using Replication Manager to assign the priority group resolution method to a column group. First, you must create a priority group. To create a priority group, do the following:

1. Define the name of the priority group and the datatype of the values in the group.
2. Define the priority level for each possible value of the "priority" column. This information is displayed in the REPPRIORITY view.

A single priority group can be used by multiple tables. Therefore, the name that you select for your priority group must be unique within a master group. The column corresponding to this priority group can have different names in different tables.

You must indicate which column in a table is associated with a particular priority group when you add the priority group conflict resolution method for the table. The priority group must therefore contain all possible values for all columns associated with that priority group.

For example, suppose that you have a replicated table, INVENTORY, with a column of type VARCHAR2, STATUS, that could have three possible values: ORDERED, SHIPPED, and BILLED. Now suppose that you want to resolve update conflicts based upon the value of the STATUS column. After suspending replication activity for the associated master group ACCT, complete the following steps at the master definition site to configure priority group resolution for the INVENTORY table.

1. Use Replication Manager to create a column group for the INVENTORY table that includes the STATUS column.
2. Create and populate the STATUS priority group that is associated with the master group ACCT. To do this, use the following API calls:

```
DBMS_REPCAT.DEFINE_PRIORITY_GROUP(
            gname                 => 'acct',
            pgroup                => 'status',
            datatype               =>'varchar2');
DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
           gname                  => 'acct',
          pgroup                 => 'status',
            value                 => 'ordered',
            priority              => 1);
DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
            sname                 => 'acct',
            pgroup                => 'status',
            value                 => 'shipped',
            priority              => 2);
DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
            sname                 => 'acct',
            pgroup                => 'status',
            value                 => 'billed',
            priority              => 3);
```

3. Use Replication Manager to designate the PRIORITY GROUP conflict resolution method for the target column group.

**Note:** Before creating or managing a priority group, use Replication Manager to suspend replication activity for the corresponding master group at the master definition site. After managing priority groups in any way, regenerate replication support for the associated replicated table before resuming replication activity for the master group.

The next several sections describe more about managing priority groups.

### Creating a Priority Group

Use the DEFINE_PRIORITY_GROUP procedure in the DBMS_REPCAT package to create a new priority group for a master group, as shown in the following example:

```
DBMS_REPCAT.DEFINE_PRIORITY_GROUP(
           gname     => 'acct',
           pgroup    => 'status',
           datatype  => 'varchar2');
```

This example creates a priority group called STATUS for the ACCT master group. The members of this priority group have values of type VARCHAR2.

**Additional Information:** The parameters for the DEFINE_PRIORITY_GROUP procedure are described in Table 9–122, and the exceptions are listed in Table 9–123.

### Adding Members to a Priority Group

There are several different procedures in the DBMS_REPCAT package for adding members to a priority group. These procedures are of the form ADD_PRIORITY_*type*, where *type* is equivalent to the datatype that you specified when you created the priority group:

- ADD_PRIORITY_CHAR
- ADD_PRIORITY_VARCHAR2
- ADD_PRIORITY_NUMBER
- ADD_PRIORITY_DATE
- ADD_PRIORITY_RAW
- ADD_PRIORITY_NCHAR
- ADD_PRIORITY_NVARCHAR2

The specific procedure that you must call is determined by the datatype of your "priority" column. You must call this procedure once for each of the possible values of the "priority" column.

The following example adds the value SHIPPED to the STATUS priority group:

```
DBMS_REPCAT.ADD_PRIORITY_VARCHAR2(
            gname     => 'acct',
            pgroup    => 'status',
            value     => 'shipped',
            priority    => 2);
```

**Additional Information:** The parameters for the ADD_PRIORITY_*datatype* procedures are described in Table 10–77 , and the exceptions are listed in Table 10–78 .

### Altering the Value of a Member

There are several different procedures in the DBMS_REPCAT package for altering the value of a member of a priority group. These procedures are of the form ALTER_PRIORITY_*type*, where *type* is equivalent to the datatype that you specified when you created the priority group:

- ALTER_PRIORITY_CHAR
- ALTER_PRIORITY_VARCHAR2
- ALTER_PRIORITY_NUMBER
- ALTER_PRIORITY_DATE
- ALTER_PRIORITY_RAW
- ALTER_PRIORITY_NCHAR
- ALTER_PRIORITY_NVARCHAR2

The procedure that you must call is determined by the datatype of your "priority" column. Because a priority group member consists of a priority associated with a particular value, these procedures enable you to change the value associated with a given priority level.

The following example changes the recognized value of items at priority level 2 from SHIPPED to IN_SHIPPING:

```
DBMS_REPCAT.ALTER_PRIORITY_VARCHAR2(
                gname        => 'acct',
                pgroup       => 'status',
                old_value    => 'shipped',
                new_value      => 'in_shipping');
```

**Additional Information**: The parameters for the ALTER_PRIORITY_*datatype* procedures are described in Table 9–89, and the exceptions are listed in Table 10–90 .

### Altering the Priority of a Member

Use the ALTER_PRIORITY procedure in the DBMS_REPCAT package to alter the priority level associated with a given priority group member. Because a priority group member consists of a priority associated with a particular value, this procedure lets you raise or lower the priority of a given column value. Members with higher priority values are given higher priority when resolving conflicts.

The following example changes the priority of items marked as IN_SHIPPING from level 2 to level 4:

```
DBMS_REPCAT.ALTER_PRIORITY(
            gname           => 'acct',
            pgroup          => 'status',
            old_priority    => 2,
            new_priority    => 4);
```

**Additional Information:** The parameters for the ALTER_PRIORITY procedure are described in Table 10–87 , and the exceptions are listed in Table 10–88 .

### Dropping a Member by Value

There are several different procedures in the DBMS_REPCAT package for dropping a member of a priority group by value. These procedures are of the form DROP_PRIORITY_*type*, where *type* is equivalent to the datatype that you specified when you created the priority group:

- DROP_PRIORITY_CHAR
- DROP_PRIORITY_VARCHAR2
- DROP_PRIORITY_NUMBER
- DROP_PRIORITY_DATE
- DROP_PRIORITY_RAW
- DROP_PRIORITY_NCHAR
- DROP_PRIORITY_NVARCHAR2

The procedure that you must call is determined by the datatype of your "priority" column.

In the following example, IN_SHIPPING is no longer a valid state for items in the STATUS priority group:

```
DBMS_REPCAT.DROP_PRIORITY_VARCHAR2(
          gname       => 'acct',
          pgroup      => 'status',
          value         => 'in_shipping');
```

**Additional Information:** The parameters for the DROP_PRIORITY_*datatype* procedures are described in Table 10–138, and the exceptions are listed in Table 10–139.

### Dropping a Member by Priority

Use the DROP_PRIORITY procedure in the DBMS_REPCAT package to drop a member of a priority group by priority level.

In the following example, IN_SHIPPING (which was assigned to priority level 4) is no longer a valid state for items in the STATUS priority group:

```
DBMS_REPCAT.DROP_PRIORITY(
          gname         => 'acct',
          pgroup        => 'status',
          priority_num    => 4);
```

**Additional Information**: The parameters for the DROP_PRIORITY procedure are described in Table 10–136 , and the exceptions are listed in Table 10–137 .

### Dropping a Priority Group

Use the DROP_PRIORITY_GROUP procedure in the DBMS_REPCAT package to drop a priority group for a given master group. For example, the following call drops the STATUS priority group:

```
DBMS_REPCAT.DROP_PRIORITY_GROUP(
          gname      => 'acct',
          pgroup       => 'status');
```

**Attention:** Before dropping a priority group, you remove the priority group update resolution method from all column groups that depend on the priority group. Query the REPRESOLUTION view to determine which column groups depend on a priority group.

**Additional Information:** The parameters for the DROP_PRIORITY_GROUP procedure are described in Table 10–140, and the exceptions are listed in Table 10–141.

## Using Site Priority for Update Conflict Resolution

Site priority is a specialized form of priority groups. Thus, many of the procedures associated with site priority behave similarly to the procedures associated with priority groups.

If you chose to use the site priority method to resolve update conflicts, you must first create a site priority group before you can use Replication Manager to add this conflict resolution method to a column group. Creation of a site priority group consists of two steps.

1. Define the name of the site priority group.
2. Add each site to the site priority group and define its priority level. This information is displayed in the REPPRIORITY view.

In general, you need only one site priority group for a master group. This site priority group can be used by any number of replicated tables. The next several sections describe how to manage site priority groups.

### Pre and Post Steps for Managing Site Priority

When configuring or managing site priority, keep in mind the following important order dependent operations.

- Before creating or managing a site priority group, use Replication Manager to suspend replication activity for the corresponding master group.
- You must manage site priority groups from the master definition site of the corresponding master group.
- After managing site priority groups in any way, use Replication Manager to regenerate replication support for the associated replicated table. Then you can use Replication Manager to resume replication activity for the master group.

### Creating a Site Priority Group

Use the DEFINE_SITE_PRIORITY procedure in the DBMS_REPCAT package to create a new site priority group for a master group, as shown in the following example:

```
DBMS_REPCAT.DEFINE_SITE_PRIORITY(
            gname     => 'acct',
            name        => 'site');
```

This example creates a site priority group called SITE for the ACCT object group.

**Additional Information:** The parameters for the DEFINE_SITE_PRIORITY procedure are described in Table 10–124, and the exceptions are listed in Table 10–125.

### Adding a Site to the Group

Use the ADD_SITE_PRIORITY_SITE procedure in the DBMS_REPCAT package to add a new site to a site priority group, as shown in the following example:

```
DBMS_REPCAT.ADD_SITE_PRIORITY_SITE(
          gname       => 'acct',
          name        => 'site',
          site        => 'hq.widgetek.com',
          priority      => 100);
```

This example adds the HQ site to the SITE group and sets its priority level to 100.

**Note:** The highest priority is given to the site with the highest priority value. Priority values do not have to be consecutive integers.

**Additional Information:** The parameters for the ADD_SITE_PRIORITY_SITE procedure are described in Table 10–79, and the exceptions are listed in Table 10–80.

### Altering the Priority Level of a Site

Use the ALTER_SITE_PRIORITY procedure in the DBMS_REPCAT package to alter the priority level associated with a given site, as shown in the following example:

```
DBMS_REPCAT.ALTER_SITE_PRIORITY(
           gname        => 'acct',
           name         => 'site',
           old_priority => 100,
           new_priority   => 200);
```

This example changes the priority level of a site in the SITE group from 100 to 200.

**Note:** The highest priority is given to the site with the highest priority value. Priority values do not have to be consecutive integers.

**Additional Information**: The parameters for the ALTER_SITE_PRIORITY procedure are described in Table 10–91, and the exceptions are listed in Table 10–92.

### Altering the Site Associated with a Priority Level

Use the ALTER_SITE_PRIORITY_SITE procedure in the DBMS_REPCAT package to alter the site associated with a given priority level, as shown in the following example:

```
DBMS_REPCAT.ALTER_SITE_PRIORITY_SITE(
          gname         => 'acct',
          name          => 'site',
          old_site      => 'hq.widgetek.com',
          new_site        => 'hq.widgetworld.com);
```

This example changes the global database name of the HQ site to HQ.WIDGETWORLD.COM, while its priority level remains the same.

**Additional Information:** The parameters for the ALTER_SITE_PRIORITY_ SITE procedure are described in Table 10–93, and the exceptions are listed in Table 10–94.

### Dropping a Site by Site Name

Use the DROP_SITE_PRIORITY_SITE procedure in the DBMS_REPCAT package to drop a given site, by name, from a site priority group, as shown in the following example:

```
DBMS_REPCAT.DROP_SITE_PRIORITY_SITE(
          gname    => 'acct',
          name     => 'site',
          site       => 'hq.widgetek.com');
```

This example drops the HQ site from the SITE group.

**Additional Information:** The parameters for the DROP_SITE_PRIORITY_SITE procedure are described in Table 10–144, and the exceptions are listed in Table 10–145.

### Dropping a Site by Priority Level

Use the DBMS_REPCAT.DROP_PRIORITY procedure described on page 5 - 27 to drop a site from a site priority group by priority level.

### Dropping a Site Priority Group

Use the DROP_SITE_PRIORITY procedure in the DBMS_REPCAT package to drop a site priority group for a given master group, as shown in the following example:

```
DBMS_REPCAT.DROP_SITE_PRIORITY(
          gname   => 'acct',
          name      => 'site');
```

In this example, SITE is no longer a valid site priority group.

**Attention:** Before calling this procedure, you must call the DROP_UPDATE_RESOLUTION procedure for any column groups in the master group that are using the SITE PRIORITY conflict resolution method with this site priority group. You can determine which column groups are affected by querying the REPRESOLUTION view.

**Additional Information:** The parameters for the DROP_SITE_PRIORITY procedure are described in Table 10–142, and the exceptions are listed in Table 10–143.

## Sample Timestamp and Site Maintenance Trigger

In either a trigger or in your application, you must implement the logic necessary to maintain the timestamp and site information. The following example trigger considers clock synchronization problems, but needs to be modified if the application crosses time zones.

Because the example trigger uses one of the generated procedures to check whether or not the trigger should actually be fired, it is necessary to generate replication support for the corresponding CUSTOMERS table before creating the trigger. This will also allow transactions on the CUSTOMERS table to be propagated.

```
dbms_repcat.generate_replication_support(sname => 'SALES',
                                          oname => 'CUSTOMERS',
                                         type  => 'TABLE');
```

Now you can define the trigger:

```
CREATE OR REPLACE TRIGGER sales.t_customers
  BEFORE INSERT OR UPDATE ON sales.customers
  FOR EACH ROW
DECLARE
  timestamp$x DATE := SYSDATE;
  site$x VARCHAR2(128) := dbms_reputil.global_name;
BEGIN
 -- Don't fire if a snapshot refreshing;
 -- Don't fire if a master and replication is turned off
 IF (NOT (dbms_snapshot.i_am_a_refresh)
 AND dbms_reputil.replication_is_on) THEN
```

```
  IF NOT dbms_reputil.from_remote THEN
    IF INSERTING THEN
       -- set site and timestamp columns.
       :new."TIMESTAMP" := TIMESTAMP$X;
       :new."SITE" := SITE$X;
    ELSIF UPDATING THEN
        IF(:old."ADDR1" = :new."ADDR1" OR
          (:old."ADDR1" IS NULL AND :new."ADDR1" IS NULL)) AND
          (:old."ADDR2" = :new."ADDR2" OR
          (:old."ADDR2" IS NULL AND :new."ADDR2" IS NULL)) AND
          (:old."FIRST_NAME" = :new."FIRST_NAME" OR
          (:old."FIRST_NAME" IS NULL AND  :new."FIRST_NAME"
                                              IS NULL)) AND
          (:old."LAST_NAME" = :new."LAST_NAME" OR
          (:old."LAST_NAME" IS NULL AND :new."LAST_NAME" IS NULL)) AND
          (:old."SITE" = :new."SITE" OR
          (:old."SITE" IS NULL AND :new."SITE" IS NULL)) AND
          (:old."TIMESTAMP" = :new."TIMESTAMP" OR
          (:old."TIMESTAMP" IS NULL AND :new."TIMESTAMP" IS NULL)) THEN
          -- column group was not changed; do nothing
          NULL;
        ELSE
          -- column group was changed; set site and timestamp columns.
          :new."SITE" := SITE$X;
          :new."TIMESTAMP" := TIMESTAMP$X;
          -- consider time synchronization problems;
          -- previous update to this row may have originated from a site
          -- with a clock time ahead of the local clock time.
          IF :old."TIMESTAMP" IS NOT NULL AND
             :old."TIMESTAMP" > :new."TIMESTAMP" THEN
             :new."TIMESTAMP" := :old."TIMESTAMP" + 1 / 86400;
          ELSIF :old."TIMESTAMP" IS NOT NULL AND
             :old."TIMESTAMP" = :new."TIMESTAMP" AND
            (:old."SITE" IS NULL OR :old."SITE" != :new."SITE") THEN
            :new."TIMESTAMP" := :old."TIMESTAMP" + 1 / 86400;
          END IF;
        END IF;
      END IF;
    END IF;
  END IF;
END;
```

Next, use Replication Manager to add the trigger to the master group that contains the replicated table and resume replication activity for the master group.

# Configuring Uniqueness Conflict Resolution

In a typical advanced replication environment, you should try to avoid the possibility of uniqueness conflicts. However, if uniqueness conflicts must be addressed, you can assign one or more conflict resolution methods to a PRIMARY KEY or UNIQUE constraint in a replicated table to resolve uniqueness conflicts when they occur. The following sections explain how to configure a replicated table and associate uniqueness conflict resolution methods for PRIMARY KEY and UNIQUE constraints.

## Assigning a Uniqueness Conflict Resolution Method

To assign a uniqueness conflict resolution method to a constraint, use the ADD_ UNIQUE_RESOLUTION procedure of the DBMS_REPCAT package. For example, the following statement assigns the APPEND_SEQUENCE uniqueness conflict resolution method to the C_CUST_NAME constraint of the CUSTOMERS table:

```
DBMS_REPCAT.ADD_UNIQUE_RESOLUTION (
                  sname                => 'acct',
                  oname                => 'customers',
                  constraint_name      => 'c_cust_name',
                  sequence_no          =>  1,
                  method               => 'APPEND SEQUENCE',
                  comment              => 'Resolve Conflict',
                  parameter_column_name => 'last_name');
```

**Additional Information:** The parameters for the ADD_UNIQUE_RESOLUTION procedure are described in Chapter 10.

## Removing a Uniqueness Conflict Resolution Method

To remove a uniqueness conflict resolution method from a constraint, use the DROP_ UNIQUE_RESOLUTION procedure of the DBMS_REPCAT package. For example, the following statement drops the uniqueness conflict resolution method assigned in the previous example:

```
DBMS_REPCAT.DROP_UNIQUE_RESOLUTION (
     sname             => 'acct',
     oname             => 'customers',
     constraint_name   => 'c_cust_name',
     sequence_no       =>  1 );
```

**Additional Information:** The parameters for the DROP_UNIQUE_RESOLUTION procedure are described in Chapter 9.

## Prebuilt Uniqueness Resolution Methods

Oracle provides three prebuilt methods for resolving uniqueness conflicts:

- Append the global name of the originating site to the column value from the originating site.
- Append a generated sequence number to the column value from the originating site.
- Discard the row value from the originating site.

The following sections explain each uniqueness conflict resolution method in detail.

**Note:** Oracle's prebuilt uniqueness conflict resolution methods do not actually converge the data in a replicated environment; they simply provide techniques for resolving constraint violations. When you use one of Oracle's uniqueness conflict resolution methods, you should also use a notification mechanism to alert you to uniqueness conflicts when they happen and then manually converge replicated data, if necessary. For more information about data convergence, see "Guaranteeing Data Convergence" on page 5-37.

### Append Site Name/Sequence

The *append site name* and *append sequence* methods work by appending a string to a column that is generating a DUP_VAL_ON_INDEX exception. Although these methods allow the column to be inserted or updated without violating a unique integrity constraint, they do not provide any form of convergence between multiple master sites. The resulting discrepancies must be manually resolved; therefore, these methods are meant to be used with some form of a notification facility**.**

**Note:** Both append site name and append sequence can be used on character columns only.

These methods can be useful when the availability of the data may be more important than the complete accuracy of the data. To allow data to be available as soon as it is replicated

- Select append site name or append sequence.
- Use a notification scheme to alert the appropriate person to resolve the duplication, instead of logging a conflict.

When a uniqueness conflict occurs, the *append site name* method appends the global database name of the site originating the transaction to the replicated column value. The name is appended to the first period (.). For example, HOUSTON.WORLD becomes HOUSTON.

Similarly, the *append sequence* method appends a generated sequence number to the column value. The column value is truncated as needed. If the generated portion of the column value exceeds the column length, the conflict method does not resolve the error.

### Discard

The *discard uniqueness* conflict resolution method resolves uniqueness conflicts by simply discarding the row from the originating site that caused the error. This method does not guarantees convergence with multiple masters and should be used with a notification facility.

Unlike the append methods, the discard uniqueness method minimizes the propagation of data until data accuracy can be verified.

# Configuring Delete Conflict Resolution

In a typical advanced replication environment, you should try to avoid the possibility of delete conflicts. However, if the possibility of delete conflicts must be addressed, you can create your own delete conflict resolution methods and then assign them to a replicated table. The following sections explain how to configure a replicated table and associate user-defined delete conflict resolution methods to the table.

## Assigning a Delete Conflict Resolution Method

To assign a user-defined delete conflict resolution method to a replicated table, use the ADD_ DELETE_RESOLUTION procedure of the DBMS_REPCAT package. For example, the following statement assigns the CUSTOMERS_DELETE_M1 user-defined function as a delete conflict resolution method for the CUSTOMERS table:

```
DBMS_REPCAT.ADD_DELETE_RESOLUTION (
     sname                 => 'acct',
     oname                 => 'customers',
     sequence_no           => 1,
     parameter_column_name  => 'last_name',
     function_name         => 'customers_delete_m1' );
```

**Additional Information:** The parameters for the ADD_DELETE_RESOLUTION procedure are described in Chapter 10.

## Removing a Delete Conflict Resolution Method

To remove a delete conflict resolution method from a replicated table, use the DROP_ DELETE_RESOLUTION procedure of the DBMS_REPCAT package. For example, the following statement drops the delete conflict resolution method assigned in the previous example:

```
DBMS_REPCAT.DROP_DELETE_RESOLUTION (
     sname                 => 'acct',
     oname                 => 'customers',
     sequence_no           => 1 );
```

**Additional Information:** The parameters for the DROP_DELETE_RESOLUTION procedure are described in Chapter 10.

## Guaranteeing Data Convergence

Data convergence is a requirement in an advanced replication system. Data convergence happens when all replication sites ultimately have the same values for a given row. When you configure an advanced replication system, the conflict resolution strategy you design must guarantee data convergence. Table 5–1 summarizes Oracle's prebuilt update conflict resolution methods and in which types of configurations they guarantee data convergence between multiple master sites and their associated snapshot sites.

***Table 5–1 Data Convergence Properties of Update Conflict Resolution Methods***

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

**Note:** Oracle's prebuilt uniqueness conflict resolution methods do not ensure data convergence in any type of replicated environment. Therefore, you should configure conflict notification along with uniqueness conflict resolution and manually converge data, if necessary.

If you have more than one master site, the *overwrite*, *discard*, and *average* methods cannot guarantee data convergence; consequently, these methods should only be used in conjunction with a notification facility.

If you have more than two master sites, several other methods cannot guarantee convergence. Furthermore, network failures and infrequent pushing of the deferred remote procedure call (RPC) queue increase the likelihood of non-convergence for these methods.

## Avoiding Ordering Conflicts

Ordering conflicts can occur in advanced replication configurations with three or more master sites. If propagation to master site X is blocked for any reason, updates to replicated data can continue to be propagated among other master sites. When propagation resumes, these updates may be propagated to site X in a different order than they occurred on the other masters, and these updates may conflict. By default, the resulting conflicts will be recorded in the error log and can be re-executed after the transactions they depend upon are propagated and applied. Whenever possible, however, it is best to avoid or automatically resolve ordering conflicts. For example, you should select conflict resolution routines that ensure convergence in multimaster configurations where ordering conflicts are possible.

The example in Table 5–2 shows how having three master sites can lead to ordering conflicts. Master Site A has priority 30; Master Site B has priority 25; and Master Site C has priority 10; *x* is a column of a particular row in a column group that is assigned the *site-priority* conflict resolution method. The highest priority is given to the site with the highest priority value. Priority values can be any Oracle number and do not have to be consecutive integers.

***Table 5–2 Ordering Conflicts With Site Priority - More Than Two Masters***

| 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.<br>Site A and Site B agree that x = 5.<br><br>Site C is still unavailable.<br>The update transaction remains in the queue at Site A. | 5 | 5 | down |
| 5 | Site C becomes available with x = 2.<br>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.<br>Sites A and B agree that x = 7.<br>Site C still says x = 2. | 7 | 7 | 2 |
| 8 | Site B pushes the transaction to Site C.<br>Site C says the old value of x = 2;<br>Site B says the old value of x = 5.<br>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).<br>All site agree that x = 7. | 7 | 7 | 7 |
| 9 | Site A successfully pushes its transaction (x = 5) to Site C.<br>Oracle detects a conflict because the current value at Site C (x = 7) does not match the old value at Site A (x = 2).<br><br>Site A has a higher priority (30) than Site C (10).<br>Oracle resolves the conflict by applying the outdated update from Site A (x = 5).<br><br>Because of this ordering conflict, the sites no longer converge. | 7 | 7 | 5 |

You can guarantee convergence when using priority groups if you require that the flow of ownership be ordered. For example, the workflow model dictates that information flow one-way through a three-step sequence:

1. From the ORDERING site.
2. to the SHIPPING site.
3. to the BILLING site.

If the billing site receives a change to a row from the ordering site after the billing site received a change to that row from the shipping site, the billing site ignores the out-of-order change because the change from shipping has a higher priority.

**Suggestion:** To help determine which conflict resolution method to use, make a diagram or time-action table (such as Table 5–2) to help uncover any potential loopholes in your conflict resolution methodology.

## Minimizing Data Propagation for Update Conflict Resolution

To detect and resolve an update conflict for a row, the propagating site must send a certain amount of data about the new and old versions of the row to the receiving site. Depending on your environment, the amount of data that Oracle propagates to support update conflict detection and resolution can be different.

For example, when you create a replicated table and all participating sites are Oracle8 databases, you can choose minimize the amount of data that must be communicated to determine conflicts for each changed row in the table. In this case, Oracle propagates:

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

**Note:** For an inserted row, the row has no *old* value. For a deleted row, the row has no *new* value.

In general, you should choose to minimize data propagation in Oracle8-only advanced replication systems to reduce the amount of data that Oracle needs to transmit across the network. As a result, you can help to improve overall system performance.

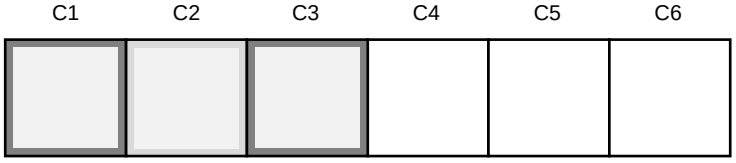
Alternatively, when a replicated environment uses both Oracle7 and Oracle8 sites, you cannot minimize the communication of row data for update conflict resolution. In this case, Oracle must propagate the entire old and new versions of each changed row to perform conflict resolution.

When you use Replication Manager to generate support for replicated tables, you can minimize data propagation by enabling the **Minimize Communications** setting of the **Edit Replication Object property** sheet. When using the replication API, you can minimize data propagation by setting the `min_communication` parameter to TRUE in the following DBMS_REPCAT procedures:

- CREATE_SNAPSHOT_REPOBJECT
- GENERATE_REPLICATION_SUPPORT
- GENERATE_REPLICATION_TRIGGER
- GENERATE_SNAPSHOT_SUPPORT

## Minimizing Communication Examples

In the replicated table below, columns 1 and 3 together compose the primary key. There are two column groups, columns 1 - 3 and columns 4 - 6.




column group 1 = C1, C2, & C3
column group 2 = C4, C5, & C6
primary key = C1 & C3

If you disable the **Minimize Communication** setting when generating replication support for the table, Oracle sends six old values (C1 -C6) and six new values (C1 - C6) for any update. For example, if you update column C4,

| | C1 | C2 | C3 | C4 | C5 | C6 |
|---|---|---|---|---|---|---|
| **old** | old value | old value | old value | old value | old value | old value |
| **new** | new value | new value | new value | **update new value** | new value | new value |

Alternatively, if you enable the **Minimize Communication** setting when generating replication support for the table, Oracle minimizes communication. For example, when you update **column C4,** Oracle sends:

- Old values for the primary key.
- Old values for the modified column group.
- NULLs for old values not in the primary key and the modified column group.
- NULL new values for all columns not updated.
- Actual new values for updated columns.

| | C1 | C2 | C3 | C4 | C5 | C6 |
|---|---|---|---|---|---|---|
| **old** | primary key | NULL | primary key | modified column group | modified column group | modified column group |
| **new** | NULL | NULL | NULL | **update new value** | NULL | NULL |

If you update columns **C2 and C4**, Oracle sends:

- Old values for both modified column groups.
- Old values for the primary key columns.

- New values for the two updates.
- NULLs for the other four new values.

| | C1 | C2 | C3 | C4 | C5 | C6 |
|---|---|---|---|---|---|---|
| **old** | modified column group1 and primary key | modified column group1 | modified column group1 and primary key | modified column group2 | modified column group2 | modified column group2 |
| **new** | NULL | **update** | NULL | **update** | NULL | NULL |

If you update **column 2,** Oracle sends:

- Old values for the primary key.
- Old values for members of the modified column group.
- The new values for the updated column.
- NULLs for the other five new values.

| | | | | | | |
|---|---|---|---|---|---|---|
| **old** | modified column group and primary key | modified column group | modified column group and primary key | NULL | NULL | NULL |
| **new** | NULL | **update** | NULL | NULL | NULL | NULL |

## Further Reducing Data Propagation

If you have minimized propagation using the method described above, you can further reduce data propagation in some cases by using the DBMS_REPCAT. SEND_AND_COMPARE_OLD_VALUES procedure to send old values only if they

are needed to detect and resolve conflicts. For example, the latest timestamp conflict detection and resolution method does not require old values for non-key and non-timestamp columns.

**Suggestion:** Further minimizing propagation of old values is particularly valuable if you are replicating LOB datatypes and do not expect conflicts on these columns.

**Attention:** You must ensure that the appropriate old values are propagated to detect and resolve anticipated conflicts. User-supplied conflict resolution procedures must deal properly with NULL old column values that are transmitted. Using SEND_AND_COMPARE_OLD_VALUES to further reduce data propagation reduces protection against unexpected conflicts.

To further reduce data propagation execute the following procedure:

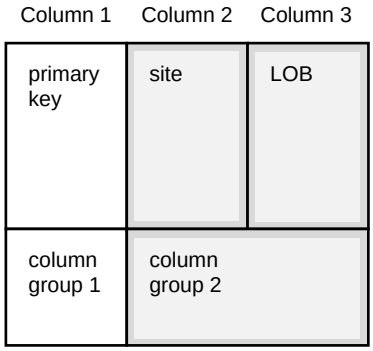
```
DBMS_REPCAT.VARCHAR2SDBMS_REPCAT.SEND_AND_COMPARE_OLD_VALUES(
    sname          IN    VARCHAR2,
    oname          IN    VARCHAR2,
    column_list    IN    VARCHAR2 |
    column_table   IN    DBMS_REPCAT.VARCHAR2s
    operation      IN    VARCHAR2 := 'UPDATE',
     send            IN     BOOLEAN  :=  TRUE);
```

After executing this procedure, you must generate replication support again with `min_communication` set to TRUE for this change to take effect.

**Note:** The operation parameter allows you to decide whether or not to transmit old values for non-key columns when rows are deleted and/or when non-key columns are updated. If you do not send the old value, Oracle sends a NULL in place of the old value and assumes the old value is equal to the current value of the column at the target side when the update or delete is applied.

The specified behavior for old column values is exposed in two columns in the REPCOLUMN view: COMPARE_OLD_ON_DELETE ('Y' or 'N') and COMPARE_OLD_ON_UPDATE ('Y' or 'N').

The following example shows how you can further reduce data propagation by using SEND_AND_COMPARE_OLD_VALUES. Consider a table called 'SCOTT.REPORTS' with 3 columns. Column 1 is the primary key and is in its own column group (column group 1). Column 2 and column 3 are in a second column group (column group 2).




The conflict resolution strategy for the second column group is site priority. Column 2 is a VARCHAR2 column containing the site name. Column 3 is a LOB column. Whenever you update the LOB, you must also update column 2 with the global name of the site at which the update occurs. Because there are no triggers for piecewise updates to LOBs, you must explicitly update column 2 whenever you do a piecewise update on the LOB.

Suppose you generate replication support for SCOTT.REPORTS with `min_communication` set to TRUE and then use an UPDATE statement to modify column 2 (the site name) and column 3 (the LOB). The deferred remote procedure call (RPC) contains the new value of the site name and the new value of the LOB because they were updated. The deferred RPC also contains the old value of the primary key (column 1), the old value of the site name (Column 2), and the old value of the LOB (Column 3).

**Note:** The conflict detection and resolution strategy does not require the old value of the LOB. Only column C2 (the site name) is required for both conflict detection and resolution. Sending the old value for the LOB could add significantly to propagation time.

To ensure that the old value of the LOB is not propagated when either column C2 or column C3 is updated, make the following call:

```
dbms_repcat.send_and_compare_old_values(
            sname          => 'SCOTT',
            oname          => 'REPORTS'
            column_list    => 'C3',
            operation      => 'UPDATE',
            send           => FALSE);
```

You must generate replication support for SCOTT.REPORTS again with `min_communication` set to TRUE for this change to take effect. Suppose you subsequently use an UPDATE statement to modify column 2 (the site name) and column 3 (the LOB). The deferred RPC contains the old value of the primary key (column 1), the old and new values of the site name (column 2), and just the new value of the LOB (column 3). The deferred RPC contains NULLs for the new value of the primary key and the old value of the LOB.

**Additional Information:** The parameters for the SEND_AND_COMPARE_OLD_VALUES procedure are described in Table 9–182 on page 9 - 164 and the exceptions are described in Table 9–183 on page 9 - 164.

# User-Defined Conflict Resolution Methods

Oracle allows you to write your own conflict resolution or notification methods. A user-defined conflict resolution method is a PL/SQL function that returns either TRUE or FALSE. TRUE indicates that the method has successfully resolved all conflicting modifications for a column group. If the method cannot successfully resolve a conflict, it should return FALSE. Oracle continues to evaluate available conflict resolution methods, in sequence order, until either a method returns TRUE or there are no more methods available.

If the conflict resolution method raises an exception, Oracle stops evaluation of the method, and, if any other methods were provided to resolve the conflict (with a later sequence number), Oracle does not evaluate them.

## Conflict Resolution Method Parameters

The parameters needed by a user-defined conflict resolution method are determined by the type of conflict being resolved (unique, update, or delete) and the columns of the table being replicated. All conflict resolution methods take some combination of old, new, and current column values for the table.

- The old value represents the value of the row at the initiating site before you made the change.
- The new value represents the value of the row at the initiating site after you made the change.
- The current value represents the value of the equivalent row at the receiving site.

**Note:** Recall that Oracle uses the primary key (or the key specified by SET_COLUMNS) to determine which rows to compare.

The conflict resolution function should accept as parameters the values for the columns specified in the PARAMETER_COLUMN_NAME argument to the DBMS_REPCAT.ADD_*conflicttype*_RESOLUTION procedures. The column parameters are passed to the conflict resolution method in the order listed in the PARAMETER_COLUMN_NAME argument, or in ascending alphabetical order if you specified '*' for this argument. When both old and new column values are passed as parameters (for update conflicts), the old value of the column immediately precedes the new value.

**Attention:** Type checking of parameter columns in user-defined conflict resolution methods is not performed until you regenerate replication support for the associated replicated table.

## Resolving Update Conflicts

For update conflicts, a user-defined function should accept the following values for each column in the column group:

- Old column value from the initiating site. The mode for this parameter is IN. This value should not be changed.
- New column value from the initiating site. The mode for this parameter is IN OUT. If the function can resolve the conflict successfully, it should modify the new column value as needed.
- Current column value from the receiving site. The mode for this parameter is IN.

The old, new, and current values for a column are received consecutively. The final argument to the conflict resolution method should be a Boolean flag. If this flag is FALSE, it indicates that you have updated the value of the IN OUT parameter, new, and that you should update the current column value with this new value. If this flag is TRUE, it indicates that the current column value should not be changed.

## Resolving Uniqueness Conflicts

Uniqueness conflicts can occur as the result of an INSERT or UPDATE. Your uniqueness conflict resolution method should accept the new column value from the initiating site in IN OUT mode for each column in the column group. The final parameter to the conflict resolution method should be a Boolean flag.

If the method can resolve the conflict, it should modify the new column values so that Oracle can insert or update the current row with the new column values. Your function should set the Boolean flag to TRUE if it wants to discard the new column values, and FALSE otherwise.

Because a conflict resolution method cannot guarantee convergence for uniqueness conflicts, a user-defined uniqueness resolution method should include a notification mechanism.

## Resolving Delete Conflicts

Delete conflicts occur when you successfully delete from the local site, but the associated row cannot be found at the remote site (for example, because it had been updated). For delete conflicts, the function should accept old column values in IN OUT mode for the entire row. The final parameter to the conflict resolution method should be a BOOLEAN flag.

If the conflict resolution method can resolve the conflict, it modifies the old column values so that Oracle can delete the current row that matches all old column values. Your function should set the Boolean flag to TRUE if it wants to discard these column values, and FALSE otherwise.

If you perform a delete at the local site and an update at the remote site, the remote site detects the delete conflict, but the local site detects an unresolvable update conflict. This type of conflict cannot be handled automatically. The conflict will raise a NO_DATA_FOUND exception and Oracle logs the transaction as an error transaction.

Designing a mechanism to properly handle these types of update / delete conflicts is difficult. It is far easier to avoid these types of conflicts entirely, by simply "marking" deleted rows, and then purging them using procedural replication.

**Additional Information:** See "Avoiding Delete Conflicts" on page 7-20.

## Restrictions

You should avoid the following types of SQL commands in user-defined conflict resolution methods. Use of such commands can result in unpredictable results.

- Data Definition Language statements (for example, CREATE, ALTER, DROP).
- Transaction control statements (for example, COMMIT, ROLLBACK).
- Session control (for example, ALTER SESSION).
- System control (for example, ALTER SYSTEM).

## Example User-Defined Conflict Resolution Method

The following examples show user-defined methods that are variations on the standard MAXIMUM and ADDITIVE prebuilt conflict resolution methods. Unlike the standard methods, these custom functions can handle nulls in the columns used to resolve the conflict.

### Maximum User Function

```
-- User function similar to MAXIMUM method.
-- If curr is null or curr < new, use new values.
-- If new is null or new < curr, use current values.
-- If both are null, no resolution.
-- Does not converge with > 2 masters, unless
-- always increasing.

FUNCTION max_null_loses(old                 IN    NUMBER,
                        new                 IN  OUT  NUMBER,
                        cur                 IN    NUMBER,
                        ignore_discard_flag OUT    BOOLEAN)
  RETURN BOOLEAN IS
BEGIN
   IF (new IS NULL AND cur IS NULL) OR new = cur THEN
       RETURN FALSE;
   END IF;
   IF new IS NULL THEN
       ignore_discard_flag := TRUE;
   ELSIF cur IS NULL THEN
       ignore_discard_flag := FALSE;
   ELSIF new < cur THEN
       ignore_discard_flag := TRUE;
   ELSE
       ignore_discard_flag := FALSE;
   END IF;
   RETURN TRUE;
END max_null_loses;
```

### Additive User Function

```
-- User function similar to ADDITIVE method.
-- If old is null, old = 0.
-- If new is null, new = 0.
-- If curr is null, curr = 0.
-- new = curr + (new - old) -> just like ADDITIVE method.

FUNCTION additive_nulls(old                 IN    NUMBER,
```

```
                        new                 IN  OUT  NUMBER,
                        cur                 IN    NUMBER,
                        ignore_discard_flag OUT    BOOLEAN)
   RETURN BOOLEAN IS
   old_val NUMBER := 0.0;
   new_val NUMBER := 0.0;
   cur_val NUMBER := 0.0;
BEGIN
   IF old IS NOT NULL THEN
      old_val := old;
   END IF;
   IF new IS NOT NULL THEN
      new_val := new;
   END IF;
   IF cur IS NOT NULL THEN
      cur_val := cur;
   END IF;
   new := cur_val + (new_val - old_val);
   ignore_discard_flag := FALSE;
   RETURN TRUE;
END additive_nulls;
```

## User-Defined Conflict Notification Methods

A conflict notification method is a user-defined function that provides conflict notification rather than or in addition to conflict resolution. For example, you can write your own conflict notification methods to log conflict information in a database table, send an email message, or page an administrator. After you write a conflict notification method, you can assign it to a column group (or constraint) in a specific order so that Oracle notifies you when a conflict happens, before attempting subsequent conflict resolution methods, or after Oracle attempts to resolve a conflict but cannot do so.

To configure a replicated table with a user-defined conflict notification mechanism, you must complete the following steps:

1. Create a conflict notification log.
2. Create the user-defined conflict notification method in a package.

The following sections explain each step.

## Creating a Conflict Notification Log

When configuring a replicated table to use a user-defined conflict notification method, the first step is to create a database table that can record conflict notifications. You can create a table to log conflict notifications for one or many tables in a master group.

To create a conflict notification log table at all master sites, use the replication execute DDL facility. For more information, "Executing DDL Within a Master Group" on page 6-2. Do *not* generate replication support for the conflict notification tables because their entries are specific to the site that detects a conflict.

### Sample Conflict Notification Log Table

The following CREATE TABLE statement creates a table that you can use to log conflict notifications from several tables in a master group.

```
CREATE TABLE conf_report (
 line          NUMBER(2),    --- used to order message text
 txt           VARCHAR2(80), --- conflict notification message
 timestamp     DATE,         --- time of conflict
 table_name    VARCHAR2(30), --- table in whic the
                             --- conflict occurred
 table_owner   VARCHAR2(30), --- owner of the table
 conflict_type VARCHAR2(6)   --- INSERT, DELETE or UNIQUE
)
```

## Creating a Conflict Notification Package

To create a conflict notification method, you must define the method in a PL/SQL package and then replicate the package as part of a master group along with the associated replicated table.

A conflict notification method can perform conflict notification only, or both conflict notification and resolution. If possible, you should always use one of Oracle's prebuilt conflict resolution methods to resolve conflicts. When a user-defined conflict notification method performs only conflict notification, assign the user-defined method to a column group (or constraint) along with conflict resolution methods that can resolve conflicts.

**Note:** If Oracle cannot ultimately resolve a replication conflict, Oracle rolls back the entire transaction, including any updates to a notification table. If notification is necessary independent of transactions, you can design a notification mechanism to use the Oracle DBMS_PIPES package or the database interface to Oracle Office.

### Sample Conflict Notification Package

The following package and package body perform a simple form of *conflict notification* by logging uniqueness conflicts for a CUSTOMERS table into the previously defined CONF_REPORT table.

**Note:** This example of *conflict notification* does not resolve any conflicts. You should either provide a method to resolve conflicts (for example, *discard* or *overwrite*), or provide a notification mechanism that will succeed (for example, using e-mail) even if the error is not resolved and the transaction is rolled back. With simple modifications, the following user-defined conflict notification method can take more active steps. For example, instead of just recording the notification message, the package can use the DBMS_OFFICE utility package to send an Oracle Office email message to an administrator.

```
CREATE OR REPLACE PACKAGE notify AS
 -- Report uniqueness constraint violations on CUSTOMERS table
 FUNCTION customers_unique_violation (
   first_name         IN OUT VARCHAR2,
   last_name          IN OUT VARCHAR2,
   discard_new_values IN OUT BOOLEAN)
 RETURN BOOLEAN;
END notify;
/

CREATE OR REPLACE PACKAGE BODY notify AS
 -- Define a PL/SQL table to hold the notification message
 TYPE message_table IS TABLE OF VARCHAR2(80) INDEX BY BINARY_INTEGER;

 PROCEDURE report_conflict (
   conflict_report IN MESSAGE_TABLE,
   report_length   IN NUMBER,
   conflict_time   IN DATE,
   conflict_table  IN VARCHAR2,
   table_owner     IN VARCHAR2,
   conflict_type   IN VARCHAR2) IS
 BEGIN
   FOR idx IN 1..report_length LOOP
     BEGIN
       INSERT INTO sales.conf_report
         (line, txt, timestamp, table_name, table_owner, conflict_type)
         VALUES (idx, SUBSTR(conflict_report(idx),1,80), conflict_time,
                  conflict_table, table_owner, conflict_type);
     EXCEPTION WHEN others THEN NULL;
     END;
   END LOOP;
```

```
 END report_conflict;

 -- This is the conflict resolution method that will be called first when
 -- a uniqueness constraint violated is detected in the CUSTOMERS table.
 FUNCTION customers_unique_violation (
   first_name  IN OUT VARCHAR2,
   last_nameIN OUT VARCHAR2,
   discard_new_valuesIN OUT BOOLEAN)
  RETURN BOOLEAN IS
   local_node  VARCHAR2(128);
   conf_report MESSAGE_TABLE;
   conf_time   DATE := SYSDATE;
  BEGIN
  -- Get the global name of the local site
    BEGIN
      SELECT global_name INTO local_node FROM global_name;
    EXCEPTION WHEN others THEN local_node := '?';
    END;
  -- Generate a message for the DBA
  conf_report(1) := 'UNIQUENESS CONFLICT DETECTED IN TABLE CUSTOMERS ON ' ||
                    TO_CHAR(conf_time, 'MM-DD-YYYY HH24:MI:SS');
  conf_report(2) := '  AT NODE ' || local_node;
  conf_report(3) := 'ATTEMPTING TO RESOLVE CONFLICT USING ' ||
                    'APPEND SEQUENCE METHOD';
  conf_report(4) := 'FIRST NAME: ' || first_name;
  conf_report(5) := 'LAST NAME:  ' || last_name;
  conf_report(6) := NULL;
  --- Report the conflict
  report_conflict(conf_report, 5, conf_time, 'CUSTOMERS',
                'OFF_SHORE_ACCOUNTS', 'UNIQUE');
  --- Do not discard the new column values. They are still needed by
  --- other conflict resolution Methods
  discard_new_values := FALSE;
  --- Indicate that the conflict was not resolved.
    RETURN FALSE;
  END customers_unique_violation;
END notify;
/
```

# Viewing Conflict Resolution Information

Oracle provides replication catalog (REPCAT) views that you can use to determine what conflict resolution methods are being used by each of the tables and column groups in your replicated environment. Each view has three versions: USER_*, ALL_*, SYS.DBA_*. The views available include the following:

| | |
|---|---|
| REPRESOLUTION_ METHOD | Lists all of the available conflict resolution methods. |
| REPCOLUMN_GROUP | Lists all of the column groups defined for the database. |
| REPGROUPED_COLUMN | Lists all of the columns in each column group in the database. |
| REPPRIORITY_GROUP | Lists all of the priority groups and site priority groups defined for the database. |
| REPPRIORITY | Lists the values and corresponding priority levels for each priority or site priority group. |
| REPCONFLICT | Lists the types of conflicts (delete, update, or uniqueness) for which you have specified a resolution method, for the tables, column groups, and unique constraints in the database. |
| REPRESOLUTION | Shows more specific information about the conflict resolution method used to resolve conflicts on each object. |
| REPPARAMETER_ COLUMN | Shows which columns are used by the conflict resolution methods to resolve a conflict. |

**Additional Information:** Chapter 10, “Data Dictionary Views”.

# 6

# Administering a Replicated Environment

This chapter describes how to administer your replicated database environment. The topics include the following:

- Advanced Management of Master and Snapshot Groups.
- Monitoring an Advanced Replication System.
- Database Backup and Recovery in Replication Systems.
- Auditing Successful Conflict Resolution.
- Determining Differences Between Replicated Tables.
- Updating The Comments Fields in Views.

# Advanced Management of Master and Snapshot Groups

Chapters 3 and 4 in this book discuss the most commonly performed procedures that you use to manage master and snapshot groups in an advanced replication environment. The following sections explain several less commonly used administrative procedures that involve the management of master and snapshot groups.

## Executing DDL Within a Master Group

Replication Manager lets you propagate one or more SQL DDL statements to some or all of the master sites in a master group. This option lets you execute unique DDL that is not specifically supported within Oracle's replication management API. For example, you might want to create rollback segments and users that are necessary to support a replication environment.

**Warning:** Do not execute DDL that could damage global database integrity in a multimaster environment. For example, do not execute DDL statements to alter a replication object at any site.

To execute DDL at selected master sites in a master group:

1. Click the target master group at its master definition site.
2. Click the **Properties** toolbar button.
3. Click the **Operations** page of the **Edit Master Group** property sheet.
4. Click **Execute DDL**.
5. Use the **Execute DDL** dialog to execute DDL statements at selected destinations.

**API Equivalent:** DBMS_REPCAT.EXECUTE_DDL

## Validating a Master Group

To validate the integrity of a master group across all master sites in a multimaster replication environment, you can validate the master group. To validate a master group:

1. Click the target master group at its master definition site.
2. Click the **Properties** toolbar button.
3. Click the **Validation** page of the **Edit Master Group** property sheet.

The **Validation** page lets you target specific items to validate, including:

- The replication support and validity of objects in the master group.
- The existence and scheduling of the master group's scheduled links.

When you are ready to validate the master group, Click **Validate**.

## Relocating a Master Group's Definition Site

If the master definition site of a master group becomes unavailable or you simply want to relocate the master destination site for the group:

1. Click the target master group at any master site.
2. Click the **Properties** toolbar button.
3. Click the **Destinations** page of the **Edit Master Group** property sheet.
4. Click **Change** to change the groups master definition site.
5. Use the **Change Masterdef** dialog to change the group's master definition site.

When you relocate the master definition site for a master group, you can choose to notify:

- All other master sites.
- The current master definition site—uncheck this setting when the current master definition site is unavailable.

**API Equivalent:** DBMS_REPCAT.RELOCATE_MASTERDEF

## Changing a Snapshot Group's Master Site

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

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

In this example, the master site for the ACCT object group is changed to the DBS2 master site.

You must call this procedure at the snapshot site whose master site you want to change. The new database must be a master site in the replicated environment.

When you call this procedure, Oracle uses the new master to perform a full refresh of each snapshot in the local snapshot group.

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

**Additional Information**: The parameters for the SWITCH_SNAPSHOT_ MASTER procedure are described in Table 9–188 on page 9-168, and the exceptions are listed in Table 9–189 on page 9-168.

# Monitoring an Advanced Replication System

Oracle uses its internal system of deferred transactions and job queues to propagate changes among the sites in an advanced replication system. It is important that you monitor regularly the internal workings of a replication environment to ensure that it is running smoothly. The following sections explain how you can use Oracle Replication Manager to view and manage administration requests, deferred transactions, error transactions, and job queues.

## Managing Administration Requests

An administration request is a specific call to a procedure or function in Oracle's replication management API. For example, when you use Replication Manager to create a new master group, Replication Manager completes the task by making a call to the DBMS_REPCAT.CREATE_MASTER_REPGROUP procedure. All DDL changes to replication groups and the objects within generate administration requests. Many top-level administration requests generate additional replication management API calls to complete the request.

Administration requests are inserted by the master definition site in the administration request queues at all master sites as one distributed transaction. Each master site has a scheduled job, DO_DEFERRED_REPCAT_ ADMIN, that executes requests simultaneously at each site. As administration requests are processed, each site reports back to the master definition site. Oracle removes requests that complete successfully from the administration request queue at the master definition site. However, if any errors are encountered, the administration request remains in the master definition site's administration request queue with an error status.

Replication Manager allows you view and update the status of administration requests. The following sections explain more about how to manage administration requests for a master group in a multimaster advanced replication environment.

### Displaying Administration Requests

To display the administration requests for a master group at a master site:

1. Open the site's **Configuration** folder.
2. Open the subordinate **Master Groups** folder.
3. Open the target master group.
4. Click the **Admin Requests** folder of the target master group.

The detail panel of Replication Manager lists information for all pending master group administration requests at the corresponding master site, including:

- The corresponding replication management API call for the request.
- The current status of the request (for example, ready, error, awaiting callback, and so on).
- The source database for the administration request.
- Any errors associated with the administration request.

Replication Manager does not automatically update the display of the detail panel. To obtain a more current list of pending administration requests, refresh the display.

**Additional Information:** You can also view administration requests by querying the REPCATLOG data dictionary view.

### Applying Administration Requests

As long as each instance in an Oracle advanced replication facility has one or more SNP processes, each server 1fgruns a job at a regular interval to execute all administration requests. If you do not want to wait for Oracle to execute administration requests, use Replication Manager to apply all pending administration requests manually.

1. Display the administration requests for a master group at the target master site.
2. Right-click anywhere in the detail window.
3. Click **Apply all admin requests now**.

**Note:** Oracle automatically attempts to apply all administration requests synchronously among all master sites.

**API Equivalent:** DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN

### Deleting Administration Requests

Sometimes it is necessary to remove pending administration requests for a master group. For example, certain administration requests might return errors. Even after you resolve the corresponding error situation, administration requests remain in the server's queue unless you manually purge them.

To remove administration requests for a master group:

1. Display the administration requests for a master group at the target master site.
2. Click one or more target requests.
3. Click the **Delete** toolbar button.

**API Equivalent:** DBMS_REPCAT.PURGE_MASTER_LOG

### More about Administration Request Mechanisms

When you use Replication Manager or make a call to a procedure in the DBMS_REPCAT pac1fgage to administer an advanced replication system, Oracle uses its internal mechanisms to broadcast the request using synchronous replication. If a synchronous broadcast fails for any reason, Oracle returns an error message and rolls back the encompassing transaction.

When an Oracle Server receives an administration request, it records the request in the REPCATLOG view and the corresponding DDL statement in a child table of the REPCATLOG view. When you view administration requests for a master group at a master site, you might observe requests that are awaiting a callback from another master site. Whenever you use Replication Manager to create an administration request for a replication group, Oracle automatically inserts a job into the local job queue, if one does not already exist for the group. This job periodically executes the procedure DO_DEFERRED_REPCAT_ADMIN. Whenever you synchronously broadcast a request, Oracle attempts to start this job immediately in order to apply the replicated changes at each master site.

Assuming that Oracle does not encounter any errors, DO_DEFERRED_REPCAT_ ADMIN will be run whenever a background process is available to execute the job. The initialization parameter JOB_QUEUE_INTERVAL determines how often the background process wakes up. You can experience a delay if you do not have enough background processes available to execute the outstanding jobs.

**Note:** When JOB_QUEUE_PROCESSES = 0 at a site, you must apply administration requests manually for all groups at the site. See "Applying Administration Requests" on page 6-5 for more information.

For each call of DO_DEFERRED_REPCAT_ADMIN at a master site, the site checks the REPCATLOG view to see if there are any requests that need to be performed. When one or more administration requests are present, Oracle applies the request and updates any local views as appropriate. This event can occur asynchronously at each master site.

DO_DEFERRED_REPCAT_ADMIN executes the local administration requests in the proper order. When DO_DEFERRED_REPCAT_ADMIN is executed at a master that is not the master definition site, it does as much as possible. Some asynchronous activities such as populating a replicated table require communication with the master definition site. If this communication is not possible, DO_DEFERRED_REPCAT_ADMIN stops executing administration requests to avoid executing requests out of order. Some communication with the master definition site, such as the final step of updating or deleting an administration request at the master definition site, can be deferred and will not prevent DO_DEFERRED_REPCAT_ADMIN from executing additional requests.

The success or failure of an administration request at each master site is noted in the REPCATLOG view at each site. For each master group, Replication Manager displays the corresponding status of each administration request. Ultimately, each master site propagates the status of its administration requests to the master definition site. If a request completes successfully at a master site, Oracle removes the callback for the site from the REPCATLOG view at the master definition site. If the event completes successfully at all sites, all entries in the REPCATLOG view at all sites, including the master definition site, will be removed.

By synchronously broadcasting the change, Oracle ensures that all sites are aware of the change, and thus are capable of remaining in synch. By allowing the change to be applied at the site at a future point in time, Oracle provides you with the flexibility to choose the most appropriate time to apply changes at a site.

If an object requires automatically generated replication support, you must regenerate replication support after altering the object. Oracle then activates the internal triggers and regenerates the packages to support replication of the altered object at all master sites.

**Note:** Although the DDL must be successfully applied at the master definition site in order for these procedures to complete without error, this does not guarantee that the DDL is successfully applied at each master site. Replication Manager displays the status of all administration requests. Additionally, the REPCATLOG view contains interim status and any asynchronous error messages generated by the request.

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

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

### Diagnosing Problems with Administration Requests

After you administer an advanced replication environment, you should take a look at the administration requests to make sure that everything is running smoothly. While monitoring your system, you might find that there are problems related to administration requests. The following list provides you with some troubleshooting ideas for administration request problems.

- If administration requests are backed up at a site and not because of an error condition, ensure that there is not a problem with the local job running DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN. See "Diagnosing Problems with Jobs" on page 6-14 for more information about diagnosing job queue problems.

  **Note:** Additionally, check the LOG_USER column in the DBA_JOBS view to ensure that the replication job is being run on behalf of the replication administrator. Check the USERID column of the DBA_REPCATLOG view to ensure that the replication administrator was the user that submitted the request. DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN only performs those administrative requests submitted by the user that calls this procedure.

- Ensure the relevant databases are running and communication is possible.
- Ensure that the necessary private database links are available for the advanced replication facility, and that the user designated in the CONNECT TO clause (generally the replication administrator) has the necessary privileges. If you used the Replication Manager setup wizard to setup your sites, you should not have any problems.

## Managing Deferred Transactions

An advanced replication system using asynchronous data propagation uses an internal system of deferred transactions and job queues to push changes from one site to another. Replication Manager lets you view and manage deferred and error transactions queued at each server in an advanced replication system.

### Displaying Deferred Transactions

To display a summary list of a replication site's deferred transactions by their outgoing destination:

1. Open the **Administration** folder of the site.
2. Click on the **Deferred Transactions by Dest** folder.

The detail panel of Replication Manager lists summary information for all pending deferred transactions by destination.

To display individual deferred transactions for a particular destination:

1. Click on the destination in the **Deferred Transactions by Dest** folder.
2. Expand the destination in the tree panel.

To display the properties for an individual deferred transaction:

1. Click on the individual deferred transaction.
2. Click the **Properties** toolbar button.

To display the deferred calls for an individual deferred transaction:

1. Expand the individual deferred transaction.
2. Review the transaction's deferred calls and call destinations, and the transaction's destinations.

**Note:** Replication Manager does not automatically update the display of the detail panel. To obtain a more current list of pending deferred transactions, refresh the display.

**Additional Information:** You can also view deferred transactions by querying the DEFCALL, DEFCALLDEST, DEFLOB, DEFSCHEDULE, DEFTRAN, and DEFTRANDEST views.

### Executing Deferred Transactions

Each instance in an Oracle advanced replication facility that has one or more SNP processes and the necessary scheduled links automatically runs a job at a regular interval to execute deferred transactions automatically at targeted destinations. If you do not want to wait for Oracle to execute a deferred transaction, use Replication Manager to manually push all pending deferred transactions for a particular destination.

1. Click the destination in the **Deferred Transactions by Dest** folder for a site.

2. Click the **Properties** toolbar button.
3. Click **Reexecute all transactions now** of the **View Database Destination** property sheet to force the execution of all corresponding deferred transactions.

**API Equivalent:** DBMS_DEFER_SYS.PUSH

### Deleting Deferred Transactions

In some cases, you might know that a deferred transaction in the queue at your local site will cause an error transaction at the receiving site if pushed to the site. In such cases, it might be appropriate to delete the deferred transaction from the local queue to prevent an error transaction from happening.

**Warning:** See "Managing Error Transactions" on page 6-10 for more information about error transactions and when to delete deferred transactions.

To delete an individual deferred transaction:

1. Right-click on the individual deferred transaction.
2. Click **Delete Deferred Transaction** to delete the target transaction locally or click **Delete to all destinations** to delete the target transaction from all target replication sites.

**API Equivalent:** DBMS_DEFER_SYS.DELETE_TRAN

## Managing Error Transactions

When Oracle pushes a deferred transaction from a snapshot or master site to another master site, Oracle ensures that the transaction is not removed from the local queue until it has been successfully propagated to the remote site. However, a transaction can be successfully propagated to a master site without being successfully applied at the site. An error in applying a deferred transaction may be the result of a database problem, such as a lack of available space in a table that you are attempting to update, or may be the result of an unresolvable replication conflict. If an error occurs, Oracle performs the following actions at the receiving master site:

- Rolls back the transaction.
- Logs the error transaction in the receiving site's replication catalog.

### Displaying Error Transactions

You should frequently check each site in a replicated environment for error transactions and then resolve them. To display a summary list of a replication site's local error transactions:

1. Open the **Administration** folder of the site.
2. Click on the **Local Errors** folder.

The detail panel of Replication Manager lists summary information for all error transactions at the site.

To resolve an error transaction properly, you need specific information about what the transaction is attempting to perform. Deferred transactions consist of a series of deferred remote procedure calls that must be applied in a given order to maintain transaction consistency.

To display more information about an individual error transaction:

1. Click on the individual error transaction.
2. Click the **Properties** toolbar button.

To display the deferred calls for an individual error transaction:

1. Expand the individual error transaction.
2. Review the transaction's deferred calls and call destinations, and the transaction's destinations.

### Resolving Error Transactions

Once you have determined the cause of an error transaction, you may need to perform one or more of the following actions at the destination site after fixing the error:

- Re-execute the error transaction.
- Delete the error transaction from the local site.

### Executing Error Transactions

When you have resolved the problem that caused an error transaction, you can re-execute the error transaction. If the error transaction executes successfully, Oracle automatically removes the transaction from the local site's replication catalog.

To re-execute an error transaction:

1. Right-click on the individual error transaction.

2. Click **Re-execute Transaction**.

**API Equivalent:** DBMS_DEFER_SYS.EXECUTE_ERROR.

When Oracle re-executes an error transaction at the receiving site, Oracle executes the transaction in the *security context of the original receiver*. If the original receiver is no longer a valid user (that is, if the user was dropped), the deferred transaction can be re-executed *under the security context of a different user* using the DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER procedure. See this procedure, which is described in Table 9–25, "Return Values for DISABLED".

**Attention:** If you re-execute a single error transaction, Oracle does not commit the transaction, even if it executes successfully. If you are satisfied with the results of the transaction, you should issue the SQL command COMMIT WORK. If EXECUTE_ERROR re-executes multiple transactions, each transaction is committed as it completes.

### Deleting Error Transactions

Sometimes, you must manually resolve an error transaction (in other words, not re-execute the transaction to resolve it). When you have resolved the problem that caused an error transaction, you should delete the error transaction. To delete an error transaction:

1. Click on the individual error transaction.
2. Click the **Delete** toolbar button.

**API Equivalent:** DBMS_DEFER_SYS.DELETE_ERROR.

## Managing Local Jobs

Oracle runs jobs in each site's job queue to complete certain tasks automatically that are necessary to manage an advanced replication environment. For example, when you use Replication Manager to create and enable a scheduled link to a remote site, the local server places a job in its job queue that Oracle periodically runs to push local changes to a remote master.

Replication Manager has several features that you can use to view and manage the job queues of each server in an advanced replication system. The following sections explain more about viewing and managing a server's local jobs.

### Displaying a Site's Local Jobs

Oracle creates a job in a site's local job queue on behalf of the following operations:

- When you create a scheduled link, Oracle creates a job to push deferred transactions to the target remote master site. The PL/SQL for this type of job will include the API call DBMS_DEFER_SYS.PUSH.
- When you schedule purging of a site's deferred transaction queue, Oracle creates a job to perform this operation. The PL/SQL for this type of job will include the API call DBMS_DEFER_SYS.PURGE.
- When you create a master group, Oracle creates a job to push corresponding administration requests to the other master sites that manage the group (one job per destination). The PL/SQL for this type of job will include the API call DBMS_REPCAT.DO_DEFERRED_ REPCAT_ADMIN.
- When you create a refresh group, Oracle creates a job to perform regular refreshes for the group. The PL/SQL for this type of job will include the API call DBMS_REFRESH.REFRESH.

To display a summary list of a replication site's local job queue:

1. Open the **Administration** folder of the site.
2. Click on the **Local Jobs** folder.

The detail panel of Replication Manager lists summary information for all local jobs.

To display the properties for an individual local job:

1. Click on the individual job.
2. Click the **Properties** toolbar button.

The pages of the **Edit Job** property sheet let you view the various properties of the job, including:

- The job's next execution date and execution interval.
- The job's PL/SQL text.
- The most recent execution date for the job.
- The current status of the job (normal or broken).
- The number of failed executions for the job.
- The user environment under which the job executes.

### Editing the Properties of a Local Job

In certain situations, you might want to edit the properties of a local job. For example, the default execution interval for a master group's administration

requests is 10 minutes; the only way to edit this setting is to edit the scheduling properties for the corresponding job. To edit the properties of a local job:

1. Click the target job.
2. Click the **Properties** toolbar button.

The **Edit Job** property sheet lets you edit the following properties of a job:

- The job's next execution date and execution interval.
- The job's PL/SQL text.

### Manually Running a Job

Rather than wait for the next execution date for a job, you can force a job to run. To run a job immediately:

1. Click the target job.
2. Click the **Properties** toolbar button.
3. Click **Run Now**.

### Breaking and Enabling a Job

In certain cases, you might need to temporarily disable (break) a job and then later enable the job again. Or, you might need to enable a broken job after fixing a problem the prevented the job from running properly. For example, if you accidentally drop a database link upon which a job depends, the job will fail and eventually break (after 16 successive failures). After you recreate the necessary database link, you can then enable the broken job. To break or enable (make normal) a local job:

1. Click the target job.
2. Click the **Properties** toolbar button.
3. Click **Make Broken/Make Normal**.

### Diagnosing Problems with Jobs

Many operations in an advanced replication environment rely on jobs to perform work. While monitoring your system, you might find that there are problems related to jobs. You should use Replication Manager to monitor local jobs regularly.

If a job fails to execute, first check to see the status of the job. If the job is broken, check the alert log and trace files for error information and fix the error. If the job is not broken (normal), Oracle should ultimately re-execute it.

If the job is OK but has never executed, there may be a problem with the availability of SNP background processes. Check the initialization parameter JOB_QUEUE_PROCESSES to determine the number of background processes available and JOB_QUEUE_INTERVAL to determine how frequently each background processes wakes up. You can also query the DBA_JOBS_RUNNING view to explore what jobs these processes are currently running (you may have a problem with a runaway job), and the alert log and trace file can provide you with additional information about potential problems with the background process.

# Database Backup and Recovery in Replication Systems

Databases using advanced replication are distributed databases. Follow the guidelines for distributed database backups outlined in the *Oracle8 Administrator's Guide* when creating backups of advanced replication databases. Follow the guidelines for coordinated distributed recovery in the *Oracle8 Administrator's Guide* when recovering an advanced replication database.

If you fail to follow the coordinated distributed recovery guidelines, there is no guarantee that your advanced replication databases will be consistent. For example, a restored master site may have propagated different transactions to different masters. You may need to perform extra steps to correct for an incorrect recovery operation. One such method is to drop and recreate all replicated objects in the recovered database.

**Recommendation**: Remove pending deferred transactions and deferred error records from the restored database, and resolve any outstanding distributed transactions *before* dropping and recreating replicated objects. If the restored database was a master definition site for some replicated environments, you should designate a new master definition site before dropping and creating objects. Any snapshots mastered at the restored database should be fully refreshed, as well as any snapshots in the restored database.

To provide continued access to your data, you may need to change master definition sites (assuming the database being recovered was the master definition site), or remaster snapshot sites (assuming their master site is being recovered).

## Performing Checks on Imported Data

After performing an export/import of a replicated object or an object used by the advanced replication facility (for example, the REPSITES view), you should run the REPCAT_IMPORT_CHECK procedure in the DBMS_REPCAT package.

In the following example, the procedure checks the objects in the ACCT replicated object group at a snapshot site to ensure that they have the appropriate object identifiers and status values:

```
DBMS_REPCAT.REPCAT_IMPORT_CHECK( gname     =>   'acct',
                                 master    =>   FALSE);
```

**Additional Information:** The parameters for the REPCAT_IMPORT_CHECK procedure are described in Table 9–178 on page 9-161, and the exceptions are listed in Table 9–179 on page 9-161.

# Auditing Successful Conflict Resolution

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

## Gathering Conflict Resolution Statistics

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

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

**Additional Information**: The parameters for the REGISTER_STATISTICS procedure are described in Table 9–172 on page 9-156, and the exceptions are listed in Table 9–173 on page 9-156.

## Viewing Conflict Resolution Statistics

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

**Additional Information**: The REPRESOLUTION_STATISTICS view is described in "REPRESOLUTION_STATISTICS View" on page 10-16 and the DEFERROR view is described in "DEFERROR View" on page 10-22.

### Canceling Conflict Resolution Statistics

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

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

**Additional Information:** The parameters for the CANCEL_STATISTICS procedure are described in Table 9–97 on page 9-100.

### Deleting Statistics Information

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

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

```
DBMS_REPCAT.PURGE_STATISTICS(sname      =>  'acct_rec',
                            oname      =>  'emp',
                            start_date =>   '01-JAN-95',
                             end_date   =>   '31-MAR-95');
```

**Additional Information:** The parameters for the PURGE_STATISTICS procedure are described in Table 9–166 on page 9-152, and the exceptions are listed in Table 9–167 on page 9-152.

## Determining Differences Between Replicated Tables

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

### DIFFERENCES

The DIFFERENCES procedure compares two replicas of a table, and determines all rows in the first replica that are not in the second and all rows in the second that are not in the first. The output of this procedure is stored in two user-created tables.

The first table stores the values of the missing rows, and the second table is used to indicate which site contains each row.

## RECTIFY

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

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

1. Select one copy of the table to be the "reference" table. This copy will be used to update all other replicas of the table as needed.
2. Determine if it is necessary to check all rows and columns in the table for differences, or only a subset. For example, it may not be necessary to check rows that have not been updated since the last time that you checked for differences. Although it is not necessary to check all columns, your column list must include all columns that make up the primary key (or that you designated as a substitute identity key) for the table.
3. After determining which columns you will be checking in the table, you need to create two tables to hold the results of the comparison.

   You must create one table that can hold the data for the columns being compared. For example, if you decide to compare the EMPNO, SAL, and BONUS columns of the EMPLOYEE table, your CREATE statement would need to be similar to the one shown below.

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

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

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

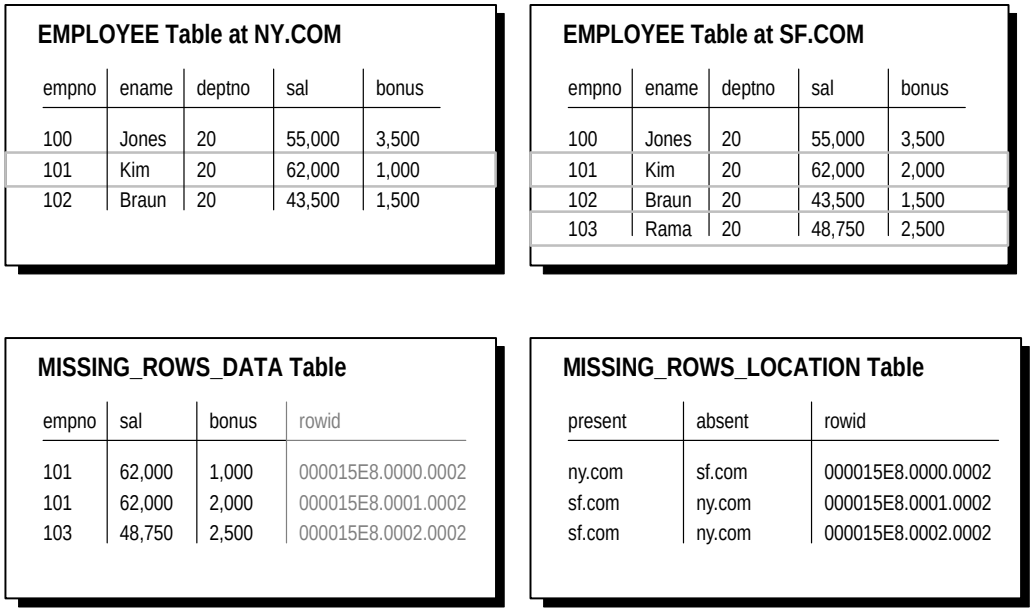
4. Suspend replication activity for the object group containing the tables that you want to compare. Although suspending replication activity for the group is not a requirement, rectifying tables that were not quiesced first can result in inconsistencies in your data.

5. At the site containing the "reference" table, call the DBMS_RECTIFIER_DIFF.DIFFERENCES procedure. For example, if you wanted to compare the EMPLOYEE tables at the New York and San Francisco sites, your procedure call would look similar to the following:

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

Figure 6–1 shows an example of two replicas of the EMPLOYEE table and what the resulting missing rows tables would look like if you executed the DIFFERENCES procedure on these replicas.

***Figure 6–1 Determining Differences Between Replicas***



**EMPLOYEE Table at NY.COM**

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

**EMPLOYEE Table at SF.COM**

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

**MISSING_ROWS_DATA Table**

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

**MISSING_ROWS_LOCATION Table**

| present | absent | rowid |
|---|---|---|
| ny.com | sf.com | 000015E8.0000.0002 |
| sf.com | ny.com | 000015E8.0001.0002 |
| sf.com | ny.com | 000015E8.0002.0002 |

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

**6.** Now you can rectify the table at the "comparison" site to be equivalent to the table at the "reference" site by calling the DBMS_RECTIFIER_DIFF.RECTIFY procedure as shown in the following example:

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

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

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

7. After you have successfully executed the RECTIFY procedure, your missing rows tables should be empty. You can now repeat steps 5 and 6 for the remaining copies of the replicated table. Remember to use the same “reference” table each time to ensure that all copies are identical when you complete this procedure.

8. You may now resume replication activity for the master group.

# Troubleshooting Common Problems

The following sections contain troubleshooting guidelines for managing an advanced replication environment.

## Diagnosing Problems with Database Links

If you think a database link is not functioning properly, you can drop and recreate it using Oracle Enterprise Manager, Oracle Server Manager, or another tool.

- Make sure that the scheduled interval is what you want.
- Make sure that the scheduled interval is not shorter than the required execution time.

If you used a connection qualifier in a database link to a given site, the other sites that link to that site must have the exact same connection qualifier. For example, if you create a database link as follows:

```
CREATE DATABASE LINK myethernet CONNECT TO repsys IDENTIFIED BY secret
USING 'connect_string_myethernet'
```

All the sites, whether masters or snapshots, associated with the `myethernet` database link must include the '`connect_string_myethernet'` connect string.

## Diagnosing Problems with Master Sites

There are a number problems that might arise in a multimaster advanced replication system. The next few sections discuss some problems and ways to troubleshoot them.

### Replicated Objects Not Created at New Master Site

If you add a new master site to a master group, and the appropriate objects are not created at the new site, try the following:

- Ensure that the necessary private database links exist between the new master site and the existing master sites. If you used the Replication Manager setup wizard to setup your sites, you should not have any problems. You must have links both to the new site from each existing site, and from the new site to each existing site.
- Make sure that the administration requests at all sites have completed successfully. If requests have not been executed yet, you can manually execute pending administration requests to complete the operation immediately.

### DDL Changes Not Propagated to Master Site

If you create a new master group object or alter the definition of a master group object at the master definition site and the modification is not propagated to a master site, first ensure that the administration requests at all sites have completed successfully. If requests are pending execution, you can manually execute them to complete the operation immediately.

When you execute DDL statements through the replication API, Oracle executes the statements on behalf of the user who submits the DDL. When a DDL statement applies to an object in a schema other than the submitter's schema, the submitter needs appropriate privileges to execute the statement. In addition, the statement must explicitly name the schema. For example, assume that you, the replication administrator, supply the following as the DDL_TEXT parameter to the DBMS_REPCAT.CREATE_MASTER_ REPOBJECT procedure:

```
CREATE TABLE scott.new_emp AS SELECT * FROM hr.emp WHERE ...;
```

Because each table name contains a schema name, this statement works whether the replication administrator is SCOTT, HR, or another user—as long as the administrator has the required privileges.

**Suggestion:** Qualify the name of every schema object with the appropriate schema.

### DML Changes Not Asynchronously Propagated to Other Sites

If you make an update to your data at a master site, and that change is not properly asynchronously propagated to the other sites in your replicated environment, try the following:

- Use Replication Manager to check whether the corresponding deferred transaction has been pushed to the destination. If not, you can also check to see how much longer it will be before the scheduled link pushes the queue to the destination site. If you do not want to wait for the next scheduled push across a link, you can execute deferred transaction manually. See "Executing Deferred Transactions" on page 6-9.
- If a scheduled link's interval has passed and corresponding deferred transactions have not been pushed, check the corresponding job for the link. See "Displaying a Site's Local Jobs" on page 6-12.
- Even after propagating a deferred transaction to a destination, it might not execute because of an error. To learn more about error transactions, see "Managing Error Transactions" on page 6-10.

### DML Cannot be Applied to Replicated Table

If you receive the DEFERRED_RPC_QUIESCE exception when you attempt to modify a replicated table, one or more master groups at your local site are "quiescing" or "quiesced". To proceed, your replication administrator must resume replication activity for the group.

### Bulk Updates and Constraint Violations

A single update statement applied to a replicated table can update zero or more rows. The update statement causes zero or more update requests to be queued for deferred execution, one for each row updated. This distinction is important when constraints are involved, because Oracle effectively performs constraint checking at the end of each statement. While a bulk update might not violate a uniqueness constraint, for example, some equivalent sequence of individual updates might violate uniqueness.

If the ordering of updates is important, update one row at a time in an appropriate order. This lets you define the order of update requests in the deferred RPC queue.

### Re-Creating a Replicated Object

If you add an object such as a package, procedure, or view to a master group, the status of the object must be VALID. If the status of an object is INVALID, recompile the object, or drop and recreate the object before adding it to a master group.

### Unable to Generate Replication Support for a Table

When you generate replication support for a table, Oracle activates an internal trigger at the local site. If the table will be propagating changes asynchronously, this trigger uses the DBMS_DEFER package to build the calls that are placed in the local deferred transaction queue. EXECUTE privileges for most of the packages involved with advanced replication, such as DBMS_REPCAT and DBMS_DEFER, need to be granted to replication administrators and users that own replicated objects. The Replication Manager setup wizard and the DBMS_REPCAT_ADMIN package performs the grants needed by the replication administrators for many typical replication scenarios. When the owner of a replicated object is not a replication administrator, however, you must explicitly grant EXECUTE privilege on DBMS_DEFER to the object owner.

### Problems with Replicated Procedures or Triggers

If you discover an unexpected unresolved conflict, and you were mixing procedural and row-level replication on a table, carefully review the procedure to ensure that the replicated procedure did not cause the conflict. Ensure that ordering

conflicts between procedural and row-level updates are not possible. Check if the replicated procedure locks the table in EXCLUSIVE mode before performing updates (or uses some other mechanism of avoiding conflicts with row-level updates). Check that row-level replication is disabled at the start of the replicated procedure and re-enabled at the end. Ensure that row-level replication is re-enabled even if exceptions occur when the procedure executes. In addition, check to be sure that the replicated procedure executed at all master sites. You should perform similar checks on any replicated triggers that you have defined on replicated tables.

## Diagnosing Problems with the Deferred Transaction Queue

If deferred transactions at a site are not being pushed to their destinations, there can be several reasons for the problem. The following sections explain some possible causes.

### Check Jobs for Scheduled Links

When you create a scheduled link, Oracle adds a corresponding job to the site's job queue. If you have scheduled a link to push deferred transactions at a periodic interval, and you encounter a problem, you should first be certain that you are not experiencing a problem with the job queue.

### Distributed Transaction Problems

When Oracle pushes a deferred transaction to a remote site using serial propagation, it uses a distributed transaction to ensure that the transaction has been properly committed at the remote site before the transaction is removed from the queue at the local site. For information on diagnosing problems with distributed transactions (two-phase commit), see the book *Oracle8 Distributed Database Systems*.

### Incomplete Database Link Specifications

If you notice that transactions are not being pushed to a given remote site, you may have a problem with how you have specified the destination for the transaction. When you create a scheduled link, you must provide the full database link name. If you use Replication Manager, you should not have any problems.

### Incorrect Replication Catalog Views

Having the wrong view definitions can lead to erroneous deferred transaction behavior. The DEFCALLDEST and DEFTRANDEST views are defined differently in CATDEFER.SQL and CATREPC.SQL. The definitions in CATREPC.SQL should be used whenever advanced replication is used. If CATDEFER.SQL is ever

(re)loaded, ensure that the view definitions in CATREPC.SQL are subsequently loaded.

## Diagnosing Problems with Snapshots

There are a number problems that might happen with snapshot sites in an advanced replication system. The next few sections discuss some problems and ways to troubleshoot them.

### Problems Creating Replicated Objects at Snapshot Site

If you unsuccessfully attempt to create a new object at a snapshot site, try the following:

- For an updatable snapshot, check that the associated master table has a snapshot log.
- Make sure that you have the necessary privileges to create the object. See "Preparing for Snapshot Site Replication" on page 4-5.
- If you are trying to add an existing snapshot to a snapshot group, try recreating the snapshot when you add it to the group.

### Problems with Snapshot Refresh

If you have a problem refreshing a snapshot, try the following:

- Check the NEXT value in the DBA_SNAPSHOTS view to determine if the refresh has been scheduled.
- If the refresh interval has passed, check the DBA_REFRESH view for the associated job number for the snapshot refresh and then follow the instructions for diagnosing a problem with job queues in "Managing Local Jobs" on page 6-12.
- You may also encounter an error if you attempt to define a master detail relationship between two snapshots. You should define master detail relationships only on the master tables by using declarative referential integrity constraints; the related snapshots should then be placed in the same refresh group to preserve this relationship.
- If you encounter a situation where your snapshots are being continually refreshed, you should check the refresh interval that you specified. This interval is evaluated before the snapshot is refreshed. If the interval that you specify is less than the amount of time it takes to refresh the snapshot, the

snapshot will be refreshed each time the SNP background process checks the queue of outstanding jobs.

- If there are any outstanding conflicts recorded at the master site for the snapshots, you can only refresh the snapshots by setting the parameter REFRESH_AFTER_ERRORS to TRUE. This parameter can be set when you create or alter a snapshot refresh group. (There is a corresponding parameter for Replication Manager property sheets.)
- If your snapshot logs are growing too large, see "Managing Snapshot Logs" on page 2-28.
- Snapshots in the same refresh groups will have their rows updated in a single transaction. Such a transaction can be very large, requiring either a large rollback segment at the snapshot site (with the rollback segment specified to be used during refresh) or you will need to use more frequent refreshes to reduce the transaction size.
- If Oracle error ORA-12004 occurs, the master site may have run out of rollback segments when trying to maintain the snapshot log, or the snapshot log may be out of date. For example, the snapshot log may have been purged or recreated. See "Managing Snapshot Logs" on page 2-28.
- Complete refreshes of a single table internally use the TRUNCATE feature to increase speed and reduce rollback segment requirements. However, until the snapshot refresh is complete, users may temporarily see no data in the snapshot. Refreshes of multiple snapshots (for example, refresh groups) do not use the TRUNCATE feature.
- Reorganization of the master table (for example, to reclaim system resources) should TRUNCATE the master table to force ROWID snapshots to do complete refreshes, otherwise, the snapshots will have incorrect references to master table ROWIDs. For more information, see "Master Table Reorganization and ROWID Snapshots" on page 2-43.

**Additional Information:** See "Troubleshooting Refresh Problems" on page 2-39.

# Updating The Comments Fields in Views

There are several procedures in the DBMS_REPCAT package that allow you to update the comment information in the various views associated with replication. Table 6–1 lists the appropriate procedure to call for each view.

***Table 6–1 Updating Comments in Advanced Replication Facility Views***

| View | DBMS_REPCAT Procedure | Additional Information |
|---|---|---|
| REPGROUP | COMMENT_ON_REPGROUP(<br>gname IN VARCHAR2,<br>Comment IN VARCHAR2) | The parameters for the COMMENT_ON_REPGROUP procedure are described in Table 9–103, and the exceptions are listed in Table 9–104. |
| REPOBJECT | COMMENT_ON_REPOBJECT(<br>sname IN VARCHAR2,<br>oname IN VARCHAR2,<br>type IN VARCHAR2,<br>comment IN VARCHAR2) | The parameters for the COMMENT_ON_REPOBJECT procedure are described in Table 9–107, and the exceptions are listed in Table 9–108. |
| REPSITES | COMMENT_ON_REPSITES(<br>gname IN VARCHAR2,<br>master IN VARCHAR,<br>comment IN VARCHAR2) | The parameters for the COMMENT_ON_REPSITES procedure are described in Table 9–105, and the exceptions are listed in Table 9–106. |
| REPCOLUMN_ GROUP | COMMENT_ON_COLUMN_GROUP(<br>sname IN VARCHAR2,<br>oname IN VARCHAR2,<br>column_group IN VARCHAR2,<br>comment IN VARCHAR2) | The parameters for the COMMENT_ON_COLUMN_GROUP procedure are described in Table 9–99, and the exceptions are listed in Table 9–100. |
| REPPRIORITY_ GROUP | COMMENT_ON_PRIORITY_GROUP(<br>gname IN VARCHAR2,<br>pgroup IN VARCHAR2)<br>comment IN VARCHAR2) | The parameters for the COMMENT_ON_PRIORITY_GROUP procedure are described in Table 9–101, and the exceptions are listed in Table 9–102. |
| REPPRIORITY_ GROUP (site priority group) | COMMENT_ON_SITE_PRIORITY(<br>gname IN VARCHAR2,<br>name IN VARCHAR2,<br>comment IN VARCHAR2) | The parameters for the COMMENT_ON_SITE_PRIORITY procedure are described in Table 9–101, and the exceptions are listed in Table 9–102. |

***Table 6–1 Updating Comments in Advanced Replication Facility Views***

| View | DBMS_REPCAT Procedure | Additional Information |
|---|---|---|
| REPRESOLUTION (uniqueness conflicts) | `COMMENT_ON_UNIQUE_RESOLUTION( sname IN VARCHAR2, oname IN VARCHAR2, constraint_name IN VARCHAR2, sequence_no IN NUMBER, Comment IN VARCHAR2)` | The parameters for the COMMENT_ON_UNIQUE_RESOLUTION procedures are described in Table 9–109, and the exceptions are listed in Table 9–110. |
| REPRESOLUTION (update conflicts) | `COMMENT_ON_UPDATE_RESOLUTION( sname IN VARCHAR2, oname IN VARCHAR2, column_group IN VARCHAR2, sequence_no IN NUMBER, Comment IN VARCHAR2)` | The parameters for the COMMENT_ON_UNIQUE_RESOLUTION procedures are described in Table 9–109, and the exceptions are listed in Table 9–110. |
| REPRESOLUTION (delete conflicts) | `COMMENT_ON_DELETE_RESOLUTION( sname IN VARCHAR2, oname IN VARCHAR2, sequence_no IN NUMBER, comment IN VARCHAR2)` | The parameters for the COMMENT_ON_UNIQUE_RESOLUTION procedures are described in Table 9–109, and the exceptions are listed in Table 9–110. |

# 7

# Advanced Techniques

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

- Using Procedural Replication.
- Using Synchronous Data Propagation.
- Designing for Survivability.
- Snapshot Cloning and Offline Instantiation.
- Security Setup for Multimaster Replication.
- Security Setup for Snapshot Replication.
- Alternative Security Setup for an Advanced Replication Snapshot Site.
- Avoiding Delete Conflicts.
- Using Dynamic Ownership Conflict Avoidance.
- Modifying Tables without Replicating the Modifications.

# Using Procedural Replication

Procedural replication can offer performance advantages for large batch-oriented operations operating on large numbers of rows that can be run serially within a replicated environment.

A good example of an appropriate application is a purge operation (also referred to as an archive operation) that you run infrequently (for example, once per quarter) during off hours to remove old data, or data that was "logically" deleted from the online database. An example using procedural replication to purge deleted rows is described in "Avoiding Delete Conflicts" on page 7-25.

## Restrictions on Procedural Replication

All parameters for a replicated procedure must be IN parameters; OUT and IN / OUT modes are not supported. The datatypes supported for these parameters are: NUMBER, DATE, VARCHAR2, CHAR, ROWID, RAW, BLOB, CLOB, NCHAR, NVARCHAR, and NCLOB.

Oracle cannot detect update conflicts produced by replicated procedures. Replicated procedures must detect and resolve conflicts themselves. Because of the difficulties involved in writing your own conflict resolution routines, it is best to simply avoid the possibility of conflicts altogether.

Adhering to the following guidelines will help you ensure that your tables remain consistent at all sites when you plan to use procedural replication.

- You must disable row-level replication within the body of the deferred procedure. For more information, see "Modifying Tables without Replicating the Modifications" on page 7-32.
- Only one replicated procedure should be executed at a time, as described in the next section, "Serialization of Transactions" on page 7-3.
- Deferred transactions should be propagated serially. For more information, see "Guidelines for Scheduled Links" on page 3-10.
- The replicated procedure must be packaged and the package cannot contain any functions. Standalone deferred procedures and standalone or packaged deferred functions are not currently supported.
- The deferred procedures must reference only locally owned data.
- The procedures should not use locally generated fields, values, or environmentally dependent SQL functions (for example, the procedure should not call SYSDATE).

- Your data ownership should be statically partitioned (that is, ownership of a row should not change between sites).

## Serialization of Transactions

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

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

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

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

## Generating Support for Replicated Procedures

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

- DBMS_REPUTIL.REPLICATION_ON
- DBMS_REPUTIL.REPLICATION_OFF

**Additional Information:** See “Disabling the Advanced Replication Facility” on page 7-33.

When you generate replication support for your replicated package, Oracle creates a wrapper package *in the schema of the replication propagator.*

**Note:** Unregistering the current propagator drops all existing generated wrappers in the propagator’s schema. Replication support for wrapped stored procedures must be regenerated after you register a new propagator.

The wrapper package has the same name as the original package, but its name is prefixed with the string you supply when you generate replication support for the procedure. If you do not supply a prefix, Oracle uses the default prefix, "defer_". The wrapper procedure has the same parameters as the original, along with two additional parameters: CALL_LOCAL and CALL_REMOTE. These two boolean parameters determine where the procedure gets executed. When CALL_LOCAL is TRUE, the procedure is executed locally. When CALL_REMOTE is TRUE, the procedure will ultimately be executed at all other master sites in the replicated environment.

The remote procedures are called directly if you are propagating changes synchronously. Or calls to these procedures are added to the deferred transaction queue if you are propagating changes asynchronously. By default, CALL_LOCAL is FALSE, and CALL_REMOTE is TRUE.

Oracle generates replication support for a package in two phases. The first phase creates the package specification at all sites. Phase two generates the package body at all sites. These two phases are necessary to support synchronous replication.

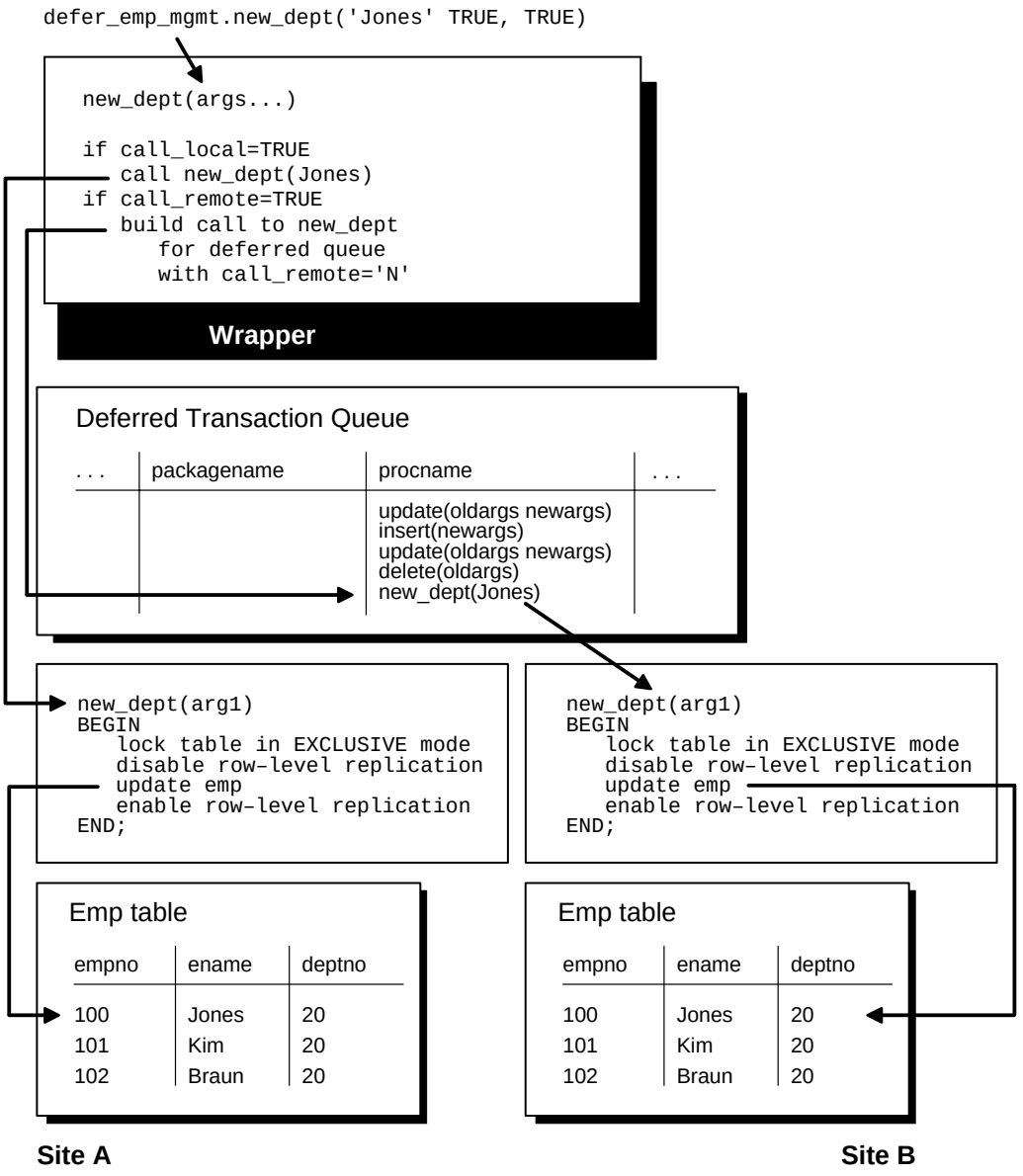
For example, suppose you create the package EMP_MGMT containing the procedure NEW_DEPT, which takes one argument, ENAME. To replicate this package to all master sites in your system, you can use Replication Manager to add the package to a master group and then generate replication support for the object. See "Managing Master Groups" on page 3-15 for more information about managing master groups and replicated objects using Replication Manager. After completing these steps, an application can call procedure in the replicated package as follows:

```
defer_emp_mgmt.new_dept( ename         => 'Jones',
                         call_local    => TRUE,
                         call_remote   => TRUE);
```

As shown in Figure 7–1, the logic of the wrapper procedure ensures that the procedure is called at the local site and subsequently at all remote sites. The logic of the wrapper procedure also ensures that when the replicated procedure is called at the remote sites, CALL_REMOTE is FALSE, ensuring that the procedure is not further propagated.

If you are operating in a mixed replicated environment with static partitioning of data ownership (that is, if you are not preventing row-level replication), the replication facility will preserve the order of operations at the remote node, since both row-level and procedural replication use the same asynchronous queue.

***Figure 7–1 Asynchronous Procedural Replication***



# Using Synchronous Data Propagation

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

**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.

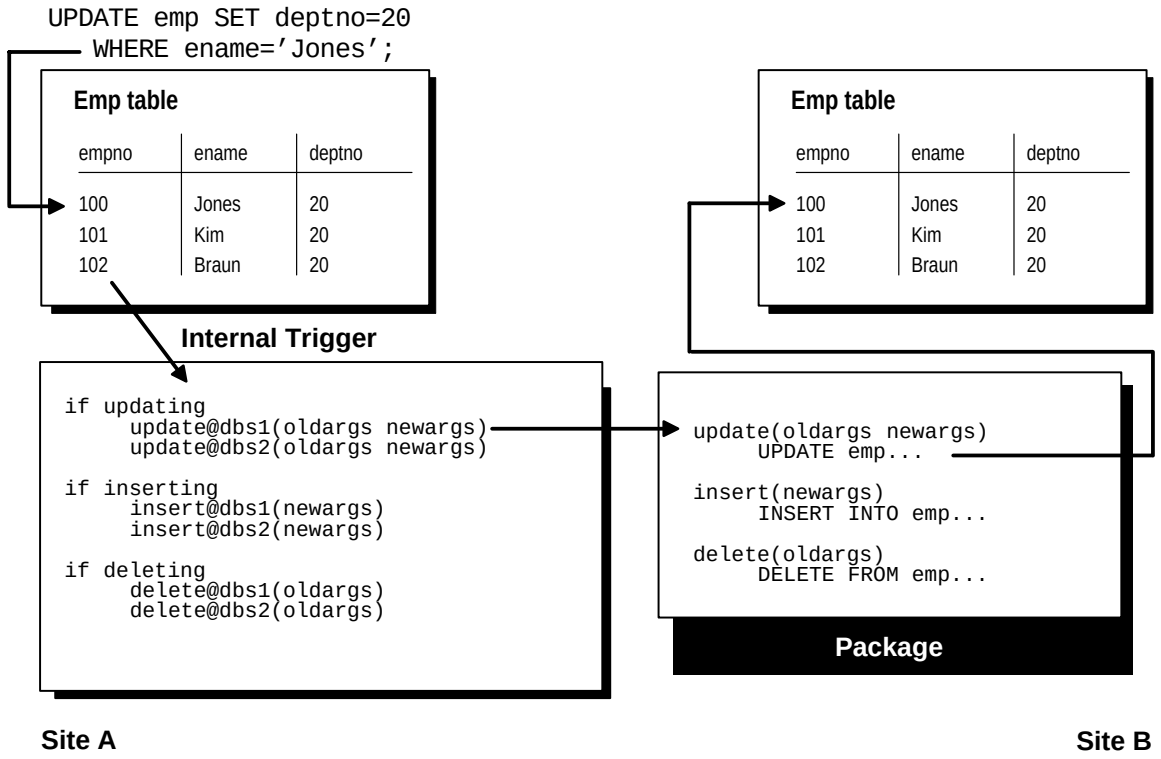
The following sections explain more about synchronous data propagation and how to manage a replicated database system that uses synchronous data propagation.

## Understanding Synchronous Data Propagation

As shown in Figure 7–2, whenever an application makes a DML change to a local replicated table and the replication group is using synchronous row-level replication, the change is synchronously propagated to the other master sites in the replicated environment using internal triggers. When the application applies a local change, the internal triggers issue calls to generated procedures at the remote master sites *in the security context of the replication propagator.* Oracle ensures that all distributed transactions either commit or rollback in the event of a failure.

**Additional Information:** See the discussion of distributed updates in the book *Oracle8 Distributed Database Systems.*

***Figure 7–2 Propagating Changes using Synchronous Row-Level Replication***




### Restrictions

Because of the locking mechanism used by synchronous replication, deadlocks can occur. When an application performs a synchronous update to a replicated table, Oracle first locks the local row and then uses an AFTER ROW trigger to lock the corresponding remote row. Oracle releases the locks when the transaction commits at each site.

### Destination of Synchronously Replicated Transactions

The necessary remote procedure calls to support synchronous replication are included in the internal trigger for each object. When you generate replication support for a replicated object, Oracle activates the triggers at all master sites to add the necessary remote procedure calls for the new site. Conversely, when you

remove a master site from a master group, Oracle removes the calls from the internal triggers.

### Conflict Detection

If all sites of a master group communicate synchronously with one another, applications should never experience replication conflicts. However, if even one site is sending changes asynchronously to another site, applications can experience conflicts at any site in the replicated environment.

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

**Additional Information**: See Chapter 5, "Conflict Resolution".

## Adding New Sites to an Advanced Replication Environment

When you add a new master or snapshot site for to a replication group in an advanced replication environment, Replication Manager allows you to select the data propagation mode (method) for the new site.

- A new master site's data propagation mode determines how the site both sends changes to and receives changes from all other master sites.
- A new snapshot site's data propagation mode determines how it sends changes to it's master site.

See Chapter 3, "Using Multimaster Replication" and Chapter 4, "Using Snapshot Site Replication" for more information about adding master and snapshot sites to an advanced replication environment, respectively.

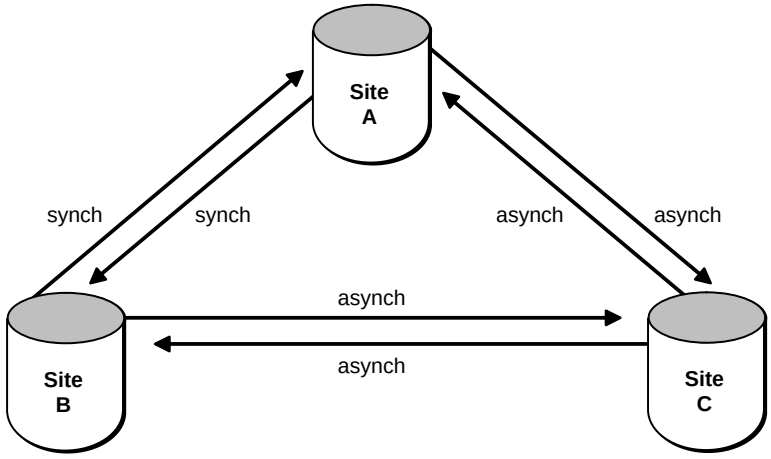
### Understanding Mixed-Mode Multimaster Systems

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

For example, suppose that you have three master sites: A, B, and C. If you first create site A as the master definition site, and then add site B with a synchronous propagation mode, site A will send changes to site B synchronously and site B will send changes to site A synchronously. There is no need to concern yourself with the scheduling of links at either site, because neither site is creating deferred transactions.

Now suppose that you create master site C with an asynchronous propagation mode. The propagation modes are now as illustrated in Figure 7–3.

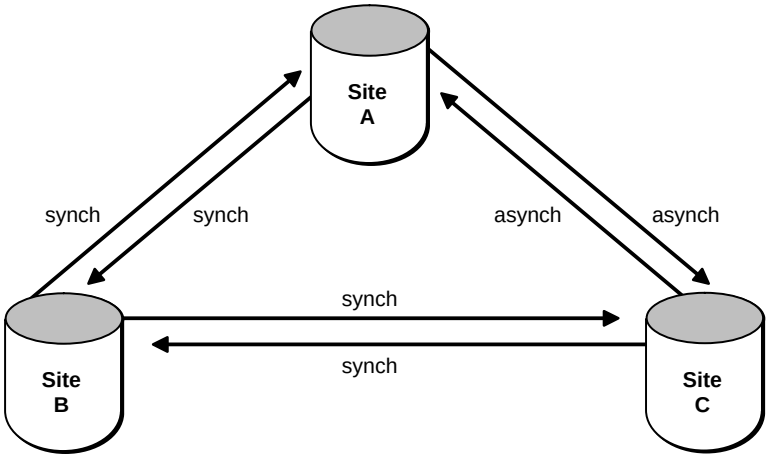
***Figure 7–3 Selecting a Propagation Mode***




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 7–4.

***Figure 7–4 Ordering Considerations***




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

**Tip:** You can view the data propagation modes between master group sites in a multimaster system quickly by using a Replication Manager destination map. See "Displaying a Destination Map for a Master Group" on page 3-29 for more information about master group destination maps.

## Altering a Master Site's Data Propagation Mode

To change the data propagation mode from one master site to another in a master group, use the destination map for the group in Replication Manager.

1. Suspend replication activity for the master group.
2. Right-click the link that you want to alter.
3. Click **Make Asynchronous** or **Make Synchronous**.
4. Resume replication activity for the master group.

**API Reference:** DBMS_REPCAT.ALTER_MASTER_PROPAGATION

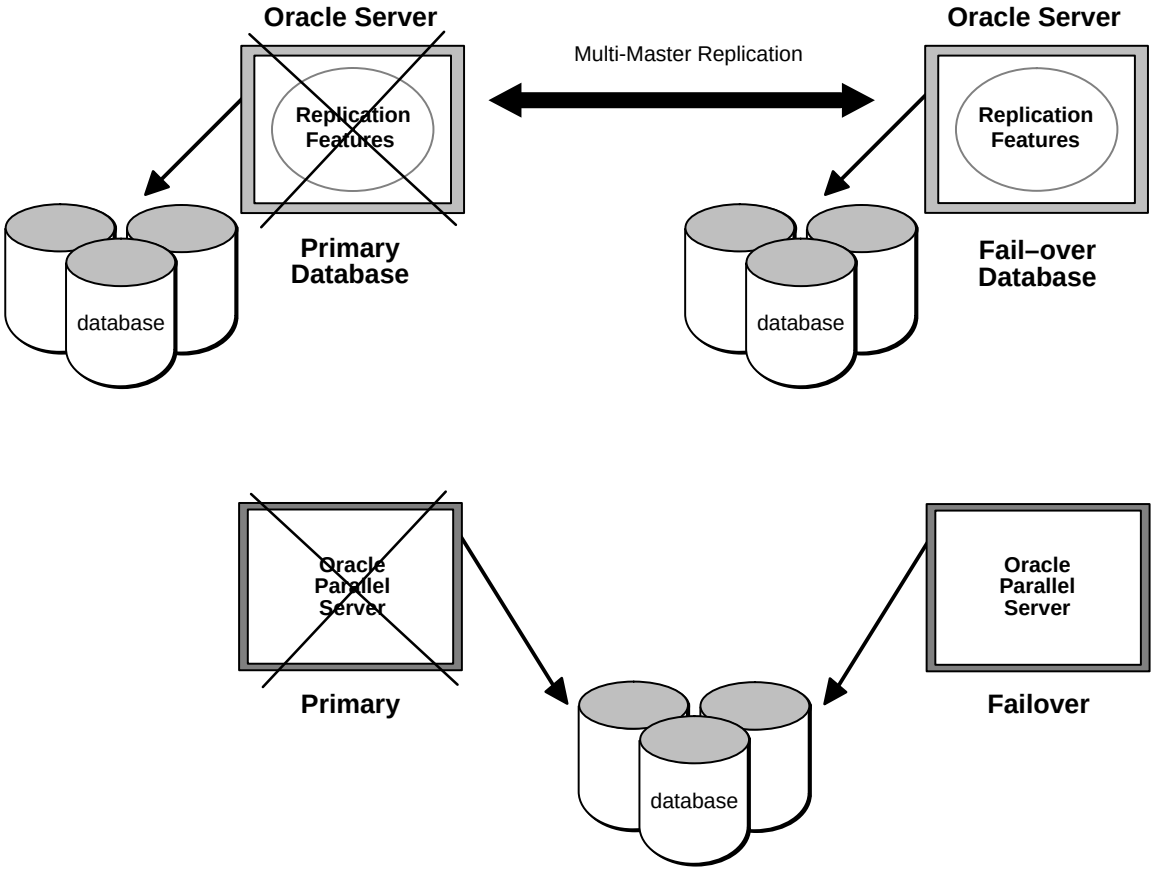
After you switch the propagation mode between one or more master sites in a master group:

- It is not necessary to regenerate replication support for any master group objects in Oracle8 databases.
- You must regenerate replication support for all master group objects in Oracle7 databases.
- If you switch from synchronous to asynchronous, be sure to schedule the propagation and purging for deferred transaction queues at the affected sites.

## Designing for Survivability

Survivability provides the capability to continue running applications despite system or site failures. Survivability allows you to run applications on a fail-over system, accessing the same, or very nearly the same, data as these systems accessed on the primary system when it failed. As shown in Figure 7–5, the Oracle Server provides two different technologies for accomplishing survivability: the Oracle Parallel Server and the advanced replication facility.

***Figure 7–5 Survivability Methods: Advanced Replication vs. Parallel Server***




## Oracle Parallel Server versus Advanced Replication

The Oracle Parallel Server supports fail-over to surviving systems when a system supporting an instance of the Oracle Server fails. The Oracle Parallel Server requires a cluster or massively parallel hardware platform, and thus is applicable for protection against processor system failures in the local environment where the cluster or massively parallel system is running.

In these environments, the Oracle Parallel Server is the ideal solution for survivability — supporting high transaction volumes with no lost transactions or data inconsistencies in the event of an instance failure. If an instance fails, a

surviving instance of the Oracle Parallel Server automatically recovers any incomplete transactions. Applications running on the failed system can execute on the fail-over system, accessing all data in the database.

The Oracle Parallel Server does not, however, provide survivability for site failures (such as flood, fire, or sabotage) that render an entire site, and thus the entire cluster or massively parallel system, inoperable. To provide survivability for site failures, you can use the advanced replication facility to maintain a replica of a database at a geographically remote location.

Should the local system fail, the application can continue to execute at the remote site. Advanced replication, however, cannot guarantee the protection of all transactions. Also, special care must be taken to prevent data inconsistencies when recovering the primary site.

**Note:** You can also configure a standby-database to protect an Oracle database from site failures. For more information about Oracle's standby database feature, see the *Oracle8 Backup and Recovery Guide*.

## Designing for Survivability

If you choose to use the advanced replication facility for survivability, you should consider the following issues:

- The advanced replication facility must be able to keep up with the transaction volume of the primary system. This is application specific, but generally much lower than the throughput supported if you are using the Oracle Parallel Server.
- If a failure occurs at the primary site, recently committed transactions at the primary site may not have been asynchronously propagated to the fail-over site yet. These transactions will appear to be lost.
- These "lost" transactions must be dealt with when the primary site is recovered.

  Suppose, for example, you are running an order-entry system that uses replication to maintain a remote fail-over order-entry system, and the primary system fails.

  At the time of the failure, there were two transactions recently executed at the primary site that did not have their changes propagated and applied at the fail-over site. The first of these was a transaction that entered a new order, and the second was a transaction that cancelled an existing order.

  In the first case, someone may notice the absence of the new order when processing continues on the fail-over system, and re-enter it. In the second case,

the cancellation of the order may not be noticed, and processing of the order may proceed; that is, the canceled item may be shipped and the customer billed.

What happens when you restore the primary site? If you simply push all of the changes executed on the fail-over system back to the primary system, you will encounter conflicts.

Specifically, there will be duplicate orders for the item originally ordered at the primary system just before it failed. Additionally, there will be data changes resulting from the transactions to ship and bill the order that was originally canceled on the primary system.

You must carefully design your system to deal with these situations. The next section explains this process.

### Implementing a Survivable System

Oracle's advanced replication facility can be used to provide survivability against site failures by using multiple replicated master sites. You must configure your system using one of the following methods. These methods are listed in order of increasing implementation difficulty.

- The fail-over site is used for read access only. That is, no updates are allowed at the fail-over site, even when the primary site fails.
- After a failure, the primary site is restored from the fail-over site using export/import, or via full backup.
- Full conflict resolution is employed for all data/transactions. This requires careful design and implementation. You must ensure proper resolution of conflicts that can occur when the primary site is restored, such as duplicate transactions.
- Provide your own special applications-level routines and/or procedures to deal with the inconsistencies that occur when the primary site is restored, and the queued transactions from the active fail-over system are propagated and applied to the primary site.

## Snapshot Cloning and Offline Instantiation

By default, Oracle builds and populates replicas when you:

- Create a snapshot in a basic replication environment.
- Add master site or snapshot site to an advanced replication environment.

When building a large replicated environment, the amount of data necessary to build and populate replicas throughout the system can generate an excessive amount of network traffic. To avoid saturating the network with the data necessary to build a large replicated environment, Oracle lets you perform offline instantiation of new sites in both basic and advanced replication systems. The following sections explain how to clone snapshots in a basic replication environment and offline instantiate master and snapshot sites in an advanced replication environment.

## Snapshot Cloning for Basic Replication Environments

To reduce the network overhead associated with the creation of the same set of snapshots in many databases, you can perform snapshot "cloning" by following these steps:

1. Create the snapshots and corresponding refresh groups that you want to clone in a database separate from the database that contains the master tables. The database in which you create snapshots of the master table data is the template snapshot database for the cloning process. If you are not familiar with creating snapshots in a basic replication environment, see Chapter 2, "Using Basic Replication".
2. At the template snapshot database, export the schemas containing the snapshots and refresh groups that you want to clone to other snapshot databases. The Export utility does not support the export of individual snapshots. Therefore, you must export snapshots at the schema level.
3. Prepare all other snapshot databases for snapshots of the master database. If you are not familiar with setting up the accounts and database links necessary to prepare a snapshot database, see "Preparing a Database for Snapshots" on page 2-4.

   **Note:** To make sure that you have prepared a snapshot database properly, connect to each snapshot schema in the database. Next, execute the defining queries of the proposed snapshots to ensure that they execute without error. Additionally, check to make sure that each new snapshot database has enough free space to hold the new snapshots.
4. At each new snapshot database, import the snapshot schemas that were exported from the template snapshot database. This creates and populates the same snapshots that were exported from the template snapshot database.
5. Perform a fast refresh of all fast-refreshable snapshots. Doing so registers the new snapshots in the master database.

## Offline Instantiation of a Master Site in an Advanced Replication System

Offline instantiation of a master site lets you add the new master site to a master group while limiting the amount of downtime required for existing sites in the multimaster replication system. Offline instantiation of a master site is useful primarily for systems with very large databases where the time required to transfer the data through network links to a new site would be prohibitive.

Assuming that you are using a default replication configuration that uses asynchronous row-level replication, the steps necessary to create a new master site using offline instantiation are as follows:

1. Prepare the new master site with the Replication Manager setup wizard. At the new master site, make sure to create the schemas that will contain objects of the master group. If you are not sure how to prepare a master site, see "The Replication Setup Wizard" on page 3-4 and "Adding a Master Site to a Master Group" on page 3-24 for more information.
2. Synchronize the master sites that replicate the master group. This process includes pushing of all outstanding administration requests at each master site, resolving any administration requests that have an error status, resolving error transactions at each master site, and so on.
3. Suspend replication activity for the master group.

   **Warning:** Do not resume replication activity or do other replication administration for the master group until the new master site appears at the master definition site. Otherwise, changes that you make at any site will not be propagated to the new site, and you might have problems synchronizing the group's data.
4. From the group's master definition site, call the replication management API procedure DBMS_OFFLINE_OG.BEGIN_INSTANTIATION with the parameters *gname* (the name of the master group) and *new_site* (the name of the new master site). This procedure adds the new site to the master group.
5. Export the individual objects in the master group from any master site. When a master group contains replication objects that originate from more than one schema, make sure to export the objects from *each* schema.
6. From the master definition site, call the replication management API procedure DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS with the parameters *gname* (the name of the master group) and *new_site* (the name of the new master site).

At this point, normal non-administrative activities can resume at the existing master sites. However, replication activity for the group remains suspended at the new site.

7. Transfer the export file to the new master site.
8. At the new master site, call the replication management API procedure DBMS_OFFLINE_OG.BEGIN_LOAD with the parameters *gname* (the name of the master group) and *new_site* (the name of the new master site). This procedure ensures that applications cannot update master group objects while you import master group data in the next step, disables internal replication triggers, and disables propagation of the deferred RPC queue from all other master sites to the new site.
9. Use the Import utility to import the data from the Export file.
10. When the import is complete at the new master site, call the replication management API procedure DBMS_OFFLINE_OG.END_LOAD (*gname*, *new_site*) to enable the new site.
11. Return to the master definition site and call the procedure DBMS_OFFLINE_OG.END_INSTANTIATION with the parameters *gname* (the name of the master group) and *new_site* (the name of the new master site). This procedure resumes replication activity at the new site.

**Additional Information:** See the book *Oracle8 Utilities* to learn about the Import and Export utilities.

Please note the following:

- When you add multiple new sites, you must perform the above steps for each new site. You cannot instantiate a number of new sites as a group.
- Be careful to call the listed procedures from the appropriate site. Failure to do so could cause unexpected results.

## Offline Instantiation of a Snapshot Site in an Advanced Replication System

Offline instantiation of a snapshot site in an advanced replication environment is useful when you need to create a large snapshot site, and the time required to transfer replication data through network to the new site would be prohibitive.

Use the following steps to create a snapshot site using offline instantiation:

**At the Master Site** **1.** Before you begin, create a snapshot log for each master table.

2. Create a snapshot of each master group table. Create each snapshot in the same schema as its master table. To accomplish this, each snapshot's name must be different from its corresponding master table. Additionally, the snapshot's defining query must use a loopback database link to reference the master table. For example, to create a snapshot EMPLOYEE of the EMP table in the same database DBS1:

   ```
   CREATE SNAPSHOT sales.employee AS SELECT * FROM sales.emp@dbs1
   ```

   **Attention:** Before creating snapshots, make sure that the master database has ample storage space.

3. As schema owner, use Export to export all schemas that contain the snapshots.
4. Drop the snapshots at the master site that were created for offline instantiation.
5. Transfer the Export file(s) to the new snapshot site.

**At a New Snapshot Site**

1. Using the Replication Manager setup wizard, prepare the new snapshot site to communicate with the master site. If you are not familiar with this process, see "Preparing for Snapshot Site Replication" on page 4-5.
2. Using the Replication Manager snapshot group wizard, create an empty snapshot group to correspond with the master group. Do not choose to replicate any objects of the master group.
3. For each schema and snapshot, use the procedure DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD (*gname*, *sname*, *master_site*, *snapshot_oname*) to create empty snapshots in the specified schema and object group at the new snapshot site, as well as all the necessary supporting objects.
4. Import only the snapshot base tables from the Export file(s).
5. When the import is complete, for each schema and snapshot, use the procedure DBMS_OFFLINE_SNAPSHOT.END_LOAD (*gname, sname, snapshot_oname*).

**Additional Information:** See the book *Oracle8 Utilities*.

## Security Setup for Multimaster Replication

Nearly all users should find it easiest to use the Replication Manager setup wizard when configuring multimaster replication security. However, for certain case you may need to use the replication management API to perform these setup operations.

To configure a replication environment the replication administrator must have DBA privileges including the ability to connect as SYS.

First set up user accounts at each master site with the appropriate privileges to configure and maintain the replication environment and to propagate and apply replicated changes. You must also define links for users at each master site.

In addition to the end users who will access replicated objects, there are three special categories of "users" in a replication environment:

- Replication administrators who are responsible for configuring and maintaining a replication environment.
- Propagators who are responsible for propagating deferred transactions.
- Receivers at remote sites who are responsible for applying these transactions.

Typically, a single user acts as administrator, propagator, and receiver. However, you can have separate users perform each of these functions. You can choose to have a single, global replication administrator or, if your replication groups do not span schema boundaries, you may prefer to have separate replication administrators for different schemas. Note, however, that you can have only one registered propagator for each database.

Table 7–1 describes the necessary privileges that must be assigned to these specialized accounts. Most privileges needed by these users are granted to them through calls to the replication management API. You will also need to grant certain privileges directly.

***Table 7–1 Required User Accounts***

| User | Privileges |
|---|---|
| global replication administrator | DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA |
| schema-level replication administrator | DBMS_REPCAT_ADMIN.GRANT_ADMIN_SCHEMA |
| propagator | DBMS_DEFER_SYS.REGISTER_PROPAGATOR |
| receiver | grant EXECUTE ANY PROCEDURE<br>grant CREATE SESSION<br>or, if you have stricter security requirements:<br>grant CREATE SESSION<br>grant EXECUTE on DBMS_DEFER_INTERNAL_SYS<br>grant EXECUTE on necessary $RP, $RL and wrapper packages |

After you have created these accounts, create the following private database links, including username and password between each site:

- From the local replication administrator to the remote replication administrator.
- From the local propagator to the remote receiver.

Assuming you have designated a single user account to act as replication administrator, propagator, and receiver, you will need to create N(N-1) links, where N is the number of master sites in your replication environment.

After creating these links, you must call DBMS_DEFER_SYS.SCHEDULE_PUSH and DBMS_DEFER_SYS.SCHEDULE_PURGE, at each location, to define how frequently you want to propagate your deferred transaction queue to each remote location, and how frequently you wish to purge this queue. You will need to call DBMS_DEFER_SYS.SCHEDULE_PUSH multiple times at each site, once for each remote location.

A sample script for setting up multimaster replication between HQ.WORLD and SALES.WORLD is shown below:

```
/*--- Create global replication administrator at HQ ---*/
connect system/manager@hq.world
create user repadmin identified by repadmin
execute dbms_repcat_admin.grant_admin_any_schema(username => 'repadmin')

/*--- Create global replication administrator at Sales ---*/
connect system/manager@sales.world
create user repadmin identified by repadmin
execute dbms_repcat_admin.grant_admin_any_schema(username => 'repadmin')

/*--- Create single user to act as both propagator and receiver at HQ ---*/
connect system/manager@hq.world
create user prop_rec identified by prop_rec
/*--- Grant privileges necessary to act as propagator ---*/
execute dbms_defer_sys.register_propagator(username => 'prop_rec')
/*--- Grant privileges necessary to act as receiver ---*/
grant execute any procedure to prop_rec
grant create session to prop_rec

/*--- Create single user to act as both propagator and receiver at Sales ---*/
connect system/manager@sales.world
create user prop_rec identified by prop_rec
/*--- Grant privileges necessary to act as propagator ---*/execute
dbms_defer_sys.register_propagator(username => 'prop_rec')
/*--- Grant privileges necessary to act as receiver ---*/
```

```
grant execute any procedure to prop_rec
grant create session to prop_rec

/*--- Create public link from HQ to Sales with necessary USING clause ---*/
connect system/manager@hq.world
create public database link sales.world using sales.world

/*--- Create private repadmin to repadmin link ---*/
connect repadmin/repadmin@hq.world
create database link sales.world connect to repadmin identified by repadmin

/*--- Schedule replication from HQ to Sales ---*/
execute dbms_defer_sys.schedule_push(
     destination => 'sales.world',
     interval => 'sysdate + 1/24',
     next_date => sysdate,
     stop_on_error => FALSE,
     delay_seconds => 0,
     parallelism => 1)

/*--- Schedule purge of def tran queue at HQ ---*/
execute dbms_defer_sys.schedule_purge(
     next_date => sysdate,
     interval = 'sysdate + 1',
     delay_seconds => 0,
     rollback_segment =>'')

/*--- Create link from propagator to receiver for scheduled push ---*/
connect prop_rec/prop_rec@hq.world
create database link sales.world connect to prop_rec identified by prop_rec

/*--- Create public link from Sales to HQ with necessary USING clause ---*/
connect system/manager@sales.world
create public database link hq.world using hq.world

/*--- Create private repadmin to repadmin link ---*/
connect repadmin/repadmin@sales.world
create database link hq.world connect to repadmin identified by repadmin

/*--- Schedule replication from Sales to HQ ---*/
execute dbms_defer_sys.schedule_push(
     destination => 'hq.world',
     interval => 'sysdate + 1/24',
     next_date => sysdate,
     stop_on_error => FALSE,
```

```
     delay_seconds => 0,
     parallelism => 1)

/*--- Schedule purge of def tran queue at Sales ---*/
execute dbms_defer_sys.schedule_purge(
     next_date => sysdate,
     interval = 'sysdate + 1',
     delay_seconds => 0,
     rollback_segment =>'')

/*--- Create link from propagator to receiver for scheduled push ---*/
connect prop_rec/prop_rec@sales.world
create database link hq.world connect to prop_rec identified by prop_rec
```

## Security Setup for Snapshot Replication

Nearly all users should find it easiest to use the Replication Manager setup wizard when configuring snapshot replication security. However, for certain specialized cases, you may need to use the replication management API to perform these setup operations.

To configure a replication environment you must have DBA privileges, including the ability to connect as SYS.

First set up user accounts at each snapshot site with the appropriate privileges to configure and maintain the replication environment and to propagate replicated changes. You must also define links for these users to the associated master site. You may need to create additional users, or assign additional privileges to users at the associated master site.

In addition to end users who will be accessing replicated objects, there are three special categories of "users" at a snapshot site:

- Replication administrators who are responsible for configuring and maintaining a replication environment.
- Propagators who are responsible for propagating deferred transactions.
- Refreshers who are responsible for pulling down changes to the snapshots from the associated master tables.

Typically, a single user performs each of these functions. However, there may be situations where you need different users performing these functions. For example, snapshots may be created by a snapshot site administrator and refreshed by another end user.

Table 7–2 describes the privileges needed to create and maintain a snapshot site.

***Table 7–2 Required Snapshot Site User Accounts***

| User | Privileges |
|---|---|
| snapshot site replication administrator | DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA |
| propagator | DBMS_DEFER_SYS.REGISTER_PROPAGATOR |
| refresher | CREATE ANY SNAPSHOT<br>ALTER ANY SNAPSHOT |

In addition to creating the appropriate users at the snapshot site, you may need to create additional users at the associated master site, as well. Table 7–3 describes the privileges need by master site users to support a new snapshot site.

***Table 7–3 Required Master Site User Accounts***

| User | Privileges |
|---|---|
| proxy snapshot site administrator | DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA<br>or, grant the following privileges:<br>CREATE SESSION, CREATE ANY TRIGGER, CREATE ANY PROCEDURE, SELECT ANY TABLE, and EXECUTE on DBMSOBJGWRAPPER, DBMS_REPCAT_UTL2, DBMS_REPCAT_UNTRUSTED<br>in addition, the proxy admin requires the following privileges in order to create snapshot logs:<br>COMMENT ANY TABLE, CREATE ANY TABLE, and SELECT ANY TABLE or SELECT on master tables |
| receiver | grant EXECUTE ANY PROCEDURE<br>grant CREATE SESSION<br>or<br>grant CREATE SESSION<br>grant EXECUTE on DBMS_DEFER_INTERNAL_SYS<br>grant EXECUTE on necessary $RP, $RL and wrapper packages |

***Table 7–3 Required Master Site User Accounts***

| User | Privileges |
|---|---|
| proxy refresher | grant CREATE SESSION<br>grant SELECT ANY TABLE<br><br>or<br><br>grant CREATE SESSION<br>grant SELECT on necessary master tables and snapshot logs |

After creating the accounts at both the snapshot and associated master sites, you need to create the following private database links, including username and password, from the snapshot site to the master:

- From the snapshot replication administrator to the proxy snapshot replication administrator.
- From the propagator to the receiver.
- From the refresher to the proxy refresher.

Assuming you have designated a single user account to act as replication administrator, propagator, and receiver, you will need to create one link for each snapshot site. You do not need a link from the master site to the snapshot site.

After creating these links, you must call DBMS_DEFER_SYS.SCHEDULE_PUSH and DBMS_DEFER_SYS.SCHEDULE_PURGE at the snapshot site to define how frequently you want to propagate your deferred transaction queue to the associated master site, and how frequently you wish to purge this queue. You must also call DMBS_REFRESH. REFRESH at the snapshot site to schedule how frequently to pull changes from the associated master site.

## Alternative Security Setup for an Advanced Replication Snapshot Site

You can configure snapshot site security using an alternative approach that provides both greater simplicity and security. Set up this configuration using the replication management API. The Replication Manager setup wizard does not support this approach, nor can you use Replication Manager, in most cases, to administer snapshot sites that use this configuration.

This approach requires that all snapshots at the snapshot site, and corresponding master tables at the master site, be contained within a single schema. The schema owner is then authorized as the primary snapshot site administration for the specific schema and as the propagator for the snapshot site. Similarly, the schema owner is then authorized as the snapshot replication receiver at the master site.

This approach is simpler and more secure because the snapshot site administrator does not need privileges to administer objects in other schemas.

A sample script implementing this approach between HQ, as the master site, and Sales, as the snapshot site, for the order_entry schema is shown below.

```
/*--- Grant snap_repadmin privileges to schema owner at snapshot site ---*/
connect system/manager@sales.world
execute dbms_repcat_admin.grant_admin_schema(username => 'order_entry')

/*--- Register schema owner as propagator at snapshot site ---*/
execute dbms_defer_sys.register_propagator(username => 'order_entry')

/*--- Grant proxy snap_repadmin privileges to schema owner at master site ---*/
connect system/manager@hq.world
grant execute on sys.dbmsobjgwrapper to order_entry
grant execute on sys.dbms_defer_internal_sys to order_entry
grant execute on sys.dbms_repcat_untrusted to order_entry
grant execute on sys.dbms_repcat_utl2 to order_entry

/*--- Create link from snapshot schema owner to master schema owner ---*/
connect order_entry/order_entry@sales.world
create database link hq.world
  connect to 'order_entry' identified by 'order_entry' using 'hq.world'
```

## Avoiding Delete Conflicts

To avoid encountering delete conflicts, you might find it easiest to mark rows as deleted and purge them later. This section outlines a simple technique for purging these marked rows using procedural replication.

Suppose that your database contains the following MAIL_LIST table:

```
Name                 Null?      Type
-------------------- ---------- --------     --------------
CUSTNO               NOT NULL   NUMBER(4)    PRIMARY KEY
CUSTNAME                        VARCHAR2(10)
ADDR1                           VARCHAR2(30)
ADDR2                           VARCHAR2(30)
CITY                            VARCHAR2(30)
STATE                           VARCHAR2(2)
ZIP                             NUMBER(9)
PHONE                           NUMBER(10)
REMOVE_DATE                    DATE
```

Instead of deleting a customer when he or she requests to be removed from your mailing list, the REMOVE_DATE column would be used to indicate former customers; A NULL value would be used for current customers. After customers request removal from the mailing list, their rows are no longer updated. Such a convention avoids conflicts when the rows are actually deleted sometime later. A view of current customers could be defined as follows:

```
CREATE OR REPLACE VIEW corp.current_mail_list AS
 SELECT custno, custname, addr1, addr2, city, state, zip, phone
  FROM corp.mail_list WHERE remove_date IS NULL;
```

Periodically, perhaps once a year after the holiday sales, the former customers would be purged from the table using the REMOVE_DATE field. Such a delete could be performed using row-level replication just by performing the following delete:

```
DELETE corp.mail_list
  WHERE remove_date IS NOT NULL AND remove_date<'01-JAN-95';
```

However, for a large company with an extensive mail order business, the number of former customers could be quite large resulting in a lot of undesired network traffic and database overhead. Instead, the procedural replication could be used using the following package:

```
CREATE OR REPLACE PACKAGE corp.purge AS
 PROCEDURE remove_cust(purge_date IN DATE);
END;
/
CREATE OR REPLACE PACKAGE BODY corp.purge AS
 PROCEDURE remove_cust(purge_date IN DATE) IS
 BEGIN
  -- turn off row-level replication for set delete
  dbms_reputil.replication_off;
  -- prevent phantom reads
  LOCK TABLE corp.mail_list IN EXCLUSIVE MODE;
  DELETE corp.mail_list WHERE remove_date IS NOT NULL AND
                              remove_date < purge_date;
  dbms_reputil.replication_on;
 EXCEPTION WHEN others THEN
  dbms_reputil.replication_on;
 END;
END;
```

The DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT procedure would have been used to generate the DEFER_PURGE package during the initial

replication setup. Then, the procedural replication package could be called as follows by a single master site:

```
BEGIN
 defer_purge.remove_cust('14-APR-97','Y');
END;
```

The procedure, PURGE.REMOVE_CUST, would be executed locally and asynchronously executed at each master, resulting in many rows being deleted with only minimal network traffic.

To ensure that there are no outstanding transactions against the rows to be purged, your application should be written to *never* update logically deleted rows and the REMOVE_DATE should be old enough to ensure that the logical delete of the row is propagated before the row is purged. Thus, in the previous example, it is probably not necessary to lock the table in EXCLUSIVE mode; although this is another method of guaranteeing that these rows not be updated during the purge.

## Using Dynamic Ownership Conflict Avoidance

This section describes a more advanced method of designing your applications to avoid conflicts. This method, known as *token passing*, is similar to the workflow method described in Chapter 1. Although this section describes how to use this method to control the ownership of an entire row, you can use a modified form of this method to control ownership of the individual column groups within a row.

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

### Workflow

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

Take the simple example of separate sites for ordering, shipping, and billing. Here, the identifier columns are used to indicate the status of an order. The status determines which site can update the row. After a user at the ordering site has entered the order, he or she updates the status of this row to SHIP. Users at the

ordering site are no longer allowed to modify this row — ownership has been pushed to the shipping site.

After shipping the order, the user at the shipping site will update the status of this row to BILL, thus pushing ownership to the billing site, and so on.

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

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

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

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

The PRIORITY conflict resolution method successfully converges for more than one master as long as the priority value is always increasing.

## Token Passing

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

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

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

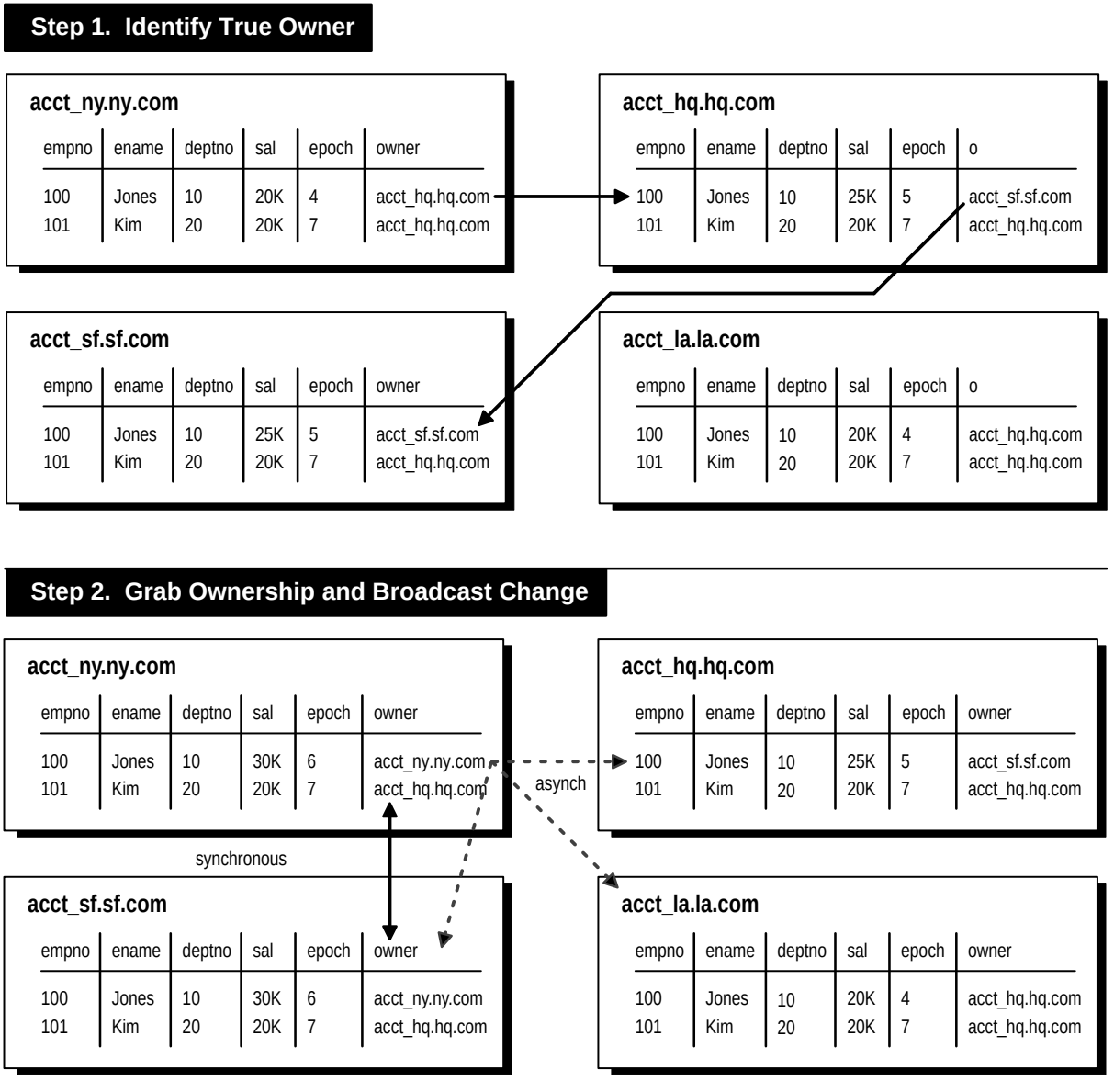
Whenever you attempt to update a row, your application should:

1. Locate the current owner of the row.
2. Lock the row to prevent updates while ownership is changing.
3. Grab ownership of the row.

4. Perform the update. (Oracle releases the lock when you commit your transaction.)

For example, Figure 7–6 illustrates how ownership of employee 100 passes from the ACCT_SF database to the ACCT_NY database.

***Figure 7–6 Grabbing the Token***



## Locating the Owner of a Row

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

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

## Obtaining Ownership

After locating the owner of the row, the ACCT_NY site gets ownership from the ACCT_SF site by completing the following steps:

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

3. Update the row information at the new owner site, NY, with the information from the current owner site, SF. This data is guaranteed to be the most recent. This time, the change at the NY site should not be replicated. Any queued changes to this data at the SF site will be propagated to all other sites in the usual manner. When the SF change is propagated to NY, it will be ignored because of the values of the epoch numbers, as described in the next bullet point.
4. Update the epoch number at the new owner site to be one greater than the value at the previous site. Perform this update at the new owner only, and then asynchronously propagate this update to the other master sites. Incrementing the epoch number at the new owner site prevents ordering conflicts.

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

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

### Applying the Change

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

## Modifying Tables without Replicating the Modifications

You may encounter a situation where you need to modify a replicated object, but you do not want this modification replicated to the other sites in the replicated environment. For example, you might want to disable replication in the following situations:

- When you are using procedural replication to propagate a change, you should always disable row-level replication at the start of your procedure.
- You may need to disable replication in triggers defined on replicated tables to avoid replicating trigger actions multiple times. See “Triggers and Replication” on page 7-34.

- Sometimes when you manually resolve a conflict, you might not want to replicate this modification to the other copies of the table.

  You might need to do this, for example, if you need to correct the state of a record at one site so that a conflicting replicated update will succeed when you reexecute the error transaction. Or you might use an unreplicated modification to undo the effects of a transaction at its origin site because the transaction could not be applied at the destination site. In this example, you can use Replication Manager to delete the conflicting transaction from the destination site.

To modify tables without replicating the modifications, use the REPLICATION_ON and REPLICATION_OFF procedures in the DBMS_REPUTIL package. These procedures take no arguments and are used as flags by the generated replication triggers.

**Note:** To enable and disable replication, you must have the EXECUTE privilege on the DBMS_REPUTIL package.

## Disabling the Advanced Replication Facility

The DBMS_REPUTIL.REPLICATION_OFF procedure sets the state of an internal replication variable for the current session to FALSE. Because all replicated triggers check the state of this variable before queuing any transactions, modifications made to the replicated tables that use row-level replication do not result in any queued deferred transactions.

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

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

```
CREATE OR REPLACE PACKAGE update AS
  PROCEDURE update_emp(adjustment IN NUMBER);
END;
/
CREATE OR REPLACE PACKAGE BODY update AS
  PROCEDURE update_emp(adjustment IN NUMBER) IS
  BEGIN
   -- turn off row-level replication for set update
   dbms_reputil.replication_off;
```

```
    UPDATE emp . . .;
    -- re-enable replication
    dbms_reputil.replication_on;
   EXCEPTION WHEN OTHERS THEN
    . . .
    dbms_reputil.replication_on;
   END;
END;
```

## Re-enabling the Advanced Replication Facility

After resolving any conflicts, or at the end of your replicated procedure, be certain to call DBMS_REPUTIL.REPLICATION_ON to resume normal replication of changes to your replicated tables or snapshots. This procedure takes no arguments. Calling REPLICATION_ON sets the internal replication variable to TRUE.

## Triggers and Replication

If you have defined a replicated trigger on a replicated table, you may need to ensure that the trigger fires only once for each change that you make. Typically, you will only want the trigger to fire when the change is first made, and you will not want the remote trigger to fire when the change is replicated to the remote site.

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

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

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

## Enabling/Disabling Replication for Snapshots

To disable all local replication triggers for snapshots at your current site, set the internal refresh variable to TRUE by calling SET_I_AM_A_REFRESH, as shown in the following example:

```
DBMS_SNAPSHOT.SET_I_AM_A_REFRESH(value => TRUE);
```

To re-enable the triggers, set the internal refresh variable to FALSE, as shown below:

```
DBMS_SNAPSHOT.SET_I_AM_A_REFRESH(value => FALSE);
```

To determine the value of the internal refresh variable, call the I_AM_A_REFRESH function as shown below:

```
ref_stat := DBMS_SNAPSHOT.I_AM_A_REFRESH;replication:advanced
techniques<$startrange>;advanced replication:techniques<$startrange>
```

# 8

# Using Deferred Transactions

This chapter describes the following topics:

- Listing Information about Deferred Transactions.
- Creating a Deferred Transaction.
- LOB Storage.

# Listing Information about Deferred Transactions

Oracle provides several tables and views for you to use in administering deferred transactions. These views provide information about each deferred transaction, such as the transaction destinations, the deferred calls that make up the transaction, and any errors encountered during attempted execution of the transaction.

**Attention:** You should not modify these tables directly; use the procedures provided in the DBMS_DEFER and DBMS_DEFER_SYS packages.

These views are briefly described below. For more information, see Chapter 10, "Data Dictionary Views".

| Data Dictionary View | Description |
|---|---|
| DEFCALL | Records all deferred remote procedure calls (RPCs). |
| DEFCALLDEST | Lists the destinations for each deferred remote procedure call. |
| DEFDEFAULT DEST | Lists the default destination for deferred remote procedure calls. |
| DEFERROR | Provides information about transactions that could not be applied. |
| DEFLOB | Storage for LOB parameters to deferred RPCs. |
| DEFSCHEDULE | Displays information about when a job is next scheduled to be executed. |
| DEFTRAN | Records all deferred transactions. |
| DEFTRANDEST | Lists the destinations for a deferred transaction. |

# Creating a Deferred Transaction

Every well formed deferred transaction must consist of zero or one DBMS_DEFER.TRANSACTION calls followed by zero or more well formed deferred remote procedure calls, followed by a SQL COMMIT statement.

**Attention:** The procedures for which you are building deferred calls must be part of a package. Deferred calls to standalone procedures are not supported.

Every well formed deferred remote procedure call must consist of one DBMS_DEFER.CALL call, followed by zero or more DBMS_DEFER.*datatype*_ARG calls. The number of calls to the appropriate *datatype*_ARG procedures is determined by the value of the ARG_COUNT parameter passed to the CALL procedure.

If you do not call DBMS_DEFER.TRANSACTION to indicate the start of a transaction, Oracle considers your first call to DBMS_DEFER.CALL to be the start of a new transaction.

## Security

To create your own deferred transactions, you must have the EXECUTE privilege on the DBMS_DEFER package. This package is owned by SYS. Because deferred transactions are executed in the *privilege domain of the replication propagator,* EXECUTE privileges on the DBMS_DEFER package should not be widely granted.

**Suggestion:** To control access to these procedures, you should create a cover package in the replication propagator's schema, and grant EXECUTE on this cover package.

## Specifying a Destination

In addition to building the calls that make up a deferred transaction, you must also specify the destination for this transaction. Transactions placed into the deferred transaction queue by the advanced replication facility are queued to all of the asynchronous locations (dblinks) for the replicated object, as listed in the DBA_REPPROP view. When you use the procedures in the DBMS_DEFER package to add a deferred transaction to the queue, you must specify a destination using one of the following methods. These methods are listed in order of precedence:

1. If you meet the following conditions, the DBA_REPPROP view is used to determine destinations for the deferred transaction:
   - You do not use the NODES parameter to specify a destination in the call to DBMS_DEFER.TRANSACTION.

- You do not use the NODES parameter to specify a destination for any calls to DBMS_DEFER.CALL.
- Every call corresponds to a procedure in a package generated for an object in DBA_REPOBJECT.

**Note:** This method cannot be combined with any of the following methods.

2. You specify one or more fully qualified database names as the NODES parameter to the DBMS_DEFER.CALL procedure. This value applies to the current deferred remote procedure call only.
3. You specify one or more fully qualified database names as the NODES parameter to the DBMS_DEFER.TRANSACTION procedure. This value applies to all deferred calls that make up the transaction.
4. If you do not use one of the previous mechanisms to specify a destination, Oracle uses the contents of the "DEFDEFAULTDEST View" on page 10-21 to determine the destination for the calls.

## Initiating a Deferred Transaction

Indicate the start of a new deferred transaction by calling the TRANSACTION procedure in the DBMS_DEFER package, as shown in the following example:

```
nodes dbms_defer.node_list_t;
node(1) := 'acct_hq.hq.com';
node(2) := 'acct_ny.ny.com';
DBMS_DEFER.TRANSACTION(nodes);
```

In this example, any calls that make up the deferred transaction for which you do not specify a destination when you call DBMS_DEFER.CALL, will be queued for the ACCT_HQ and ACCT_NY databases.

The call to TRANSACTION is optional. If you do not call TRANSACTION, Oracle considers your first call to DBMS_DEFER.CALL to be the start of a new transaction. Calling TRANSACTION is useful if you want to specify a list of nodes to which to forward the deferred calls, and the list is the same for all calls in the deferred transaction.

All deferred transactions are recorded in the DEFTRAN view. Each destination of the transaction is noted in the DEFTRANDEST view.

**Additional Information:** The parameters for the TRANSACTION procedure are described in Table 9–7 on page 9 - 9, and the exceptions are listed in Table 9–8 on page 9 - 9.

## Deferring a Remote Procedure Call

To build a deferred call to a remote procedure, call the CALL procedure in the DBMS_DEFER package, as shown in the following example:

```
DBMS_DEFER.CALL(
     schema_name       =>   'accts_rec',
     package_name      =>   'hr',
     proc_name         =>   'hire_emp',
     arg_count         =>   3);
```

This example builds a deferred call to the HR.HIRE_EMP procedure in the ACCTS_REC schema. This HIRE_EMP procedure takes three arguments. No destination is specified for the deferred call, so the destination must have been specified using one of the other methods outlined on page 8 - 3.

All deferred remote procedure calls are recorded in the DEFCALL view. Each destination for the call is noted in the DEFCALLDEST view.

**Additional Information:** The parameters for the CALL procedure are described in Table 9–1 on page 9 - 6, and the exceptions are listed in Table 9–2 on page 9 - 6.

## Queuing a Parameter Value for a Deferred Call

After deferring a call to a remote procedure, you must provide the data that is passed to this procedure (only IN parameters are supported). There must be one call for each of the arguments that is passed to the remote procedure, and these calls must be made in the order that the arguments must be passed. The type of the data determines which procedure in the DBMS_DEFER package you must call. For example, suppose you deferred a call to the HIRE_EMP procedure, and it took three arguments, as shown below:

```
HIRE_EMP(ename IN VARCHAR2, empno IN NUMBER, salary IN NUMBER)
```

After building the deferred call to HIRE_EMP, you could pass the necessary data to this procedure by making the following three calls:

```
DBMS_DEFER.VARCHAR2_ARG('scott');
DBMS_DEFER.NUMBER_ARG(12345);
DBMS_DEFER.NUMBER_ARG(30000);
```

Depending upon the type of the data that you need to pass to the procedure, you need to call one of the following procedures in the DBMS_DEFER package for each argument to the procedure:

```
DBMS_DEFER.NUMBER_ARG(arg IN NUMBER);
```

```
DBMS_DEFER.DATE_ARG(arg IN DATE);
DBMS_DEFER.VARCHAR2_ARG(arg IN VARCHAR2);
DBMS_DEFER.NVARCHAR2_ARG(arg IN NVARCHAR2);
DBMS_DEFER.CHAR_ARG(arg IN CHAR);
DBMS_DEFER.NCHAR_ARG(arg IN NCHAR);
DBMS_DEFER.ROWID_ARG(arg IN ROWID);
DBMS_DEFER.RAW_ARG(arg IN RAW);
DBMS_DEFER.BLOB_ARG(arg IN BLOB);
DBMS_DEFER.CLOB_ARG(arg IN CLOB);
DBMS_DEFER.NCLOB_ARG(arg IN NCLOB);
```

**Note:** The RAW_ARG, CHAR_ARG, NCHAR_ARG, VARCHAR2_ARG, and NVARCHAR2_ARG procedures can raise an ORA-23323 exception if the argument that you pass to the procedure is too long.

## Adding a Destination to the DEFDEFAULTDEST View

If you use the DBMS_DEFER package to build a deferred transaction, and you do not supply a destination for a deferred transaction or the calls within that transaction, Oracle uses the DEFDEFAULTDEST view to determine the destination databases to which you want to defer a remote procedure call.

To add a destination database to this view call the ADD_DEFAULT_DEST procedure in the DBMS_DEFER_SYS package as shown in the following example:

```
DBMS_DEFER_SYS.ADD_DEFAULT_DEST( dblink => 'acct_ny.ny.com');
```

In this example, any *future* deferred transactions for which no destination has been specified will be queued for the ACCT_NY database.

**Additional Information:** The parameter for the ADD_DEFAULT_DEST procedure is described in Table 9–18 on page 9 - 19, and the exception is listed in Table 9–19 on page 9 - 19.

## Removing a Destination from the DEFDEFAULTDEST View

To remove a destination database from the DEFDEFAULTDEST view, call the DELETE_DEFAULT_DEST procedure in the DBMS_DEFER_SYS package, as shown in the following example:

```
DBMS_DEFER_SYS.DELETE_DEFAULT_DEST( dblink => 'acct_ny.ny.com');
```

In this example, any future deferred transactions that you create will no longer be queued for the ACCT_NY database as the default.

To delete a transaction from the deferred transaction queue, you can use Replication Manager. For more information, see “Purging a Site’s Deferred Transaction Queue” on page 3-12.

**Additional Information:** The parameter for the DELETE_DEFAULT_DEST procedure is described in Table 9–20 on page 9 - 20.

### Executing a Deferred Transaction

When you build a deferred transaction, the transaction is added to the deferred transaction queue at your local site. The remote procedures are not executed until this queue is pushed. You can either schedule this queue to be pushed at a periodic interval by creating a scheduled link or by calling DBMS_DEFER_SYS.SCHEDULE_PUSH, or you can force the queue to be pushed immediately with Replication Manager or by calling DBMS_DEFER_SYS.PUSH. These transactions are propagated in the same manner as your DML changes are propagated by the advanced replication facility.

## LOB Storage

Oracle supports large internal objects (LOBs): binary LOBs (BLOBs); character LOBs (CLOBs); and national character LOBs (NCLOBs).

**Note:** For data manipulation language (DML) or for piecewise update, the larger the size of the LOB (update), the more the propagation time will increase.

**Note:** All sites must be Oracle8 sites to support deferred RPCs of above named objects.

**Attention:** For security, note that a LOB parameter to a (deferred) RPC is visible in the transaction only while the RPC is being executed.

### DEFLOB View of Storage for RPC

Oracle stores internal LOB parameters to deferred RPCs in a side table that is referenced only by way of a synonym. This gives the you flexibility for storage parameters and the containing schema. The following shows the default storage table for LOB parameters.

```
CREATE TABLE system.def$_lob(
 id RAW(16) CONSTRAINT def$_lob_primary PRIMARY KEY,
 deferred_tran_db VARCHAR2(128), -- origin db
 deferred_tran_id VARCHAR2(22), -- transaction id
 blob_col BLOB,
 clob_col CLOB
```

```
 nclob_col NCLOB)
/
-- make deletes fast
CREATE INDEX system.def$_lob_n1 ON system.def$_lob(
 deferred_tran_db,
 deferred_tran_id)
/
-- use a synonym in case underlying table is moved
CREATE SYNONYM sys.def$_lob FOR system.def$_lob
/
CREATE OR REPLACE VIEW DefLOB AS SELECT * FROM sys.def$_lob
/
CREATE PUBLIC SYNONYM DefLOB FOR DefLOB
/
```

# 9

# Replication Management API Reference

All installations of Oracle advanced replication include the replication management application programming interface (API). A server's *replication management API* is a set of PL/SQL packages that encapsulates procedures and functions that administrators can use to configure Oracle's advanced replication features. Oracle Replication Manager also uses the procedures and functions of each site's replication management API to perform work. This chapter describes that packages that constitute Oracle replication API, including:

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

## Packages

Oracle's replication management API includes the following packages:

- DBMS_DEFER
- DBMS_DEFER_QUERY
- DBMS_DEFER_SYS
- DBMS_OFFLINE_OG
- DBMS_OFFLINE_SNAPSHOT
- DBMS_RECTIFIER_DIFF
- DBMS_REFRESH
- DBMS_REPCAT
- DBMS_REPCAT_ADMIN
- DBMS_REPCAT_AUTH
- DBMS_REPUTIL
- DBMS_SNAPSHOT

## Examples of Using Oracle's Replication Management API

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

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

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

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

## Prerequisites to Consider

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

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

## Replication Manager and Oracle Replication Management API

Oracle's Replication Manager uses the replication management API to perform most of its functions. Using Replication Manager is much more convenient than issuing replication management API calls individually because the utility:

- Provides a GUI interface to type in and adjust API call parameters.
- Automatically orders numerous, related API calls in the proper sequence.
- Displays output returned from API calls in message boxes and error files.

An easy way to learn how to use Oracle's replication management API is to use Replication Manager scripting feature. When you start an administrative session with Replication Manager, turn scripting on. When you are finished, turn scripting off and then review the script file. The script file contains all replication management API calls that were made during the session. See the Replication Manager help documentation for more information about its scripting feature.

# DBMS_DEFER Package

The DBMS_DEFER package contains the following procedures:

- DBMS_DEFER.CALL
- DBMS_DEFER.COMMIT_WORK
- DBMS_DEFER.datatype_ARG
- DBMS_DEFER.TRANSACTION

The following pages discuss each procedure.

# DBMS_DEFER.CALL

## Purpose

To build a deferred call to a remote procedure.

## Syntax

The parameters for the CALL procedure are described in Table 9–1, and the exceptions are listed in Table 9–2. The syntax for this procedure is shown below:

```
DBMS_DEFER.CALL(
      schema_name       IN   VARCHAR2,
      package_name      IN   VARCHAR2,
      proc_name         IN   VARCHAR2,
      arg_count         IN   NATURAL,
      { nodes           IN   node_list_t
      | group_name      IN   VARCHAR2 :=''})
```

**Note:** The CALL procedure is overloaded. The NODES and GROUP_NAME parameters are mutually exclusive.

***Table 9–1 Parameters for CALL***

| Parameter | Description |
|---|---|
| schema_name | The name of the schema in which the stored procedure is located. |
| package_name | The name of the package containing the stored procedure. The stored procedure must be part of a package. Deferred calls to standalone procedures are not supported. |
| proc_name | The name of the remote procedure to which you want to defer a call. |
| arg_count | The number of parameters for the procedure. You must have one call to DBMS_DEFER.*datatype*_ARG for each of these parameters. |
| nodes | A PL/SQL table of fully qualified database names to which you want to propagate the deferred call. The table is indexed starting at position 1 and ending when a NULL entry is found, or the NO_DATA_FOUND exception is raised. The data in the table is case insensitive. This argument is optional. |
| group_name | Reserved for internal use. |

***Table 9–2 Exceptions for CALL***

| Exception | Description |
|---|---|
| ORA-23304 (malformedcall) | The previous call was not correctly formed. |
| ORA-23319 | Parameter value is not appropriate. |
| ORA-23352 | The destination list (specified by NODES or by a previous DBMS_DEFER.TRANSACTION call) contains duplicates. |

# DBMS_DEFER.COMMIT_WORK

## Purpose

To perform a transaction commit after checking for well-formed deferred remote procedure calls.

## Syntax

The parameter for the COMMIT_WORK procedure is described in Table 9–3, and the exception is listed in Table 9–4. The syntax for this procedure is shown below:

```
DBMS_DEFER.COMMIT_WORK(commit_work_comment IN VARCHAR2)
```

***Table 9–3 Parameter for COMMIT_WORK***

| Parameter | Description |
|---|---|
| commit_work_comment | Up to 50 bytes to describe the transaction in the DEF$_CALL table. |

***Table 9–4 Exception for COMMIT_WORK***

| Exception | Description |
|---|---|
| ORA-23304 (malformedcall) | The transaction was not correctly formed or terminated. |

# DBMS_DEFER.*datatype*_ARG

## Purpose

To provide the data that is to be passed to a deferred remote procedure call. Depending upon the type of the data that you need to pass to a procedure, you need to call one of the following procedures in the DBMS_DEFER package for each argument to the procedure:

```
DBMS_DEFER.NUMBER_ARG       (arg  IN NUMBER)
DBMS_DEFER.DATE_ARG         (arg  IN DATE)
DBMS_DEFER.VARCHAR2_ARG     (arg  IN VARCHAR2)
DBMS_DEFER.CHAR_ARG         (arg  IN CHAR)
DBMS_DEFER.ROWID_ARG        (arg  IN ROWID)
DBMS_DEFER.RAW_ARG          (arg  IN RAW)
DBMS_DEFER.BLOB_ARG         (arg  IN BLOB)
DBMS_DEFER.CLOB_ARG         (arg  IN CLOB)
DBMS_DEFER.NCLOB_ARG        (arg  IN NCLOB)
DBMS_DEFER.NCHAR_ARG        (arg  IN NCHAR)
DBMS_DEFER.NVARCHAR2_ARG    (arg  IN NVARCHAR2)
DBMS_DEFER.ANY_CLOB_ARG     (arg  IN CLOB)
DBMS_DEFER.ANY_VARCHAR2_ARG (arg  IN VARCHAR2)
DBMS_DEFER.ANY_CHAR_ARG     (arg  IN CHAR)
DBMS_DEFER.BFILE_ARG        (arg  IN BFILE)
DBMS_DEFER.CFILE_ARG        (arg  IN CFILE)
```

***Table 9–5 Parameter for datatype_ARG***

| Parameter | Description |
|---|---|
| arg | The value of the parameter that you want to pass to the remote procedure to which you previously deferred a call. |

***Table 9–6 Exception for datatype_ARG***

| Exception | Description |
|---|---|
| ORA-23323 | The argument value is too long. |

# DBMS_DEFER.TRANSACTION

## Purpose

To indicate the start of a new deferred transaction. If you omit this call, Oracle considers your first call to DBMS_DEFER.CALL to be the start of a new transaction.

## Syntax

The parameter for the TRANSACTION procedure is described in Table 9–7, and the exceptions are listed in Table 9–8. The syntax for this procedure is as follows:

```
DBMS_DEFER.TRANSACTION
DBMS_DEFER.TRANSACTION(nodes  IN   node_list_t)
```

**Note:** The TRANSACTION procedure is overloaded. The behavior of the version without an input parameter is similar to that of the version with an input parameter, except that the former uses the NODES in the DEFDEFAULTDEST view instead of using the nodes in the nodes parameter.

***Table 9–7 Parameter for TRANSACTION***

| Parameter | Description |
|---|---|
| nodes | A PL/SQL table of fully qualified database names to which you want to propagate the deferred calls of the transaction. The table is indexed starting at position 1 until a NULL entry is found, or the NO_DATA_FOUND exception is raised. The data in the table is case insensitive. |

***Table 9–8 Exceptions for TRANSACTION***

| Exception | Description |
|---|---|
| ORA-23304 (malformedcall) | The previous transaction was not correctly formed or terminated. |
| ORA-23319 | Parameter value is not appropriate. |
| ORA-23352 | Raised by DBMS_DEFER.CALL if the node list contains duplicates. |

# DBMS_DEFER_QUERY Package

The DBMS_DEFER_QUERY package contains the following procedures and functions:

- DBMS_DEFER_QUERY.GET_ARG_FORM
- DBMS_DEFER_QUERY.GET_ARG_TYPE
- DBMS_DEFER_QUERY.GET_CALL_ARGS
- DBMS_DEFER_QUERY.GET_datatype_ARG

The following pages discuss each procedure and function.

# DBMS_DEFER_QUERY.GET_ARG_FORM

## Purpose

To determine the form of an argument in a deferred call. For more about displaying deferred transactions, see “Displaying Deferred Transactions” on page 6-9. For more information about displaying error transactions, see “Displaying Error Transactions” on page 6-11.

## Syntax

The parameters for the GET_ARG_FORM function are described in Table 9–9, the exception is listed in Table 9–10. The syntax for this procedure is shown below:

```
DBMS_DEFER_QUERY.GET_ARG_FORM(
      callno                IN   NUMBER,
      arg_no                IN   NUMBER,
      deferred_tran_id      IN   VARCHAR2 )
  RETURN NUMBER
```

***Table 9–9 Parameters for GET_ARG_FORM***

| Parameter | Description |
|---|---|
| callno | The call identifier from the DEFCALL view. |
| arg_no | The position of desired parameter in calls argument list. Parameter positions are 1..*number* of parameters in call. |
| deferred_tran_id | The deferred transaction id. |

***Table 9–10 Exception for GET_ARG_FORM***

| Exception | Description |
|---|---|
| NO_DATA_FOUND | The input parameters do not correspond to a parameter of a deferred call. |

# DBMS_DEFER_QUERY.GET_ARG_TYPE

## Purpose

To determine the type of an argument in a deferred call. For more about displaying deferred transactions, see “Displaying Deferred Transactions” on page 6-9. For more information about displaying error transactions, see “Displaying Error Transactions” on page 6-11.

## Syntax

The parameters for the GET_ARG_TYPE function are described in Table 9–11, the exception is listed in Table 9–12, and the possible return values are described in Table 9–13. The syntax for this procedure is shown below:

```
DBMS_DEFER_QUERY.GET_ARG_TYPE(
      callno            IN   NUMBER,
      arg_no            IN   NUMBER,
      deferred_tran_id  IN   VARCHAR2)
  RETURN NUMBER
```

***Table 9–11 Parameters for GET_ARG_TYPE***

| Parameter | Description |
|---|---|
| callno | The ID number from the DEFCALL view of the deferred remote procedure call. |
| arg_no | The numerical position of the argument to the call whose type you want to determine. The first argument to a procedure is in position 1. |
| deferred_tran_id | The identifier of the deferred transaction. |

***Table 9–12 Exception for GET_ARG_TYPE***

| Exception | Description |
|---|---|
| NO_DATA_FOUND | The input parameters do not correspond to a parameter of a deferred call. |

***Table 9–13 Return Values for GET_ARG_TYPE***

| Return Value | Corresponding Datatype |
|---|---|
| 1 | VARCHAR2 |
| 2 | NUMBER |
| 11 | ROWID |
| 12 | DATE |
| 23 | RAW |
| 96 | CHAR |

# DBMS_DEFER_QUERY.GET_CALL_ARGS

## Purpose

This procedure returns the text version of the various arguments for the given call.

## Syntax

The parameters for the GET_CALL_ARGS procedure are described in Table 9–14 and the exception is listed in Table 9–15. The syntax for this procedure is as follows:

```
DBMS_DEFER_QUERY.GET_CALL_ARGS (
      callno    IN  NUMBER,
      startarg  IN  NUMBER := 1,
      argcnt    IN  NUMBER,
      argsize   IN  NUMBER,
      tran_id   IN  VARCHAR2,
      date_fmt  IN  VARCHAR2,
      types     OUT TYPE_ARY,
      forms     OUT TYPE_ARY,
      vals      OUT VAL_ARY)
```

***Table 9–14 Parameters for GET_CALL_ARGS***

| Parameter | Description |
|---|---|
| callno | The ID number from the DEFCALL view of the deferred RPC. |
| startarg | The numerical position of the first argument you want described. |
| argcnt | The number of arguments in the call. |
| argsize | The maximum size of returned argument. |
| tran_id | Identifier of the deferred transaction. |
| date_fmt | The format in which the date should be returned. |
| types | An array containing the types of arguments. |
| forms | An array containing the character set forms of arguments. |
| vals | An array containing the values of the arguments in a textual form. |

***Table 9–15 Exception for GET_CALL_ARGS***

| Exception | Description |
|---|---|
| NO_DATA_FOUND | The input parameters do not correspond to a parameter of a deferred call. |

# DBMS_DEFER_QUERY.GET_*datatype*_ARG

## Purpose

To determine the value of an argument in a deferred call. For more about displaying deferred transactions, see "Displaying Deferred Transactions" on page 6-9. For more information about displaying error transactions, see "Displaying Error Transactions" on page 6-11.

## Syntax

Depending upon the type of the argument value that you want to retrieve, the syntax for the appropriate function is as follows. The parameters for these functions are described in Table 9–16, and the exceptions are listed in Table 9–17. Each of these functions returns the value of the specified argument.

```
DBMS_DEFER_QUERY.GET_datatype_ARG (
      callno             IN   NUMBER,
      arg_no             IN   NUMBER,
      deferred_tran_id   IN   VARCHAR2 DEFAULT NULL)
  RETURN datatype
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| ROWID
| BLOB
| CLOB
| NCLOB
| NCHAR
| NVARCHAR2 }
```

*Table 9–16 Parameters for GET_datatype_ARG*

| Parameter | Description |
|---|---|
| callno | The ID number from the DEFCALL view of the deferred remote procedure call. |
| arg_no | The numerical position of the argument to the call whose value you want to determine. The first argument to a procedure is in position one. |
| deferred_tran_id | Default NULL. The identifier of the deferred transaction. Defaults to the last transaction identifier passed to GET_ARG_TYPE. |

*Table 9–17 Exceptions for GET_datatype_ARG*

| Exception | Description |
|---|---|
| NO_DATA_FOUND | The input parameters do not correspond to a parameter of a deferred call. |
| ORA-26564 | The argument in this position is not of the specified type. |

# DBMS_DEFER_SYS Package

The DBMS_DEFER_SYS package contains the following procedures and functions:

- DBMS_DEFER_SYS.ADD_DEFAULT_DEST
- DBMS_DEFER_SYS.DELETE_DEFAULT_DEST
- DBMS_DEFER_SYS.DELETE_DEF_DESTINATION
- DBMS_DEFER_SYS.DELETE_ERROR
- DBMS_DEFER_SYS.DELETE_TRAN
- DBMS_DEFER_SYS.DISABLED
- DBMS_DEFER_SYS.EXCLUDE_PUSH
- DBMS_DEFER_SYS.EXECUTE_ERROR
- DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER
- DBMS_DEFER_SYS.PURGE
- DBMS_DEFER_SYS.PUSH
- DBMS_DEFER_SYS.REGISTER_PROPAGATOR
- DBMS_DEFER_SYS.SCHEDULE_PURGE
- DBMS_DEFER_SYS.SCHEDULE_PUSH
- DBMS_DEFER_SYS.SET_DISABLED
- DBMS_DEFER_SYS.UNREGISTER_PROPAGATOR
- DBMS_DEFER_SYS.UNSCHEDULE_PURGE
- DBMS_DEFER_SYS.UNSCHEDULE_PUSH

The following pages discuss each procedure and function.

# DBMS_DEFER_SYS.ADD_DEFAULT_DEST

## Purpose

To add a destination database to the DEFDEFAULTDEST view.

## Syntax

The parameter for the ADD_DEFAULT_DEST procedure is described in Table 9–18, and the exception is listed in Table 9–19. The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.ADD_DEFAULT_DEST(dblink   IN   VARCHAR2)
```

***Table 9–18 Parameter for ADD_DEFAULT_DEST***

| Parameter | Description |
|---|---|
| dblink | The fully qualified database name of the node that you want to add to the DEFDEFAULTDEST view. |

***Table 9–19 Exception for ADD_DEFAULT_DEST***

| Exception | Description |
|---|---|
| ORA-23352 | The DBLINK that you specified is already in the default list. |

# DBMS_DEFER_SYS.DELETE_DEFAULT_DEST

## Purpose

To remove a destination database from the DEFDEFAULTDEST view.

## Syntax

The parameter for the DELETE_DEFAULT_DEST procedure is described in Table 9–20. The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.DELETE_DEFAULT_DEST(dblink   IN   VARCHAR2)
```

***Table 9–20 Parameter for DELETE_DEFAULT_DEST***

| Parameter | Description |
|---|---|
| dblink | The fully qualified database name of the node that you want to delete from the DEFDEFAULTDEST view. If Oracle does not find this dblink in the view, no action is taken. |

# DBMS_DEFER_SYS.DELETE_DEF_DESTINATION

## Purpose

To remove a destination database from the DEFSCHEDULE view.

## Syntax

The parameters for the DELETE_DEF_DESTINATION procedure is described in Table 9–21. The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.DELETE_DEF_DESTINATION(
      destination   IN   VARCHAR2,
      force         IN   BOOLEAN := FALSE)
```

***Table 9–21 Parameters for DELETE_DEF_DESTINATION***

| Parameter | Description |
|---|---|
| destination | The fully qualified database name of the destination that you want to delete from the DefSchedule view. If Oracle does not find this destination in the view, no action is taken. |
| force | When set to TRUE, Oracle ignores all safety checks and deletes the destination. |

# DBMS_DEFER_SYS.DELETE_ERROR

## Purpose

To delete a transaction from the DEFERROR view.

## Syntax

The parameters for the DELETE_ERROR procedure are described in Table 9–22. The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.DELETE_ERROR(
      deferred_tran_id     IN   VARCHAR2,
      destination          IN   VARCHAR2)
```

***Table 9–22 Parameters for DELETE_ERROR***

| Parameter | Description |
|---|---|
| deferred_tran_id | The ID number from the DEFERROR view of the deferred transaction that you want to remove from the DEFERROR view. If this parameter is null, all transactions meeting the requirements of the other parameters are removed. |
| destination | The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. If this parameter is null, all transactions meeting the requirements of the other parameters are removed from the DEFERROR view. |

# DBMS_DEFER_SYS.DELETE_TRAN

## Purpose

To delete a transaction from the DEFTRANDEST view. If there are no other DEFTRANDEST or DEFERROR entries for the transaction, the transaction is deleted from the DEFTRAN and DEFCALL views as well.

## Syntax

The parameters for the DELETE_TRAN procedure are described in Table 9–23. The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.DELETE_TRAN(
      deferred_tran_id     IN   VARCHAR2,
      destination          IN   VARCHAR2)
```

***Table 9–23 Parameters for DELETE_TRAN***

| Parameter | Description |
|---|---|
| deferred_tran_id | The ID number from the DEFTRAN view of the deferred transaction that you want to delete. If this parameter is null, all transactions meeting the requirements of the other parameters are deleted. |
| destination | The fully qualified database name from the DEFTRANDEST view of the database to which the transaction was originally queued. If this parameter is null, all transactions meeting the requirements of the other parameters are deleted. |

# DBMS_DEFER_SYS.DISABLED

## Purpose

To determine whether propagation of the deferred transaction queue from the current site to a given site is enabled. The DISABLED function returns TRUE if the deferred remote procedure call (RPC) queue is disabled for the given destination.

## Syntax

The parameter for the DISABLED function is described in Table 9–24, the return values are described in Table 9–25, and the exception is described in Table 9–26. The syntax for this function is shown below:

```
DBMS_DEFER_SYS.DISABLED(
      destination  IN   VARCHAR2)
  RETURN BOOLEAN
```

***Table 9–24 Parameter for DISABLED***

| Parameter | Description |
|---|---|
| destination | The fully qualified database name of the node whose propagation status you want to check. |

***Table 9–25 Return Values for DISABLED***

| Value | Description |
|---|---|
| TRUE | Propagation to this site from the current site is disabled. |
| FALSE | Propagation to this site from the current site is enabled. |

***Table 9–26 Exception for DISABLED***

| Exception | Description |
|---|---|
| NO_DATA_FOUND | DESTINATION does not appear in the DEFSCHEDULE view. |

# DBMS_DEFER_SYS.EXCLUDE_PUSH

## Purpose

To acquire an exclusive lock that prevents deferred transaction PUSH (either serial or parallel). This function does a commit. The lock is acquired with RELEASE_ON_COMMIT => TRUE, so that pushing of the deferred transaction queue can resume after the next commit.

## Syntax

The parameters and return values are shown below.

```
DBMS_DEFER_SYS.EXCLUDE_PUSH(
      timeout   IN   INTEGER)
  RETURN INTEGER
```

***Table 9–27 Parameter for EXCLUDE_PUSH***

| Parameter | Description |
|---|---|
| timeout | Timeout in seconds. If the lock cannot be acquired within this time period (either because of an error or because a PUSH is currently under way), the call returns a value of 1. A timeout value of DBMS_LOCK.MAXWAIT waits indefinitely. |

***Table 9–28 Return Values for EXCLUDE_PUSH***

| Value | Description |
|---|---|
| 0 | Success, lock acquired. |
| 1 | Timeout, no lock acquired. |
| 2 | Deadlock, no lock acquired. |
| 4 | Already own lock. |

# DBMS_DEFER_SYS.EXECUTE_ERROR

## Purpose

To reexecute a deferred transaction that did not initially complete successfully. This procedure raises an ORA-24275 error when illegal combinations of NULL and non-NULL parameters are used.

## Syntax

The parameters for the EXECUTE_ERROR procedure are described in Table 9–29. The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.EXECUTE_ERROR(
      deferred_tran_id IN   VARCHAR2,
      destination      IN   VARCHAR2)
```

***Table 9–29 Parameters for EXECUTE_ERROR***

| Parameter | Description |
|---|---|
| deferred_tran_id | The ID number from the DEFERROR view of the deferred transaction that you want to re-execute. If this parameter is null, all transactions queued for DESTINATION are re-executed. |
| destination | The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. This parameter must not be null. |

***Table 9–30 Exceptions for EXECUTE_ERROR***

| Exception | Description |
|---|---|
| badparam | Parameter value missing or invalid (for example, if destination is NULL). |
| missinguser | Invalid user. |

# DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER

## Purpose

To reexecute a deferred transaction that did not initially complete successfully. Each transaction is executed in the security context of the connected user. This procedure raises an ORA-24275 error when illegal combinations of NULL and non-NULL parameters are used.

## Syntax

The parameters for the EXECUTE_ERROR_AS_USER procedure are described in Table 9–31. The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER(
      deferred_tran_id IN   VARCHAR2,
      destination      IN   VARCHAR2)
```

***Table 9–31 Parameters for EXECUTE_ERROR_AS_USER***

| Parameter | Description |
|---|---|
| deferred_tran_id | The ID number from the DEFERROR view of the deferred transaction that you want to re-execute. If this parameter is null, all transactions queued for DESTINATION that originated from the DEFERRED_TRAN_DB are re-executed. |
| destination | The fully qualified database name from the DEFERROR view of the database to which the transaction was originally queued. This parameter must not be null. |

***Table 9–32 Exceptions for EXECUTE_ERROR_AS_USER***

| Exception | Description |
|---|---|
| badparam | Parameter value missing or invalid (for example, if destination is NULL). |
| missinguser | Invalid user. |

# DBMS_DEFER_SYS.PURGE

## Purpose

To purge pushed transactions from the deferred transaction queue at your current master or snapshot site.

## Syntax

The parameters for the PURGE function are shown in Table 9–33. The syntax for this function is shown below:

```
DBMS_DEFER_SYS.PURGE(
 purge_method          IN  BINARY_INTEGER := purge_method_quick,
 rollback_segment      IN  VARCHAR2       := NULL,
 startup_seconds       IN  BINARY_INTEGER := 0,
 execution_seconds     IN  BINARY_INTEGER := seconds_infinity,
 delay_seconds         IN  BINARY_INTEGER := 0,
 transaction_count     IN  BINARY_INTEGER := transactions_infinity,
 write_trace           IN  BOOLEAN        := NULL)
 RETURN BINARY_INTEGER
```

***Table 9–33 Parameters for PURGE***

| Parameter | Description |
|---|---|
| purge_method | Controls how to purge the deferred transaction queue; purge_method_quick cost less, while purge_method_precise offers better precision. |
| rollback_segment | Name of rollback segment to use for the purge, or NULL for default. |
| startup_seconds | The maximum number of seconds to wait for a previous purge of the same deferred transaction queue. |
| execution_seconds | If >0, stop push cleanly after the specified number of seconds of real time. |
| delay_seconds | Stop execution cleanly after the deferred transaction queue is empty for delay_seconds. |
| transaction_count | If > 0, shutdown cleanly after purging transaction_count number of transactions. |
| write_trace | When set to TRUE, Oracle records the result value returned by the function in the server's trace file. |

***Table 9–34 Return Values for Purge***

| Value | Description |
|---|---|
| 0 | OK, terminated after delay_seconds expired. |
| 1 | Terminated by lock timeout while starting. |
| 2 | Terminated by exceeding execution_seconds. |
| 3 | Terminated by exceeding transaction_count. |
| 4 | Terminated by exceeding delivery_order_limit. |
| 5 | Terminated after errors. |

*Table 9–35 Exceptions for PURGE*

| Exception | Description |
|---|---|
| argoutofrange | A parameter value is out of a valid range. |
| executiondisabled | The execution of deferred RPCs is disabled at the destination. |
| defererror | Internal error. |

# DBMS_DEFER_SYS.PUSH

## Purpose

To force a deferred remote procedure call queue at your current master or snapshot site to be pushed (executed, propagated) to another master site using either serial or parallel propagation.

## Syntax

The parameters for the PUSH function are shown in Table 9–36. The syntax for this function is shown below:

```
DBMS_DEFER_SYS.PUSH(
 destination          IN  VARCHAR2,
 parallelism          IN  BINARY_INTEGER := 0,
 heap_size            IN  BINARY_INTEGER := 0)
 stop_on_error        IN  BOOLEAN        := FALSE,
 write_trace          IN  BOOLEAN        := FALSE,
 startup_seconds      IN  BINARY_INTEGER := 0,
 execution_seconds    IN  BINARY_INTEGER := seconds_infinity,
 delay_seconds        IN  BINARY_INTEGER := 0,
 transaction_count    IN  BINARY_INTEGER := transactions_infinity,
 delivery_order_limit IN  NUMBER         := delivery_order_infinity)
 RETURN BINARY_INTEGER
```

***Table 9–36 Parameters for PUSH***

| Parameter | Description |
|---|---|
| destination | The fully qualified database name of the master to which you are forwarding changes. |
| parallelism | 0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process. |
| heap_size | Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set the parameter unless so directed by Oracle Worldwide Support. |

*Table 9–36 Parameters for PUSH*

| Parameter | Description |
|---|---|
| stop_on_error | The default, FALSE, indicates that the executor should continue even if errors, such as conflicts, are encountered. If TRUE, shutdown (cleanly if possible) at the first indication that a transaction encountered an error at the destination site. |
| write_trace | When set to TRUE, Oracle records the result value returned by the function in the server's trace file. |
| startup_seconds | The maximum number of seconds to wait for a previous push to the same destination. |
| execution_seconds | If >0, stop push cleanly after the specified number of seconds of real time. If transaction_count and execution_seconds are zero (the default), transactions are executed until there are no more in the queue. |
| delay_seconds | Do not return before the specified number of seconds have elapsed, even if the queue is empty. Useful for reducing execution overhead if DBMS_DEFER_SYS.PUSH is called from a tight loop. |
| transaction_count | If > 0, the maximum number of transactions to be pushed before stopping. If transaction_count and execution_seconds are zero (the default), transactions are executed until there are no more in the queue. |
| delivery_order_limit | Stop execution cleanly before pushing a transaction where delivery_order >= delivery_order_limit |

*Table 9–37 Return Values for PUSH*

| Value | Description |
|---|---|
| 0 | OK, terminated after delay_seconds expired. |
| 1 | Terminated by lock timeout while starting. |
| 2 | Terminated by exceeding execution_seconds. |
| 3 | Terminated by exceeding transaction_count. |
| 4 | Terminated by exceeding delivery_order_limit. |
| 5 | Terminated after errors. |

***Table 9–38 Exceptions for PUSH***

| Exception | Description |
|---|---|
| deferror incompleteparallelpush | Internal error. |
| executiondisabled | The execution of deferred RPCs is disabled at the destination. |
| cat_err_err | Error while creating entry in DEFERROR. |
| deferred_rpc_qiesce | Replication activity for object group is suspended. |
| commfailure | Communication failure during deferred RPC. |
| missingpropator | A propagator does not exist. |

# DBMS_DEFER_SYS.REGISTER_PROPAGATOR

## Purpose

Register the given user as the propagator for the local database. It also grants to the given user CREATE SESSION, CREATE PROCEDURE, CREATE DATABASE LINK, and EXECUTE ANY PROCEDURE privileges (so that the user can create wrappers).

## Syntax

The parameter for the REGISTER_PROPAGATOR procedure is described in Table 9–39. The syntax for this procedure is shown below, and the exceptions are listed in Table 9–40:

```
DBMS_DEFER_SYS.REGISTER_PROPAGATOR(username  IN  VARCHAR2)
```

***Table 9–39 Parameter for REGISTER_PROPAGATOR***

| Parameter | Description |
|---|---|
| username | The name of the user. |

### Exception

***Table 9–40 Exceptions for REGISTER_PROPAGATOR***

| Exception | Description |
|---|---|
| missinguser | The given user does not exist. |
| alreadypropagator | The given user is already the propagator. |
| duplicatepropagator | There is already a different propagator. |

# DBMS_DEFER_SYS.SCHEDULE_PURGE

## Purpose

To schedule a job to purge pushed transactions from the deferred transaction queue at your current master or snapshot site. You can schedule only one purge job per site.

## Syntax

The parameters for the SCHEDULE_PURGE procedure are shown in Table 9–41. The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.SCHEDULE_PURGE(
 interval             IN  VARCHAR2,
 next_date            IN  DATE,
 reset                IN  BOOLEAN        := NULL,
 purge_method         IN  BINARY_INTEGER := NULL,
 rollback_segment     IN  VARCHAR2       := NULL,
 startup_seconds      IN  BINARY_INTEGER := NULL,
 execution_seconds    IN  BINARY_INTEGER := NULL,
 delay_seconds        IN  BINARY_INTEGER := NULL,
 transaction_count    IN  BINARY_INTEGER := NULL,
 write_trace          IN  BOOLEAN        := NULL)
```

***Table 9–41 Parameters for SCHEDULE_PURGE***

| Parameter | Description |
|---|---|
| interval | Allows you to provide a function to calculate the next time to purge. This value is stored in the INTERVAL field of the DEFSCHEDULE view and calculates the NEXT_DATE field of this view. If you use the default value for this parameter, NULL, the value of this field remains unchanged. If the field had no previous value, it is created with a value of null. If you do not supply a value for this field, you must supply a value for NEXT_DATE. |

*Table 9–41 Parameters for SCHEDULE_PURGE*

| Parameter | Description |
|---|---|
| next_date | Allows you to specify a given time to purge pushed transactions from the site's queue. This value is stored in the NEXT_DATE field of the DEFSCHEDULE view. If you use the default value for this parameter, NULL, the value of this field remains unchanged. If this field had no previous value, it is created with a value of null. If you do not supply a value for this field, you must supply a value for INTERVAL. |
| reset | Set to TRUE to reset LAST_TXN_COUNT, LAST_ERROR, and LAST_MSG to NULL. |
| purge_method | Controls how to purge the deferred transaction queue; purge_method_quick cost less, while purge_method_precise offers better precision. |
| rollback_segment | Name of rollback segment to use for the purge, or NULL for default. |
| startup_seconds | The maximum number of seconds to wait for a previous purge of the same deferred transaction queue. |
| execution_seconds | If >0, stop execution cleanly after the specified number of seconds of real time. |
| delay_seconds | Stop execution cleanly after the deferred transaction queue is empty for delay_seconds. |
| transaction_count | If > 0, shutdown cleanly after purging transaction_count number of transactions. |
| write_trace | When set to TRUE, Oracle records the result value returned by the function in the server's trace file. |

# DBMS_DEFER_SYS.SCHEDULE_PUSH

## Purpose

To schedule a job to push the deferred transaction queue to a remote master destination. This procedure does a commit.

## Syntax

The parameters for the SCHEDULE_PUSH procedure are described in Table 9–42. The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.SCHEDULE_PUSH(
 destination          IN  VARCHAR2,
 interval             IN  VARCHAR2,
 next_date            IN  DATE,
 reset                IN  BOOLEAN        := FALSE,
 parallelism          IN  BINARY_INTEGER := NULL,
 heap_size            IN  BINARY_INTEGER := NULL,
 stop_on_error        IN  BOOLEAN        := NULL,
 write_trace          IN  BOOLEAN        := NULL,
 startup_seconds      IN  BINARY_INTEGER := NULL,
 execution_seconds    IN  BINARY_INTEGER := NULL,
 delay_seconds        IN  BINARY_INTEGER := NULL,
 transaction_count    IN  BINARY_INTEGER := NULL)
```

***Table 9–42 Parameters for SCHEDULE_PUSH***

| Parameter | Description |
|---|---|
| destination | The fully qualified database name of the master to which you are forwarding changes. |
| interval | Allows you to provide a function to calculate the next time to push. This value is stored in the INTERVAL field of the DEFSCHEDULE view and calculates the NEXT_DATE field of this view. If you use the default value for this parameter, NULL, the value of this field remains unchanged. If the field had no previous value, it is created with a value of null. If you do not supply a value for this field, you must supply a value for NEXT_DATE. |

***Table 9–42 Parameters for SCHEDULE_PUSH***

| Parameter | Description |
|---|---|
| next_date | Allows you to specify a given time to push deferred transactions to the master site destination. This value is stored in the NEXT_DATE field of the DEFSCHEDULE view. If you use the default value for this parameter, NULL, the value of this field remains unchanged. If this field had no previous value, it is created with a value of null. If you do not supply a value for this field, you must supply a value for INTERVAL. |
| reset | Set to TRUE to reset LAST_TXN_COUNT, LST_ERROr, and LAST_MSG to NULL. |
| parallelism | 0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process. |
| heap_size | Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set the parameter unless so directed by Oracle Worldwide Support. |
| stop_on_error | The default, FALSE, indicates that the executor should continue even if errors, such as conflicts, are encountered. If TRUE, shutdown (cleanly if possible) at the first indication that a transaction encountered an error at the destination site. |
| write_trace | When set to TRUE, Oracle records the result value returned by the function in the server's trace file. |
| startup_seconds | The maximum number of seconds to wait for a previous push to the same destination. |
| execution_seconds | If >0, stop execution cleanly after the specified number of seconds of real time. If transaction_count and execution_seconds are zero (the default), transactions are executed until there are no more in the queue. |
| delay_seconds | Do not return before the specified number of seconds have elapsed, even if the queue is empty. Useful for reducing execution overhead if DBMS_DEFER_SYS.PUSH is called from a tight loop. |

***Table 9–42 Parameters for SCHEDULE_PUSH***

| Parameter | Description |
|---|---|
| transaction_count | If > 0, the maximum number of transactions to be pushed before stopping. If transaction_count and execution_seconds are zero (the default), transactions are executed until there are no more in the queue. |

# DBMS_DEFER_SYS.SET_DISABLED

## Purpose

To disable or enable propagation of the deferred transaction queue from the current site to a given destination site. If the disabled parameter is TRUE, the procedure disables propagation to the given destination and future invocations of DBMS_DEFER_SYS.EXECUTE do not push the deferred remote procedure call (RPC) queue. SET_DISABLED affects neither a session already pushing the queue to the given destination nor sessions appending to the queue with DBMS_DEFER. If the disabled parameter is FALSE, the procedure enables propagation to the given destination and, although this does not push the queue, it permits future invocations to DBMS_DEFER_SYS.EXECUTE to push the queue to the given destination. Whether the disabled parameter is TRUE or FALSE, a COMMIT is required for the setting to take effect in other sessions.

## Syntax

The parameters for the SET_DISABLED procedure are described in Table 9–43 and the exception is listed in Table 9–44. The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.SET_DISABLED(
      destination   IN   VARCHAR2,
      disabled      IN   BOOLEAN := TRUE)
```

***Table 9–43 Parameters for SET_DISABLED***

| Parameter | Description |
|---|---|
| destination | The fully qualified database name of the node whose propagation status you want to change. |
| disabled | By default, this parameter disables propagation of the deferred transaction queue from your current site to the given destination. Set this parameter to FALSE to enable propagation. |

***Table 9–44 Exception for SET_DISABLED***

| Exception | Description |
|---|---|
| NO_DATA_FOUND | No entry was found in the DEFSCHEDULE view for the given DESTINATION. |

# DBMS_DEFER_SYS.UNREGISTER_PROPAGATOR

## Purpose

To unregister a user as the propagator from the local database. This procedure

- Deletes the given propagator from DEFPROPAGATOR.
- Revokes privileges granted by REGISTER_PROPAGATOR from the given user (including identical privileges granted independently).
- Drops any generated wrappers in the schema of the given propagator, and marks them as dropped in the replication catalog.

## Syntax

The parameters for the UNREGISTER_PROPAGATOR procedure are described in Table 9–45. The syntax for this procedure is shown below, and the exceptions are listed in Table 9–46:

```
DBMS_DEFER_SYS.UNREGISTER_PROPAGATOR(
      username  IN  VARCHAR2
      timeout   IN  INTEGER DEFAULT DBMS_LOCK.MAXWAIT)
```

***Table 9–45 Parameters for UNREGISTER_PROPAGATOR***

| Parameter | Description |
|---|---|
| username | The name of the propagator user. |
| timeout | Timeout in seconds. If the propagator is in use, the procedure waits until timeout. The default is DBMS_LOCK.MAXWAIT. |

### Exception

***Table 9–46 Exceptions for UNREGISTER_PROPAGATOR***

| Parameter | Description |
|---|---|
| missingpropagator | The given user is not a propagator. |
| propagator_inuse | The propagator is in use, and thus cannot be unregistered. Try later. |

# DBMS_DEFER_SYS.UNSCHEDULE_PURGE

## Purpose

To stop automatic purges of pushed transactions from the deferred transaction queue at a snapshot or master site.

## Syntax

The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.UNSCHEDULE_PURGE
```

# DBMS_DEFER_SYS.UNSCHEDULE_PUSH

## Purpose

To stop automatic pushes of the deferred transaction queue from a snapshot or master site to another master site.

## Syntax

The parameter for the UNSCHEDULE_PUSH procedure is described in Table 9–47, and the exception is described in Table 9–48. The syntax for this procedure is shown below:

```
DBMS_DEFER_SYS.UNSCHEDULE_PUSH(dblink    IN    VARCHAR2)
```

***Table 9–47 Parameter for UNSCHEDULE_PUSH***

| Parameter | Description |
|---|---|
| dblink | Fully qualified pathname to master database site at which you want to unschedule periodic execution of deferred remote procedure calls. |

***Table 9–48 Exception for UNSCHEDULE_PUSH***

| Exception | Description |
|---|---|
| NO_DATA_FOUND | No entry was found in the DEFSCHEDULE view for the given DBLINK. |

# DBMS_OFFLINE_OG Package

The DBMS_OFFLINE_OG package contains the following procedures:

- DBMS_OFFLINE_OG.BEGIN_INSTANTIATION
- DBMS_OFFLINE_OG.BEGIN_LOAD
- DBMS_OFFLINE_OG.END_INSTANTIATION
- DBMS_OFFLINE_OG.END_LOAD
- DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS

The following pages discuss each procedure.

# DBMS_OFFLINE_OG.BEGIN_INSTANTIATION

## Purpose

To start offline instantiation of a replicated object group. You must call this procedure from the master definition site.

## Syntax

The parameters for the BEGIN_INSTANTIATION procedure are described in Table 9–49, and the exceptions are listed in Table 9–50. The syntax for this procedure is shown below.

```
DBMS_OFFLINE_OG.BEGIN_INSTANTIATION(
      gname     IN   VARCHAR2,
      new_site  IN   VARCHAR2)
```

***Table 9–49 Parameters for BEGIN_INSTANTIATION***

| Parameter | Description |
|---|---|
| gname | The name of the object group that you want to replicate to the new site. |
| new_site | The fully qualified database name of the new site to which you want to replicate the object group. |

***Table 9–50 Exceptions for BEGIN_INSTANTIATION***

| Exception | Description |
|---|---|
| badargument | Null or empty string for object group or new master site name. |
| dbms_repcat.nonmasterdef | This procedure must be called from the master definition site. |
| sitealreadyexists | Given site is already a master site for this object group. |
| wrongstate | Status of master definition site must be QUIESCED. |
| dbms_repcat.missingrepgroup | GNAME does not exist as a replicated object group. |

# DBMS_OFFLINE_OG.BEGIN_LOAD

## Purpose

To disable triggers while data is imported to new master site as part of offline instantiation. You must call this procedure from the new master site. See "Snapshot Cloning and Offline Instantiation" on page 7-14.

## Syntax

The parameters for the BEGIN_LOAD procedure are described in Table 9–51, and the exceptions are listed in Table 9–52. The syntax for this procedure is shown below.

```
DBMS_OFFLINE_OG.BEGIN_LOAD(
      gname     IN   VARCHAR2,
      new_site  IN   VARCHAR2)
```

***Table 9–51 Parameters for BEGIN_LOAD***

| Parameter | Description |
|---|---|
| gname | The name of the object group whose members you are importing. |
| new_site | The fully qualified database name of the new site at which you will be importing the object group members. |

***Table 9–52 Exceptions for BEGIN_LOAD***

| Exception | Description |
|---|---|
| badargument | Null or empty string for object group or new master site name. |
| wrongsite | This procedure must be called from the new master site. |
| unknownsite | Given site is not recognized by object group. |
| wrongstate | Status of the new master site must be QUIESCED. |
| dbms_repcat.missingrep-group | GNAME does not exist as a replicated object group. |

# DBMS_OFFLINE_OG.END_INSTANTIATION

## Purpose

To complete offline instantiation of a replicated object group. You must call this procedure from the master definition site. See "Snapshot Cloning and Offline Instantiation" on page 7-14.

## Syntax

The parameters for the END_INSTANTIATION procedure are described in Table 9–53, and the exceptions are listed in Table 9–54. The syntax for this procedure is shown below.

```
DBMS_OFFLINE_OG.END_INSTANTIATION(
     gname     IN  VARCHAR2,
     new_site  IN  VARCHAR2)
```

***Table 9–53 Parameters for END_INSTANTIATION***

| Parameter | Description |
|---|---|
| gname | The name of the object group that you are replicating to the new site. |
| new_site | The fully qualified database name of the new site to which you are replicating the object group. |

***Table 9–54 Exceptions for END_INSTANTIATION***

| Exception | Description |
|---|---|
| badargument | Null or empty string for object group or new master site name. |
| dbms_repcat.nonmasterdef | This procedure must be called from the master definition site. |
| unknownsite | Given site is not recognized by object group. |
| wrongstate | Status of master definition site must be QUIESCED. |
| dbms_repcat.missingrepgroup | GNAME does not exist as a replicated object group. |

# DBMS_OFFLINE_OG.END_LOAD

## Purpose

To reenable triggers after importing data to new master site as part of offline instantiation. You must call this procedure from the new master site. For additional information, see “Snapshot Cloning and Offline Instantiation” on page 7-14.

## Syntax

The parameters for the END_LOAD procedure are described in Table 9–55, and the exceptions are listed in Table 9–56. The syntax for this procedure is shown below.

```
DBMS_OFFLINE_OG.END_LOAD(
    gname     IN   VARCHAR2,
    new_site  IN   VARCHAR2)
```

***Table 9–55 Parameters for END_LOAD***

| Parameter | Description |
|---|---|
| gname | The name of the object group whose members you have finished importing. |
| new_site | The fully qualified database name of the new site at which you have imported the object group members. |

***Table 9–56 Exceptions for END_LOAD***

| Exception | Description |
|---|---|
| badargument | Null or empty string for object group or new master site name. |
| wrongsite | This procedure must be called from the new master site. |
| unknownsite | Given site is not recognized by object group. |
| wrongstate | Status of the new master site must be QUIESCED. |
| dbms_repcat.missingrep-group | GNAME does not exist as a replicated object group. |

# DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS

## Purpose

To resume replication activity at all existing sites except the new site during offline instantiation of a replicated object group. You must call this procedure from the master definition site. For additional information, see "Snapshot Cloning and Offline Instantiation" on page 7-14.

## Syntax

The parameters for the RESUME_SUBSET_OF_MASTERS procedure are described in Table 9–57, and the exceptions are listed in Table 9–58. The syntax for this procedure is shown below.

```
DBMS_OFFLINE_OG.RESUME_SUBSET_OF_MASTERS(
     gname     IN  VARCHAR2,
     new_site  IN  VARCHAR2)
```

***Table 9–57 Parameters for RESUME_SUBSET_OF_MASTERS***

| Parameter | Description |
|---|---|
| gname | The name of the object group that you are replicating to the new site. |
| new_site | The fully qualified database name of the new site to which you are replicating the object group. |

***Table 9–58 Exceptions for RESUME_SUBSET_OF_MASTERS***

| Exception | Description |
|---|---|
| badargument | Null or empty string for object group or new master site name. |
| dbms_repcat.nonmasterdef | This procedure must be called from the master definition site. |
| unknownsite | Given site is not recognized by object group. |
| wrongstate | Status of master definition site must be QUIESCED. |
| dbms_repcat.missingrepgroup | GNAME does not exist as a replicated object group. |

# DBMS_OFFLINE_SNAPSHOT Package

The DBMS_OFFLINE_SNAPSHOT package contains the following procedures:

- DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD
- DBMS_OFFLINE_SNAPSHOT.END_LOAD

The following pages discuss each procedure.

# DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD

## Purpose

To prepare a snapshot site for import of a new snapshot as part of offline instantiation. You must call this procedure from the snapshot site for the new snapshot. For additional information, see "Snapshot Cloning and Offline Instantiation" on page 7-14.

## Syntax

The parameters for the BEGIN_LOAD procedure are described in Table 9–59, and the exceptions are listed in Table 9–60. The syntax for this procedure is shown below.

```
DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD(
     gname               IN   VARCHAR2,
     sname               IN   VARCHAR2,
     master_site         IN   VARCHAR2,
     snapshot_oname      IN   VARCHAR2,
     storage_c           IN   VARCHAR2 := '',
     comment             IN   VARCHAR2 := '',
     min_communication   IN   BOOLEAN  := TRUE)
```

***Table 9–59 Parameters for BEGIN_LOAD***

| Parameter | Description |
|---|---|
| gname | The name of the object group for the snapshot that you are creating using offline instantiation. |
| sname | The name of the schema for the new snapshot. |
| master_site | The fully qualified database name of the snapshot's master site. |
| snapshot_oname | The name of the temporary snapshot created at the master site. |
| storage_c | The storage options to use when creating the new snapshot at the snapshot site. |
| comment | User comment. |

*Table 9–59 Parameters for BEGIN_LOAD*

| Parameter | Description |
|---|---|
| min_communication | If TRUE, the update trigger sends the new value of a column only if the update statement modifies the column. The update trigger sends the old value of the column only if it is a key column or a column in a modified column group. |

*Table 9–60 Exceptions for BEGIN_LOAD*

| Exception | Description |
|---|---|
| badargument | Null or empty string for object group, schema, master site, or snapshot name. |
| dbms_repcat.missingrepgroup | GNAME does not exist as a replicated object group. |
| missingremotesnap | Could not locate given snapshot at given master site. |
| dbms_repcat.missingschema | The given schema does not exist. |
| snaptabmismatch | The base table name of the snapshot at the master and snapshot do not match. |

# DBMS_OFFLINE_SNAPSHOT.END_LOAD

## Purpose

To complete offline instantiation of a snapshot. You must call this procedure from the snapshot site for the new snapshot. For additional information, see "Snapshot Cloning and Offline Instantiation" on page 7-14.

## Syntax

The parameters for the END_LOAD procedure are described in Table 9–61, and the exceptions are listed in Table 9–62. The syntax for this procedure is shown below.

```
DBMS_OFFLINE_SNAPSHOT.END_LOAD(
      gname           IN  VARCHAR2,
      sname           IN  VARCHAR2,
      snapshot_oname  IN  VARCHAR2)
```

***Table 9–61 Parameters for END_LOAD***

| Parameter | Description |
|---|---|
| gname | The name of the object group for the snapshot that you are creating using offline instantiation. |
| sname | The name of the schema for the new snapshot. |
| snapshot_oname | The name of the snapshot. |

***Table 9–62 Exceptions for END_LOAD***

| Exception | Description |
|---|---|
| badargument | Null or empty string for object group, schema, or snapshot name. |
| dbms_repcat.missingrep-group | GNAME does not exist as a replicated object group. |
| dbms_repcat.nonsnapshot | This procedure must be called from the snapshot site. |

# DBMS_RECTIFIER_DIFF Package

The DBMS_RECTIFIER_DIFF package contains the following procedures:

- DBMS_RECTIFIER_DIFF.DIFFERENCES
- DBMS_RECTIFIER_DIFF.RECTIFY

The following pages discuss each procedure.

# DBMS_RECTIFIER_DIFF.DIFFERENCES

## Purpose

To determine the differences between two tables.

## Syntax

The parameters for the DIFFERENCES procedure are described in Table 9–63, and the exceptions are listed in Table 9–64. The syntax for this procedure is shown below.

```
DBMS_RECTIFIER_DIFF.DIFFERENCES(
      sname1               IN  VARCHAR2,
      oname1               IN  VARCHAR2,
      reference_site       IN  VARCHAR2 := '',
      sname2               IN  VARCHAR2,
      oname2               IN  VARCHAR2,
      comparison_site      IN  VARCHAR2 := '',
      where_clause         IN  VARCHAR2 := '',
      { column_list        IN  VARCHAR2 := '',
      | array_columns      IN  dbms_utility.name_array, }
      missing_rows_sname   IN  VARCHAR2,
      missing_rows_oname1  IN  VARCHAR2,
      missing_rows_oname2  IN  VARCHAR2,
      missing_rows_site    IN  VARCHAR2 := '',
      max_missing          IN  INTEGER,
      commit_rows          IN  INTEGER := 500)
```

**Note:** This procedure is overloaded. The COLUMN_LIST and ARRAY_COLUMNS parameters are mutually exclusive.

***Table 9–63 Parameters for DIFFERENCES***

| Parameter | Description |
|---|---|
| sname1 | The name of the schema at REFERENCE_SITE. |
| oname1 | The name of the table at REFERENCE_SITE. |
| reference_site | The name of the reference database site. The default, NULL, indicates the current site. |
| sname2 | The name of the schema at COMPARISON_SITE. |
| oname2 | The name of the table at COMPARISON_SITE. |

***Table 9–63 Parameters for DIFFERENCES***

| Parameter | Description |
|---|---|
| comparison_site | The name of the comparison database site. The default, NULL, indicates the current site. |
| where_clause | Only rows satisfying this restriction are selected for comparison. The default, NULL, indicates the current site. |
| column_list | A comma-separated list of one or more column names being compared for the two tables. You must not have any white space before or after the comma. The default, NULL, indicates that all columns be compared. |
| array_columns | A PL/SQL table of column names being compared for the two tables. Indexing begins at 1, and the final element of the array must be NULL. If position 1 is NULL, all columns are used. |
| missing_rows_sname | The name of the schema containing the tables with the missing rows. |
| missing_rows_oname1 | The name of the table at MISSING_ROWS_SITE that stores information about the rows in the table at REFERENCE site missing from the table at COMPARISON site and the rows at COMPARISON site missing from the table at REFERENCE site. |
| missing_rows_oname2 | The name of the table at MISSING_ROWS_SITE that stores about the missing rows. This table has three columns: the rowid of the row in the MISSING_ROWS_ONAME1 table, the name of the site at which the row is present, and the name of the site from which the row is absent. |
| missing_rows_site | The name of the site where the MISSING_ROWS_ONAME1 and MISSING_ROWS_ONAME2 tables are located. The default, NULL, indicates that the tables are located at the current site. |

*Table 9–63 Parameters for DIFFERENCES*

| Parameter | Description |
|---|---|
| max_missing | An integer that refers to the maximum number of rows that should be inserted into the "missing_rows_oname" table. If more than "max_missing" number of rows is missing, that many rows will be inserted into "missing_rows_oname", and the routine then returns normally without determining whether more rows are missing; this argument is useful in the cases that the fragments are so different that the missing rows table will have too many entries and there's no point in continuing. Raises exception badnumber if "max_missing" is less than 1 or NULL. |
| commit_rows | The maximum number of rows to insert to or delete from the reference or comparison table before a COMMIT occurs. By default, a COMMIT occurs after 500 inserts or 500 deletes. An empty string (' ') or NULL indicates that a COMMIT should only be issued after all rows for a single table have been inserted or deleted. |

*Table 9–64 Exceptions for DIFFERENCES*

| Exception | Description |
|---|---|
| nosuchsite | Database site could not be found. |
| badnumber | COMMIT_ROWS parameter less than 1. |
| missingprimarykey | Column list must include primary key (or SET_COLUMNS equivalent). |
| badname | NULL or empty string for table or schema name. |
| cannotbenull | Parameter cannot be NULL. |
| notshapeequivalent | Tables being compared are not shape equivalent. Shape refers to the number of columns, their column names, and the column datatypes. |
| unknowncolumn | Column does not exist. |
| unsupportedtype | Type not supported. |
| dbms_repcat.commfailure | Remote site is inaccessible. |
| dbms_repcat.missingobject | Table does not exist. |

## Restrictions

The error ORA-00001 (Unique constraint violated) is issued when there are any unique or primary key constraints on the MISSING_ROWS_DATA table.

# DBMS_RECTIFIER_DIFF.RECTIFY

## Purpose

To resolve the differences between two tables.

## Syntax

The parameters for the RECTIFY procedure are described in Table 9–65, and the exceptions are listed in Table 9–66. The syntax for this procedure is shown below.

```
DBMS_RECTIFIER_DIFF.RECTIFY(
      sname1               IN  VARCHAR2,
      oname1               IN  VARCHAR2,
      reference_site       IN  VARCHAR2 := '',
      sname2               IN  VARCHAR2,
      oname2               IN  VARCHAR2,
      comparison_site      IN  VARCHAR2 := '',
      { column_list        IN  VARCHAR2 := '',
      | array_columns      IN  dbms_utility.name_array, }
      missing_rows_sname   IN  VARCHAR2,
      missing_rows_oname1  IN  VARCHAR2,
      missing_rows_oname2  IN  VARCHAR2,
      missing_rows_site    IN  VARCHAR2 := '',
      commit_rows          IN  INTEGER := 500)
```

**Note:** This procedure is overloaded. The COLUMN_LIST and ARRAY_COLUMNS parameters are mutually exclusive.

***Table 9–65 Parameters for RECTIFY***

| Parameter | Description |
|---|---|
| sname1 | The name of the schema at REFERENCE_SITE. |
| oname1 | The name of the table at REFERENCE_SITE. |
| reference_site | The name of the reference database site. The default, NULL, indicates the current site. |
| sname2 | The name of the schema at COMPARISON_SITE. |
| oname2 | The name of the table at COMPARISON_SITE. |
| comparison_site | The name of the comparison database site. The default, NULL, indicates the current site. |

***Table 9–65 Parameters for RECTIFY***

| Parameter | Description |
|---|---|
| column_list | A comma-separated list of one or more column names being compared for the two tables. You must not have any white space before or after the comma. The default, NULL, indicates that all columns be compared. |
| array_columns | A PL/SQL table of column names being compared for the two tables. Indexing begins at 1, and the final element of the array must be NULL. If position 1 is NULL, all columns are used. |
| missing_rows_sname | The name of the schema containing the tables with the missing rows. |
| missing_rows_oname1 | The name of the table at MISSING_ROWS_SITE that stores information about the rows in the table at REFERENCE site missing from the table at COMPARISON site and the rows at COMPARISON site missing from the table at REFERENCE site. |
| missing_rows_oname2 | The name of the table at MISSING_ROWS_SITE that stores about the missing rows. This table has three columns: the rowid of the row in the MISSING_ROWS_ONAME1 table, the name of the site at which the row is present, and the name of the site from which the row is absent. |
| missing_rows_site | The name of the site where the MISSING_ROWS_ONAME1 and MISSING_ROWS_ONAME2 tables are located. The default, NULL, indicates that the tables are located at the current site. |
| commit_rows | The maximum number of rows to insert to or delete from the reference or comparison table before a COMMIT occurs. By default, a COMMIT occurs after 500 inserts or 500 deletes. An empty string (' ') or NULL indicates that a COMMIT should only be issued after all rows for a single table have been inserted or deleted. |

***Table 9–66 Exceptions for RECTIFY***

| Exception | Description |
|---|---|
| nosuchsite | Database site could not be found. |
| badnumber | COMMIT_ROWS parameter less than 1. |
| badname | NULL or empty string for table or schema name. |
| dbms_repcat.commfailure | Remote site is inaccessible. |
| dbms_repcat.missingobject | Table does not exist. |

# DBMS_REFRESH Package

The DBMS_REFRESH package contains the following procedures:

- DBMS_REFRESH.ADD
- DBMS_REFRESH.CHANGE
- DBMS_REFRESH.DESTROY
- DBMS_REFRESH.MAKE
- DBMS_REFRESH.REFRESH
- DBMS_REFRESH.SUBTRACT

The following pages discuss each procedure.

# DBMS_REFRESH.ADD

## Purpose

To add snapshots to a refresh group. For additional information, see "Managing Snapshot Refreshes and Refresh Groups" on page 2-37.

## Syntax

The parameters for the ADD procedure are described in Table 9–67. The syntax for this procedure is shown below.

```
DBMS_REFRESH.ADD(
      name     IN VARCHAR2,
      { list   IN VARCHAR2,
      | tab    IN DBMS_UTILITY.UNCL_ARRAY, }
      lax      IN BOOLEAN := FALSE)
```

**Note:** This procedure is overloaded. The LIST and TAB parameters are mutually exclusive.

***Table 9–67 Parameters for ADD***

| Parameter | Description |
|---|---|
| name | Name of the refresh group to which you want to add members. |
| list | Comma-separated list of snapshots that you want to add to the refresh group. (Synonyms are not supported.) |
| tab | Instead of a comma-separated list, you can supply a PL/SQL table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a snapshot. The first snapshot should be in position 1. The last position must be NULL. |
| lax | A snapshot can belong to only one refresh group at a time. If you are moving a snapshot from one group to another, you must set the LAX flag to TRUE to succeed. Oracle then automatically removes the snapshot from the other refresh group and updates its refresh interval to be that of its new group. Otherwise, the call to ADD generates an error message. |

# DBMS_REFRESH.CHANGE

## Purpose

To change the refresh interval for a snapshot group. For additional information, see "Managing Snapshot Refreshes and Refresh Groups" on page 2-37.

## Syntax

The parameters for the CHANGE procedure are described in Table 9–68. The syntax for this procedure is shown below:

```
DBMS_REFRESH.CHANGE(
      name                 IN VARCHAR2,
      next_date            IN DATE           := NULL,
      interval             IN VARCHAR2       := NULL,
      implicit_destroy     IN BOOLEAN        := NULL,
      rollback_seg         IN VARCHAR2       := NULL,
      push_deferred_rpc    IN BOOLEAN        := NULL,
      refresh_after_errors IN BOOLEAN        := NULL,
      purge_option         IN BINARY_INTEGER := NULL,
      parallelism          IN BINARY_INTEGER := NULL,
      heap_size            IN BINARY_INTEGER := NULL)
```

***Table 9–68 Parameters for CHANGE***

| Parameter | Description |
|---|---|
| name | Name of the refresh group for which you want to alter the refresh interval. |
| next_date | Next date that you want a refresh to occur. By default, this date remains unchanged. |
| interval | Function used to calculate the next time to refresh the snapshots in the group. This interval is evaluated immediately before the refresh. Thus, you should select an interval that is greater than the time it takes to perform a refresh. By default, the interval remains unchanged. |
| implicit_destroy | Allows you to reset the value of the IMPLICIT_DESTROY flag. If this flag is set, Oracle automatically deletes the group if it no longer contains any members. By default, this flag remains unchanged. |

***Table 9–68 Parameters for CHANGE***

| Parameter | Description |
|---|---|
| rollback_seg | Allows you to change the rollback segment used. By default, the rollback segment remains unchanged. To reset this parameter to use the default rollback segment, specify 'NULL', including the quotes. Specifying NULL without quotes indicates that you do not want to change the rollback segment currently being used. |
| push_deferred_rpc | Used by updatable snapshots only. Set this parameter to TRUE if you want to push changes from the snapshot to its associated master before refreshing the snapshot. Otherwise, these changes may appear to be temporarily lost. By default, this flag remains unchanged |
| refresh_after_errors | Used by updatable snapshots only. Set this parameter to TRUE if you want the refresh to proceed even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master. By default, this flag remains unchanged. |
| purge_option | If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), 0 = don't purge; 1 = lazy (default); 2 = aggressive. In most cases, *lazy* purge is the optimal setting. Set purge to *aggressive* to trim back the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, set purge to *don't purge* and occasionally execute PUSH with purge set to *aggressive* to reduce the queue. |
| parallelism | 0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process. |
| heap_size | Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set the parameter unless so directed by Oracle Worldwide Support. |

# DBMS_REFRESH.DESTROY

## Purpose

To remove all of the snapshots from a refresh group and delete the refresh group. For additional information, see "Managing Snapshot Refreshes and Refresh Groups" on page 2-37.

## Syntax

The parameter for the DESTROY procedure is described in Table 9–69. The syntax for this procedure is shown below:

```
DBMS_REFRESH.DESTROY(name    IN    VARCHAR2)
```

***Table 9–69 Parameter for DESTROY***

| Parameter | Description |
|---|---|
| name | Name of the refresh group that you want to destroy. |

# DBMS_REFRESH.MAKE

## Purpose

To specify the members of a refresh group and the time interval used to determine when the members of this group should be refreshed. For additional information, see "Managing Snapshot Refreshes and Refresh Groups" on page 2-37.

## Syntax

The parameters for the MAKE procedure are described in Table 9–70. The syntax for this procedure is shown below:

```
DBMS_REFRESH.MAKE(
      name                 IN    VARCHAR2
      { list               IN    VARCHAR2,
      | tab                   IN     DBMS_UTILITY.UNCL_ARRAY,}
       next_date             IN     DATE,
       interval              IN     VARCHAR2,
       implicit_destroy      IN     BOOLEAN          := FALSE,
       lax                   IN     BOOLEAN          := FALSE,
       job                   IN     BINARY INTEGER   := 0,
       rollback_seg          IN     VARCHAR2         := NULL,
       push_deferred_rpc     IN     BOOLEAN          := TRUE,
       refresh_after_errors  IN     BOOLEAN          := FALSE)
      purge_option          IN     BINARY_INTEGER := NULL,
      parallelism           IN     BINARY_INTEGER := NULL,
      heap_size             IN     BINARY_INTEGER := NULL)
```

**Note:** This procedure is overloaded. The LIST and TAB parameters are mutually exclusive.

***Table 9–70 Parameters for MAKE***

| Parameter | Description |
|---|---|
| name | Unique name used to identify the refresh group. Refresh groups must follow the same naming conventions as tables. |

*Table 9–70 Parameters for MAKE*

| Parameter | Description |
|---|---|
| list | Comma-separated list of snapshots that you want to refresh. (Synonyms are not supported.) These snapshots can be located in different schemas and have different master tables; however, all of the listed snapshots must be in your current database. |
| tab | Instead of a comma separated list, you can supply a PL/SQL table of names of snapshots that you want to refresh using the datatype DBMS_UTILITY.UNCL_ARRAY. If the table contains the names of N snapshots, the first snapshot should be in position 1 and the N + 1 position should be set to null. |
| next_date | Next date that you want a refresh to occur. |
| interval | Function used to calculate the next time to refresh the snapshots in the group. This field is used with the NEXT_DATE value. For example, if you specify NEXT_DAY(SYSDATE+1, "MONDAY") as your interval, and your NEXT_DATE evaluates to Monday, Oracle will refresh the snapshots every Monday. This interval is evaluated immediately before the refresh. Thus, you should select an interval that is greater than the time it takes to perform a refresh. See "Example Date Expressions" on page 2-26. |
| implicit_destroy | Set this argument to TRUE if you want to delete the refresh group automatically when it no longer contains any members. Oracle checks this flag only when you call the SUBTRACT procedure. That is, setting this flag still allows you to create an empty refresh group. |
| lax | A snapshot can belong to only one refresh group at a time. If you are moving a snapshot from an existing group to a new refresh group, you must set the LAX flag to TRUE to succeed. Oracle then automatically removes the snapshot from the other refresh group and updates its refresh interval to be that of its new group. Otherwise, the call to MAKE generates an error message. |
| job | This parameter is needed by the Import utility. Use the default value, 0. |

***Table 9–70 Parameters for MAKE***

| Parameter | Description |
|---|---|
| rollback_seg | Name of the rollback segment to use while refreshing snapshots. The default, null, uses the default rollback segment. |
| push_deferred_rpc | Used by updatable snapshots only. Use the default value, TRUE, if you want to push changes from the snapshot to its associated master before refreshing the snapshot. Otherwise, these changes may appear to be temporarily lost. |
| refresh_after_errors | Used by updatable snapshots only. Set this parameter to TRUE if you want the refresh to proceed even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master. |
| purge_option | If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), 0 = don't purge; 1 = lazy (default); 2 = aggressive. In most cases, *lazy* purge is the optimal setting. Set purge to *aggressive* to trim back the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, set purge to *don't purge* and occasionally execute PUSH with purge set to *aggressive* to reduce the queue. |
| parallelism | 0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process. |
| heap_size | Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set the parameter unless so directed by Oracle Worldwide Support. |

# DBMS_REFRESH.REFRESH

## Purpose

To manually refresh a refresh group. For additional information, see “Managing Snapshot Refreshes and Refresh Groups” on page 2-37.

## Syntax

The parameter for the REFRESH procedure is described in Table 9–71. The syntax for this procedure is shown below:

```
DBMS_REFRESH.REFRESH(name    IN     VARCHAR2)
```

***Table 9–71 Parameter for REFRESH***

| Parameter | Description |
|---|---|
| name | Name of the refresh group that you want to refresh manually. |

# DBMS_REFRESH.SUBTRACT

## Purpose

To remove snapshots from a refresh group. For additional information, see "Managing Snapshot Refreshes and Refresh Groups" on page 2-37.

## Syntax

The parameters for the SUBTRACT procedure are described in Table 9–72. The syntax for this procedure is shown below:

```
DBMS_REFRESH.SUBTRACT(
      name       IN    VARCHAR2,
      { list     IN    VARCHAR2,
      | tab        IN    DBMS_UTILITY.UNCL_ARRAY, }
      lax          IN    BOOLEAN := FALSE)
```

**Note:** This procedure is overloaded. The LIST and TAB parameters are mutually exclusive.

***Table 9–72 Parameters for SUBTRACT***

| Parameter | Description |
|---|---|
| name | Name of the refresh group from which you want to remove members. |
| list | Comma-separated list of snapshots that you want to remove from the refresh group. (Synonyms are not supported.) These snapshots can be located in different schemas and have different master tables; however, all of the listed snapshots must be in your current database. |
| tab | Instead of a comma-separated list, you can supply a PL/SQL table of names of snapshots that you want to refresh using the datatype DBMS_UTILITY.UNCL_ARRAY. If the table contains the names of N snapshots, the first snapshot should be in position 1 and the N+1 position should be set to NULL. |
| lax | Set this parameter to FALSE if you want Oracle to generate an error message if the snapshot you are attempting to remove is not a member of the refresh group. |

# DBMS_REPCAT Package

The DBMS_REPCAT package includes the following procedures and functions:

- DBMS_REPCAT.ADD_GROUPED_COLUMN
- DBMS_REPCAT.ADD_MASTER_DATABASE
- DBMS_REPCAT.ADD_PRIORITY_datatype
- DBMS_REPCAT.ADD_SITE_PRIORITY_SITE
- DBMS_REPCAT.ADD_conflicttype_RESOLUTION
- DBMS_REPCAT.ALTER_MASTER_PROPAGATION
- DBMS_REPCAT.ALTER_MASTER_REPOBJECT
- DBMS_REPCAT.ALTER_PRIORITY
- DBMS_REPCAT.ALTER_PRIORITY_datatype
- DBMS_REPCAT.ALTER_SITE_PRIORITY
- DBMS_REPCAT.ALTER_SITE_PRIORITY_SITE
- DBMS_REPCAT.ALTER_SNAPSHOT_PROPAGATION
- DBMS_REPCAT.CANCEL_STATISTICS
- DBMS_REPCAT.COMMENT_ON_COLUMN_GROUP
- DBMS_REPCAT.COMMENT_ON_PRIORITY_GROUP/
- DBMS_REPCAT.COMMENT_ON_REPGROUP
- DBMS_REPCAT.COMMENT_ON_REPSITES
- DBMS_REPCAT.COMMENT_ON_REPOBJECT
- DBMS_REPCAT.COMMENT_ON_conflicttype_RESOLUTION
- DBMS_REPCAT.CREATE_MASTER_REPGROUP
- DBMS_REPCAT.CREATE_MASTER_REPOBJECT
- DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP
- DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT
- DBMS_REPCAT.DEFINE_COLUMN_GROUP
- DBMS_REPCAT.DEFINE_PRIORITY_GROUP
- DBMS_REPCAT.DEFINE_SITE_PRIORITY

- DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN
- DBMS_REPCAT.DROP_COLUMN_GROUP
- DBMS_REPCAT.DROP_GROUPED_COLUMN
- DBMS_REPCAT.DROP_MASTER_REPGROUP
- DBMS_REPCAT.DROP_MASTER_REPOBJECT
- DBMS_REPCAT.DROP_PRIORITY
- DBMS_REPCAT.DROP_PRIORITY_GROUP
- DBMS_REPCAT.DROP_PRIORITY_datatype
- DBMS_REPCAT.DROP_SITE_PRIORITY
- DBMS_REPCAT.DROP_SITE_PRIORITY_SITE
- DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP
- DBMS_REPCAT.DROP_SNAPSHOT_REPOBJECT
- DBMS_REPCAT.DROP_conflicttype_RESOLUTION
- DBMS_REPCAT.EXECUTE_DDL
- DBMS_REPCAT.GENERATE_REPLICATION_PACKAGE
- DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT
- DBMS_REPCAT.GENERATE_REPLICATION_TRIGGER
- DBMS_REPCAT.GENERATE_SNAPSHOT_SUPPORT
- DBMS_REPCAT.MAKE_COLUMN_GROUP
- DBMS_REPCAT.PURGE_MASTER_LOG
- DBMS_REPCAT.PURGE_STATISTICS
- DBMS_REPCAT.REFRESH_SNAPSHOT_REPGROUP
- DBMS_REPCAT.REGISTER_SNAPSHOT_REPGROUP
- DBMS_REPCAT.REGISTER_STATISTICS
- DBMS_REPCAT.RELOCATE_MASTERDEF
- DBMS_REPCAT.REMOVE_MASTER_DATABASES
- DBMS_REPCAT.REPCAT_IMPORT_CHECK
- DBMS_REPCAT.RESUME_MASTER_ACTIVITY

- DBMS_REPCAT.SEND_AND_COMPARE_OLD_VALUES
- DBMS_REPCAT.SET_COLUMNS
- DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY
- DBMS_REPCAT.SWITCH_SNAPSHOT_MASTER
- DBMS_REPCAT.UNREGISTER_SNAPSHOT_REPGROUP
- DBMS_REPCAT.VALIDATE
- DBMS_REPCAT.WAIT_MASTER_LOG

The following pages discuss each procedure and function.

# DBMS_REPCAT.ADD_GROUPED_COLUMN

## Purpose

To add members to an existing column group. You must call this procedure from the master definition site.

## Syntax

The parameters for the ADD_GROUPED_COLUMN procedure are described in Table 9–73, and the exceptions are listed in Table 9–74. The syntax for this procedure is shown below:

```
DBMS_REPCAT.ADD_GROUPED_COLUMN(
      sname,                IN   VARCHAR2,
      oname,                IN   VARCHAR2,
      column_group          IN   VARCHAR2,
      list_of_column_names  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S)
```

***Table 9–73 Parameters for ADD_GROUPED_COLUMN***

| Parameter | Description |
|---|---|
| sname | The schema in which the replicated table is located. |
| oname | The name of the replicated table with which the column group is associated. |
| column_group | The name of the column group to which you are adding members. |
| list_of_column_names | The names of the columns that you are adding to the designated column group. This can either be a comma-separated list or a PL/SQL table of column names. The PL/SQL table must be of type dbms_repcat.varchar2s. Use the single value '*' to create a column group that contains all of the columns in your table. |

*Table 9–74 Exceptions for ADD_GROUPED_COLUMN*

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingobject | The given table does not exist. |
| missinggroup | The given column group does not exist. |
| missingcolumn | A given column does not exist in the designated table. |
| duplicatecolumn | The given column is already a member of another column group. |
| missingschema | The given schema does not exist. |
| notquiesced | The object group that the given table belongs to is not quiesced. |

# DBMS_REPCAT.ADD_MASTER_DATABASE

## Purpose

To add another master site to your replicated environment. This procedure regenerates all the triggers and their associated packages at existing master sites. You must call this procedure from the master definition site.

## Syntax

The parameters for the ADD_MASTER_DATABASE procedure are described in Table 9–75, and the exceptions are listed in Table 9–76. The syntax for this procedure is shown below:

```
DBMS_REPCAT.ADD_MASTER_DATABASE(
      gname                 IN   VARCHAR2,
      master                IN   VARCHAR2,
      use_existing_objects  IN   BOOLEAN  := TRUE,
      copy_rows             IN   BOOLEAN  := TRUE,
      comment               IN   VARCHAR2 := '',
      propagation_mode      IN   VARCHAR2 := 'ASYNCHRONOUS')
```

***Table 9–75 Parameters for ADD_MASTER_DATABASE***

| Parameter | Description |
|---|---|
| gname | The name of the object group being replicated. This object group must already exist at the master definition site. |
| master | The fully qualified database name of the new master database. |
| use_existing_objects | Indicate TRUE if you want to reuse any objects of the same type and shape that already exist in the schema at the new master site. See “Replicating Object Definitions to Master Sites” on page 3-21 for more information on how these changes are applied. |
| copy_rows | Indicate TRUE if you want the initial contents of a table at the new master site to match the contents of the table at the master definition site. |
| comment | This comment is added to the MASTER_COMMENT field of the RepSite view. |

*Table 9–75 Parameters for ADD_MASTER_DATABASE*

| Parameter | Description |
|---|---|
| propagation_mode | Method of forwarding changes to and receiving changes from new master database. Accepted values are SYNCHRONOUS and ASYNCHRONOUS. |

*Table 9–76 Exceptions for ADD_MASTER_DATABASE*

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| notquiesced | The replicated object group has not been suspended. |
| missingrepgroup | The object group does not exist at the given database site. |
| commfailure | The new master is not accessible. |
| typefailure | An incorrect propagation mode was specified. |
| notcompat | Compatibility mode must be 7.3.0.0 or greater. |
| duplrepgrp | The master site already exists. |

# DBMS_REPCAT.ADD_PRIORITY_*datatype*

## Purpose

To add a member to a priority group. You must call this procedure from the master definition site. The procedure that you must call is determined by the datatype of your "priority" column. You must call this procedure once for each of the possible values of the "priority" column.

For additional information, see "Priority Groups and Site Priority" on page 5-20.

## Syntax

The parameters for the ADD_PRIORITY_*datatype* procedure are described in Table 9–77, and the exceptions are listed in Table 9–78. The syntax for the ADD_PRIORITY_*datatype* procedure is shown below.

```
DBMS_REPCAT.ADD_PRIORITY_datatype(
      gname              IN   VARCHAR2,
      pgroup             IN   VARCHAR2,
      value              IN   datatype,
      priority           IN   NUMBER)
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

***Table 9–77 Parameters for ADD_PRIORITY_datatype***

| Parameter | Description |
|---|---|
| gname | The replicated object group for which you are creating a priority group. |
| pgroup | The name of the priority group. |
| value | The value of the priority group member. This would be one of the possible values of the associated "priority" column of a table using this priority group. |
| priority | The priority of this value. The higher the number, the higher the priority. |

***Table 9–78 Exceptions for ADD_PRIORITY_datatype***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| duplicatevalue | The given value already exists in the priority group. |
| duplicatepriority | The given priority already exists in the priority group. |
| missingrepgroup | The given replicated object group does not exist. |
| missingprioritygroup | The given priority group does not exist. |
| typefailure | The given value has the incorrect datatype for the priority group. |
| notquiesced | The given replicated object group is not quiesced. |

# DBMS_REPCAT.ADD_SITE_PRIORITY_SITE

## Purpose

To add a new site to a site priority group. You must call this procedure from the master definition site. For additional information, see "Adding a Site to the Group" on page 5-29.

## Syntax

The parameters for the ADD_SITE_PRIORITY_SITE procedure are described in Table 9–79, and the exceptions are listed in Table 9–80. The syntax for this procedure is shown below:

```
DBMS_REPCAT.ADD_SITE_PRIORITY_SITE(
      gname           IN   VARCHAR2,
      name            IN   VARCHAR2
      site            IN   VARCHAR2,
      priority        IN   NUMBER)
```

***Table 9–79 Parameters for ADD_SITE_PRIORITY_SITE***

| Parameter | Description |
|---|---|
| gname | The replicated object group for which you are adding a site to a group. |
| name | The name of the site priority group to which you are adding a member. |
| site | The global database name of the site that you are adding. |
| priority | The priority level of the site that you are adding. A higher number indicates a higher priority level. |

***Table 9–80 Exceptions for ADD_SITE_PRIORITY_SITE***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingrepgroup | The given replicated object group does not exist. |
| missingpriority | The given site priority group does not exist. |
| duplicatepriority | The given priority level already exists for another site in the group. |

***Table 9–80 Exceptions for ADD_SITE_PRIORITY_SITE***

| Exception | Description |
|---|---|
| duplicatevalue | The given site already exists in the site priority group. |
| notquiesced | The replicated object group is not quiesced. |

# DBMS_REPCAT.ADD_*conflicttype*_RESOLUTION

## Purpose

To designate a method for resolving an update, delete, or uniqueness conflict. You must call these procedures from the master definition site. The procedure that you need to call is determined by the type of conflict that the routine resolves.

| Conflict Type | Procedure Name |
|---|---|
| update | ADD_UPDATE_RESOLUTION |
| uniqueness | ADD_UNIQUE_RESOLUTION |
| delete | ADD_DELETE_RESOLUTION |

For more information about designating methods to resolve update conflicts, see "Assigning an Update Conflict Resolution Method" on page 5-15. For more information about selecting uniqueness conflict resolution methods, see "Assigning a Uniqueness Conflict Resolution Method" on page 5-33. For more information about assigning delete conflict resolution methods, see "Assigning a Delete Conflict Resolution Method" on page 5-36.

## Syntax

The parameters for the ADD_*conflicttype*_RESOLUTION procedure are described in Table 9–81, and the exceptions are listed in Table 9–82. The syntax for the ADD_UPDATE_RESOLUTION procedure is shown below:

```
DBMS_REPCAT.ADD_UPDATE_RESOLUTION(
      sname                  IN   VARCHAR2,
      oname                  IN   VARCHAR2,
      column_group           IN   VARCHAR2,
      sequence_no            IN   NUMBER,
      method                 IN   VARCHAR2,
      parameter_column_name  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S,
      priority_group         IN   VARCHAR2     := NULL,
      function_name          IN   VARCHAR2     := NULL,
      comment                IN   VARCHAR2     := NULL)
```

The syntax for the ADD_DELETE_RESOLUTION procedure is shown below:

```
DBMS_REPCAT.ADD_DELETE_RESOLUTION(
```

```
    sname                  IN   VARCHAR2,
    oname                  IN   VARCHAR2,
    sequence_no            IN   NUMBER,
    parameter_column_name  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S,
    function_name          IN   VARCHAR2,
    comment                IN   VARCHAR2     := NULL)
```

The syntax for the ADD_UNIQUE_RESOLUTION procedure is shown below:

```
DBMS_REPCAT.ADD_UNIQUE_RESOLUTION(
     sname                  IN   VARCHAR2,
     oname                  IN   VARCHAR2,
     constraint_name        IN   VARCHAR2,
     sequence_no            IN   NUMBER,
     method                 IN   VARCHAR2,
     parameter_column_name  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S,
     function_name          IN   VARCHAR2     := NULL,
     comment               IN   VARCHAR2     := NULL)
```

***Table 9–81 Parameters for ADD_conflicttype_RESOLUTION***

| Parameter | Description |
|---|---|
| sname | The name of the schema containing the table to be replicated. |
| oname | The name of the table for which you are adding a conflict resolution routine. |
| column_group | The name of the column group for which you are adding a conflict resolution routine. Column groups are required for update conflict resolution routines only. |
| constraint_name | The name of the unique constraint or unique index for which you are adding a conflict resolution routine. Use the name of the unique index if it differs from the name of the associated unique constraint. Constraint names are required for uniqueness conflict resolution routines only. |
| sequence_no | The order in which the designated conflict resolution methods should be applied. |

***Table 9–81 Parameters for ADD_conflicttype_RESOLUTION***

| Parameter | Description |
|---|---|
| method | The type of conflict resolution routine that you want to create. This can be the name of one of the standard routines provided with advanced replication, or, if you have written your own routine, you should choose USER FUNCTION, and provide the name of your routine as the FUNCTION_NAME argument. The methods supported in this release are: MINIMUM, MAXIMUM, LATEST TIMESTAMP, EARLIEST TIMESTAMP, ADDITIVE, AVERAGE, PRIORITY GROUP, SITE PRIORITY, OVERWRITE, and DISCARD (for update conflicts) and APPEND SITE NAME, APPEND SEQUENCE NUMBER, and DISCARD (for uniqueness conflicts). There are no standard methods for delete conflicts, so this argument is not used. |
| parameter_column_name | The name of the columns used to resolve the conflict. The standard methods operate on a single column. For example, if you are using the LATEST TIMESTAMP method for a column group, you should pass the name of the column containing the timestamp value as this argument. If your are using a USER FUNCTION, you can resolve the conflict using any number of columns. This argument accepts either a comma separated list of column names, or a PL/SQL table of type dbms_repcat.varchar2s. The single value '*' indicates that you want to use all of the columns in the table (or column group, for update conflicts) to resolve the conflict. If you specify '*', the columns will be passed to your function in alphabetical order. |
| priority_group | If you are using the PRIORITY GROUP or SITE PRIORITY update conflict resolution method, you must supply the name of the priority group that you have created. See "Priority Groups and Site Priority" on page 5-20. If you are using a different method, you can use the default value for this argument, NULL. This argument is applicable to update conflicts only. |
| function_name | If you selected the USER FUNCTION method, or if you are adding a delete conflict resolution routine, you must supply the name of the conflict resolution routine that you have written. If you are using one of the standard methods, you can use the default value for this argument, NULL. |

*Table 9–81 Parameters for ADD_conflicttype_RESOLUTION*

| Parameter | Description |
|---|---|
| comment | This user comment is added to the RepResolution view. |

*Table 9–82 Exceptions for ADD_conflicttype_RESOLUTION*

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingobject | The given object does not exist as a table in the given schema using row-level replication. |
| missingschema | The given schema does not exist. |
| missingcolumn | The column that you specified as part of the PARAMETER_COLUMN_NAME argument does not exist. |
| missinggroup | The given column group does not exist. |
| missingprioritygroup | The priority group that you specified does not exist for the table. |
| invalidmethod | The resolution method that you specified is not recognized. |
| invalidparameter | The number of columns that you specified for the PARAMETER_COLUMN_NAME argument is invalid. (The standard routines take only one column name.) |
| missingfunction | The user function that you specified does not exist. |
| missingconstraint | The constraint that you specified for a uniqueness conflict does not exist. |
| notquiesced | The object group that the given table belongs to is not quiesced. |
| duplicateresolution | The given conflict resolution method is already registered. |
| paramtype | The type is different from the type assigned to the priority group. |

# DBMS_REPCAT.ALTER_MASTER_PROPAGATION

## Purpose

To alter the propagation method for a given object group at a given master site. This object group must be quiesced. You must call this procedure from the master definition site. If the master appears in the dblink_list or dblink_table, ALTER_MASTER_PROPAGATION ignores that database link. You cannot change the propagation mode from a master to itself.

## Syntax

The parameters for the ALTER_MASTER_PROPAGATION procedure are described in Table 9–83, and the exceptions are listed in Table 9–84. The syntax for this procedure is shown below:

```
DBMS_REPCAT.ALTER_MASTER_PROPAGATION(
      gname               IN   VARCHAR2,
      master              IN   VARCHAR2,
      { dblink_list       IN   VARCHAR2,
      | dblink_table      IN   dbms_utility.dblink_array,}
      propagation_mode    IN   VARCHAR2 : ='asynchronous',
      comment             IN   VARCHAR2 := '')
```

**Note:** This procedure is overloaded. The DBLINK_LIST and DBLINK_TABLE parameters are mutually exclusive.

*Table 9–83 Parameters for ALTER_MASTER_PROPAGATION*

| Parameter | Description |
|---|---|
| gname | The name of the object group to which to alter the propagation mode. |
| master | The name of the master site at which to alter the propagation mode. |
| dblink_list | A comma-separated list of database links for which to alter propagation. If null, all masters except the master site being altered will be used by default. |
| dblink_table | A PL/SQL table, indexed from position 1, of database links for which to alter propagation. |
| propagation_mode | Determines the manner in which changes from the given master site are propagated to the sites identified by the list of database links. Appropriate values are SYNCHRONOUS and ASYNCHRONOUS. |
| comment | This comment is added to the RepProp view. |

*Table 9–84 Exception for ALTER_MASTER_PROPAGATION*

| Exception | Description |
|---|---|
| nonmasterdef | The local site is not the master definition site. |
| notquiesced | The local site is not quiesced. |
| typefailure | The propagation mode specified was not recognized. |
| nonmaster | The list of database links includes a site that is not a master site. |

# DBMS_REPCAT.ALTER_MASTER_REPOBJECT

## Purpose

To alter an object in your replicated environment. You must call this procedure from the master definition site.

## Syntax

The parameters for the ALTER_MASTER_REPOBJECT procedure are described in Table 9–85, and the exceptions are listed in Table 9–86. The syntax for this procedure is shown below:

```
DBMS_REPCAT.ALTER_MASTER_REPOBJECT(
      sname         IN   VARCHAR2,
      oname         IN   VARCHAR2,
      type          IN   VARCHAR2,
      ddl_text      IN   VARCHAR2,
      comment       IN   VARCHAR2     := '',
      retry         IN   BOOLEAN      := FALSE)
```

**Note**: If the DDL is supplied without specifying a schema, the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema.

***Table 9–85 Parameters for ALTER_MASTER_REPOBJECT***

| Parameter | Description |
|---|---|
| sname | The schema containing the object that you want to alter. |
| oname | The name of the object that you want to alter. |
| type | The type of the object that you are altering. The types supported are: TABLE, INDEX, SYNONYM, TRIGGER, VIEW, PROCEDURE, FUNCTION, PACKAGE, and PACKAGE BODY. |
| ddl_text | The DDL text that you want used to alter the object. Oracle does not parse this DDL before applying it; therefore, you must ensure that your DDL text provides the appropriate schema and object name for the object being altered. |
| comment | If not null, this comment will be added to the COMMENT field of the RepObject view. |
| retry | If retry is TRUE, ALTER_MASTER_REPOBJECT alters the object only at masters whose object status is not VALID. |

***Table 9–86 Exceptions for ALTER_MASTER_REPOBJECT***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| notquiesced | The associated object group has not been suspended. |
| missingobject | The object identified by SNAME and ONAME does not exist. |
| typefailure | The given type parameter is not supported. |
| ddlfailure | DDL at the master definition site did not succeed. |
| commfailure | At least one master site is not accessible. |

# DBMS_REPCAT.ALTER_PRIORITY

## Purpose

To alter the priority level associated with a given priority group member. You must call this procedure from the master definition site. See "Altering the Priority of a Member" on page 5-25.

## Syntax

The parameters for the ALTER_PRIORITY procedure are described in Table 9–87, and the exceptions are listed in Table 9–88. The syntax for this procedure is shown below:

```
DBMS_REPCAT.ALTER_PRIORITY(
      gname            IN   VARCHAR2,
      pgroup           IN   VARCHAR2,
      old_priority     IN   NUMBER,
      new_priority     IN   NUMBER)
```

***Table 9–87 Parameters for ALTER_PRIORITY***

| Parameter | Description |
|---|---|
| gname | The replicated object group with which the priority group is associated. |
| pgroup | The name of the priority group containing the priority that you want to alter. |
| old_priority | The current priority level of the priority group member. |
| new_priority | The new priority level that you want assigned to the priority group member. |

***Table 9–88 Exceptions for ALTER_PRIORITY***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| duplicatepriority | The new priority level already exists in the priority group. |
| missingrepgroup | The given replicated object group does not exist. |
| missingvalue | The value was not registered by a call to DBMS_REPCAT.ADD_PRIORITY_*datatype*. |

***Table 9–88 Exceptions for ALTER_PRIORITY***

| Exception | Description |
|---|---|
| missingprioritygroup | The given priority group does not exist. |
| notquiesced | The given replicated object group is not quiesced. |

# DBMS_REPCAT.ALTER_PRIORITY_*datatype*

## Purpose

To alter the value of a member in a priority group. You must call this procedure from the master definition site. The procedure that you must call is determined by the datatype of your "priority" column.

For additional information, see "Altering the Value of a Member" on page 5-25.

## Syntax

The parameters for the ALTER_PRIORITY_*datatype* procedure are described in Table 9–89, and the exceptions are listed in Table 9–90. The syntax for the ALTER_PRIORITY_*datatype* procedure is shown below.

```
DBMS_REPCAT.ALTER_PRIORITY_datatype(
      gname        IN   VARCHAR2,
      pgroup       IN   VARCHAR2,
      old_value    IN   datatype,
      new_value    IN   datatype)
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

***Table 9–89 Parameters for ALTER_PRIORITY_datatype***

| Parameter | Description |
|---|---|
| gname | The replicated object group with which the priority group is associated. |
| pgroup | The name of the priority group containing the value that you want to alter. |
| old_value | The current value of the priority group member. |
| new_value | The new value that you want assigned to the priority group member. |

***Table 9–90 Exceptions for ALTER_PRIORITY_datatype***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| duplicatevalue | The new value already exists in the priority group. |
| missingrepgroup | The given replicated object group does not exist. |
| missingprioritygroup | The given priority group does not exist. |
| missingvalue | The old value does not already exist. |
| paramtype | The new value has the incorrect datatype for the priority group. |
| typefailure | The given value has the incorrect datatype for the priority group. |
| notquiesced | The given replicated object group is not quiesced. |

# DBMS_REPCAT.ALTER_SITE_PRIORITY

## Purpose

To alter the priority level associated with a given site. You must call this procedure from the master definition site. See "Altering the Priority Level of a Site" on page 5-29.

## Syntax

The parameters for the ALTER_SITE_PRIORITY procedure are described in Table 9–91, and the exceptions are listed in Table 9–92. The syntax for this procedure is shown below:

```
DBMS_REPCAT.ALTER_SITE_PRIORITY(
      gname          IN   VARCHAR2,
      name           IN   VARCHAR2
      old_priority   IN   NUMBER,
      new_priority   IN   NUMBER)
```

***Table 9–91 Parameters for ALTER_SITE_PRIORITY***

| Parameter | Description |
|---|---|
| gname | The replicated object group with which the site priority group is associated. |
| name | The name of the site priority group whose member you are altering. |
| old_priority | The current priority level of the site whose priority level you want to change. |
| new_priority | The new priority level for the site. A higher number indicates a higher priority level. |

*Table 9–92 Exceptions for ALTER_SITE_PRIORITY*

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingrepgroup | The given replicated object group does not exist. |
| missingpriority | The old priority level is not associated with any group members. |
| duplicatepriority | The new priority level already exists for another site in the group. |
| missingvalue | The old value does not already exist. |
| paramtype | The new value has the incorrect datatype for the priority group. |
| notquiesced | The replicated object group is not quiesced. |

# DBMS_REPCAT.ALTER_SITE_PRIORITY_SITE

## Purpose

To alter the site associated with a given priority level. You must call this procedure from the master definition site. See "Altering the Site Associated with a Priority Level" on page 5-30.

## Syntax

The parameters for the ALTER_SITE_PRIORITY_SITE procedure are described in Table 9–93, and the exceptions are listed in Table 9–94. The syntax for this procedure is shown below:

```
DBMS_REPCAT.ALTER_SITE_PRIORITY_SITE(
      gname     IN   VARCHAR2,
      name      IN   VARCHAR2
      old_site  IN   VARCHAR2,
      new_site  IN   VARCHAR2)
```

***Table 9–93 Parameters for ALTER_SITE_PRIORITY_SITE***

| Parameter | Description |
|---|---|
| gname | The replicated object group with which the site priority group is associated. |
| name | The name of the site priority group whose member you are altering. |
| old_site | The current global database name of the site to dissociate from the priority level. |
| new_site | The new global database name that you want to associate with the current priority level. |

***Table 9–94 Exceptions for ALTER_SITE_PRIORITY_SITE***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingrepgroup | The given replicated object group does not exist. |
| missingpriority | The given site priority group does not exist. |
| missingvalue | The old site is not a group member. |

***Table 9–94 Exceptions for ALTER_SITE_PRIORITY_SITE***

| Exception | Description |
|---|---|
| notquiesced | The replicated object group is not quiesced |

# DBMS_REPCAT.ALTER_SNAPSHOT_PROPAGATION

## Purpose

To alter the propagation method for a given object group at the current snapshot site. This procedure pushes the deferred transaction queue at the snapshot site, locks the snapshot base tables, and regenerates any triggers and their associated packages. You must call this procedure from the snapshot site.

## Syntax

The parameters for the ALTER_SNAPSHOT_PROPAGATION procedure are described in Table 9–95, and the exceptions are listed in Table 9–96. The syntax for this procedure is shown below:

```
DBMS_REPCAT.ALTER_SNAPSHOT_PROPAGATION(
      gname                IN  VARCHAR2,
      propagation_mode     IN  VARCHAR2,
      comment              IN  VARCHAR2   := '')
```

***Table 9–95 Parameters for ALTER_SNAPSHOT_PROPAGATION***

| Parameter | Description |
|---|---|
| gname | The name of the object group for which to alter propagation mode. |
| propagation_mode | The manner in which changes from the current snapshot site are propagated to its associated master site. Appropriate values are SYNCHRONOUS and ASYNCHRONOUS. |
| comment | This comment is added to the RepProp view. |

***Table 9–96 Exceptions for ALTER_SNAPSHOT_PROPAGATION***

| Exception | Description |
|---|---|
| notcompat | Only databases operating in 7.3.0 or later mode can use this procedure. |
| missingrepgroup | The given replicated object group does not exist. |
| typefailure | The propagation mode was specified incorrectly. |
| nonsnapshot | The current site is not a snapshot site for the given object group. |
| commfailure | Cannot contact master. |

# DBMS_REPCAT.CANCEL_STATISTICS

## Purpose

To stop collecting statistics about the successful resolution of update, uniqueness, and delete conflicts for a table.

## Syntax

The parameters for the CANCEL_STATISTICS procedure are described in Table 9–97, and the exceptions are listed in Table 9–98. The syntax for this procedure is shown below:

```
DBMS_REPCAT.CANCEL_STATISTICS(
      sname     IN   VARCHAR2,
      oname     IN   VARCHAR2)
```

***Table 9–97 Parameters for CANCEL_STATISTICS***

| Parameter | Description |
|---|---|
| sname | The name of the schema in which the table is located. |
| oname | The name of the table for which you do not want to gather conflict resolution statistics. |

***Table 9–98 Exceptions for CANCEL_STATISTICS***

| Exception | Description |
|---|---|
| missingschema | The given schema does not exist. |
| missingobject | The given table does not exist. |
| statnotreg | The given table is not currently registered to collect statistics. |

# DBMS_REPCAT.COMMENT_ON_COLUMN_GROUP

## Purpose

To update the comment field in the RepColumn_Group view for a column group. This comment is not added at all master sites until the next call to DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT.

## Syntax

The parameters for the COMMENT_ON_COLUMN_GROUP procedure are described in Table 9–99, and the exceptions are listed in Table 9–100. The syntax for this procedure is shown below:

```
DBMS_REPCAT.COMMENT_ON_COLUMN_GROUP(
      sname           IN   VARCHAR2,
      oname           IN   VARCHAR2,
      column_group    IN   VARCHAR2,
      comment         IN   VARCHAR2)
```

***Table 9–99 Parameters for COMMENT_ON_COLUMN_GROUP***

| Parameter | Description |
|---|---|
| sname | The name of the schema in which the object is located. |
| oname | The name of the replicated table with which the column group is associated. |
| column_group | The name of the column group. |
| comment | The text of the updated comment that you want included in the GROUP_COMMENT field of the RepColumn_Group view. |

***Table 9–100 Exceptions for COMMENT_ON_COLUMN_GROUP***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| missinggroup | The given column group does not exist. |
| missingobj | The object is missing. |

# DBMS_REPCAT.COMMENT_ON_PRIORITY_GROUP/

# DBMS_REPCAT.COMMENT_ON_SITE_PRIORITY

## Purpose

COMMENT_ON_PRIORITY_GROUP updates the comment field in the REPPRIORITY_GROUP view for a priority group. This comment is not added at all master sites until the next call to DBMS_REPCAT.GENERATE_ REPLICATION_SUPPORT.

COMMENT_ON_SITE_PRIORITY updates the comment field in the REPPRIORITY_GROUP view for a site priority group. This procedure is a wrapper for the COMMENT_ON_COLUMN_GROUP procedure and is provided as a convenience only. This procedure must be issued at the master definition site.

## Syntax

The parameters for the COMMENT_ON_PRIORITY_GROUP and COMMENT_ON_SITE_PRIORITY procedures are described in Table 9–101, and the exceptions are listed in Table 9–102.

The syntax for the COMMENT_ON_PRIORITY_GROUP procedure is shown below:

```
DBMS_REPCAT.COMMENT_ON_PRIORITY_GROUP(
      gname       IN   VARCHAR2,
      pgroup      IN   VARCHAR2,
      comment     IN   VARCHAR2)
```

The syntax for the COMMENT_ON_SITE_PRIORITY procedure is shown below:

```
DBMS_REPCAT.COMMENT_ON_SITE_PRIORITY(
      gname       IN   VARCHAR2,
      name        IN   VARCHAR2,
      comment     IN   VARCHAR2)
```

***Table 9–101 Parameters for COMMENT_ON_PRIORITY_GROUP and COMMENT_ON_SITE_PRIORITY***

| Parameter | Description |
|---|---|
| gname | The name of the replicated object group. |
| pgroup/name | The name of the priority or site priority group. |
| comment | The text of the updated comment that you want included in the PRIORITY_COMMENT field of the RepPriority_Group view. |

***Table 9–102 Exceptions for COMMENT_ON_PRIORITY_GROUP and COMMENT_ON_SITE_PRIORITY***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| missingrepgroup | The given replicated object group does not exist. |
| missingprioritygroup | The given priority group does not exist. |

# DBMS_REPCAT.COMMENT_ON_REPGROUP

## Purpose

To update the comment field in the REPGROUP view for a replicated object group. This procedure must be issued at the master definition site.

## Syntax

The parameters for the COMMENT_ON_REPGROUP procedure are described in Table 9–103, and the exceptions are listed in Table 9–104. The syntax for this procedure is shown below:

```
DBMS_REPCAT.COMMENT_ON_REPGROUP(
      gname      IN   VARCHAR2,
      comment    IN   VARCHAR2)
```

***Table 9–103 Parameters for COMMENT_ON_REPGROUP***

| Parameter | Description |
|---|---|
| gname | The name of the object group that you want to comment on. |
| comment | The updated comment to include in the SCHEMA_COMMENT field of the RepGroup view. |

***Table 9–104 Exceptions for COMMENT_ON_REPGROUP***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| commfailure | At least one master site is not accessible. |

# DBMS_REPCAT.COMMENT_ON_REPSITES

## Purpose

To update the comment field in the RepSite view for a replicated site. This procedure must be issued at the master definition site.

## Syntax

The parameters for the COMMENT_ON_REPSITES procedure are described in Table 9–105, and the exceptions are listed in Table 9–106. The syntax for this procedure is shown below:

```
DBMS_REPCAT.COMMENT_ON_REPSITES(
      gname       IN   VARCHAR2,
      [ master    IN   VARCHAR,]
      comment     IN   VARCHAR2)
```

***Table 9–105 Parameters for COMMENT_ON_REPSITES***

| Parameter | Description |
|---|---|
| gname | The name of the object group. This avoids confusion if a database is a master site in more than one replicated environment. |
| master | Optional; the fully qualified database name of the master site that you want to comment on. To update comments at a snapshot site, omit this parameter. |
| comment | The text of the updated comment that you want to include in the MASTER_COMMENT field of the RepSites view. |

***Table 9–106 Exceptions for COMMENT_ON_REPSITESS***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| nonmaster | The invocation site is not a master site. |
| commfailure | At least one master site is not accessible. |

# DBMS_REPCAT.COMMENT_ON_REPOBJECT

## Purpose

To update the comment field in the RepObject view for a replicated object. This procedure must be issued at the master definition site.

## Syntax

The parameters for the COMMENT_ON_REPOBJECT procedure are described in Table 9–107, and the exceptions are listed in Table 9–108. The syntax for this procedure is shown below:

```
DBMS_REPCAT.COMMENT_ON_REPOBJECT(
      sname    IN   VARCHAR2,
      oname    IN   VARCHAR2,
      type     IN   VARCHAR2,
      comment  IN   VARCHAR2)
```

***Table 9–107 Parameters for COMMENT_ON_REPOBJECT***

| Parameter | Description |
|---|---|
| sname | The name of the schema in which the object is located. |
| oname | The name of the object that you want to comment on. |
| type | The type of the object. |
| comment | The text of the updated comment that you want to include in the OBJECT_COMMENT field of the RepObject view. |

***Table 9–108 Exceptions for COMMENT_ON_REPOBJECT***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| missingobject | The given object does not exist. |
| typefailure | The given type parameter is not supported. |
| commfailure | At least one master site is not accessible. |

# DBMS_REPCAT.COMMENT_ON_*conflicttype*_RESOLUTION

## Purpose

To update the comment field in the RepResolution view for a conflict resolution routine. The procedure that you need to call is determined by the type of conflict that the routine resolves. These procedures must be issued at the master definition site.

| Conflict Type | Procedure Name |
|---|---|
| update | COMMENT_ON_UPDATE_RESOLUTION |
| uniqueness | COMMENT_ON_UNIQUE_RESOLUTION |
| delete | COMMENT_ON_DELETE_RESOLUTION |

The comment is not added at all master sites until the next call to DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT.

## Syntax

The parameters for the COMMENT_ON_*conflicttype*_RESOLUTION procedures are described in Table 9–109, and the exceptions are listed in Table 9–110.

The syntax for the COMMENT_ON_UPDATE_RESOLUTION procedure is shown below:

```
DBMS_REPCAT.COMMENT_ON_UPDATE_RESOLUTION(
      sname                 IN   VARCHAR2,
      oname                 IN   VARCHAR2,
      column_group          IN   VARCHAR2,
      sequence_no           IN   NUMBER,
      comment               IN   VARCHAR2)
```

The syntax for the COMMENT_ON_UNIQUE_RESOLUTION procedure is shown below:

```
DBMS_REPCAT.COMMENT_ON_UNIQUE_RESOLUTION(
      sname                 IN   VARCHAR2,
      oname                 IN   VARCHAR2,
      constraint_name       IN   VARCHAR2,
      sequence_no           IN   NUMBER,
      comment               IN   VARCHAR2)
```

The syntax for the COMMENT_ON_DELETE_RESOLUTION procedure is shown below:

```
DBMS_REPCAT.COMMENT_ON_DELETE_RESOLUTION(
      sname                 IN   VARCHAR2,
      oname                 IN   VARCHAR2,
      sequence_no           IN   NUMBER,
      comment               IN   VARCHAR2)
```

***Table 9–109 Parameters for COMMENT_ON_conflicttype_RESOLUTION***

| Parameter | Description |
|---|---|
| sname | The name of the schema. |
| oname | The name of the replicated table with which the conflict resolution routine is associated. |
| column_group | The name of the column group with which the update conflict resolution routine is associated. |
| constraint_name | The name of the unique constraint with which the uniqueness conflict resolution routine is associated. |
| sequence_no | The sequence number of the conflict resolution procedure. |
| comment | The text of the updated comment that you want included in the RESOLUTION_COMMENT field of the RepResolution view. |

***Table 9–110 Exceptions for COMMENT_ON_conflicttype_RESOLUTION***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| missingobject | The given object does not exist. |
| missingresolution | SEQUENCE_NO or COLUMN_GROUP is not registered. |

# DBMS_REPCAT.CREATE_MASTER_REPGROUP

## Purpose

To create a new, empty, quiesced master replication object group.

## Syntax

The parameters for the CREATE_MASTER_REPGROUP procedure are described in Table 9–111, and the exceptions are listed in Table 9–112. The syntax for this procedure is shown below:

```
DBMS_REPCAT.CREATE_MASTER_REPGROUP(
      gname            IN   VARCHAR2,
      group_comment    IN   VARCHAR2     := '',
      master_comment   IN   VARCHAR2     := ''),
      qualifier        IN   VARCHAR2     := '')
```

***Table 9–111 Parameters for CREATE_MASTER_REPGROUP***

| Parameter | Description |
|---|---|
| gname | The name of the object group that you want to create. |
| group_comment | This comment is added to the RepCat view. |
| master_comment | This comment is added to the RepGroup view. |
| qualifier | Connection qualifier for object group. Be sure to use the @ sign, as shown in the example: See “Using Connection Qualifiers for a Master Group” on page 3-16. |

***Table 9–112 Exceptions for CREATE_MASTER_REPGROUP***

| Exception | Description |
|---|---|
| duplicaterepgroup | The object group already exists. |
| ddlfailure | There is a problem creating the rep$what_am_i package or package body. |
| norepopt | The advanced replication option is not installed. |
| missingrepgrp | The object group name was not specified. |
| qualifiertoolong | Connection qualifier is too long. |

# DBMS_REPCAT.CREATE_MASTER_REPOBJECT

## Purpose

To indicate that an object is a replicated object.

## Syntax

The parameters for the CREATE_MASTER_REPOBJECT procedure are shown in Table 9–113, and the exceptions are listed in Table 9–114. The syntax for this procedure is shown below:

```
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
      sname                 IN   VARCHAR2,
      oname                 IN   VARCHAR2,
      type                  IN   VARCHAR2,
      use_existing_object   IN   BOOLEAN     := TRUE,
      ddl_text              IN   VARCHAR2    := NULL,
      comment               IN   VARCHAR2    := '',
      retry                 IN   BOOLEAN     := FALSE
      copy_rows             IN   BOOLEAN     := TRUE,
      gname                 IN   VARCHAR2    := '')
```

**Note:** If the DDL is supplied without specifying a schema, the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema.

***Table 9–113 Parameters for CREATE_MASTER_REPOBJECT***

| Parameters | Description |
|---|---|
| sname | The name of the schema in which the object that you want to replicate is located. |
| oname | The name of the object you are replicating. If DDL_TEXT is NULL, this object must already exist in the given schema. To ensure uniqueness, table names should be a maximum of 27 bytes long, and packages should be no more than 24 bytes. |
| type | The type of the object that you are replicating. The types supported are: TABLE, INDEX, SYNONYM, TRIGGER, VIEW, PROCEDURE, FUNCTION, PACKAGE, and PACKAGE BODY. |

***Table 9–113 Parameters for CREATE_MASTER_REPOBJECT***

| Parameters | Description |
|---|---|
| use_existing_object | Indicate TRUE if you want to reuse any objects of the same type and shape at the current master sites. See Table 9–116 for more information on how these changes are applied. |
| ddl_text | If the object does not already exist at the master definition site, you must supply the DDL text necessary to create this object. PL/SQL packages, package bodies, procedures, and functions must have a trailing semicolon. SQL statements do not end with trailing semicolon. Oracle does not parse this DDL before applying it; therefore, you must ensure that your DDL text provides the appropriate schema and object name for the object being created. |
| comment | This comment will be added to the OBJECT_COMMENT field of the RepObject view. |
| retry | Indicate TRUE if you want Oracle to reattempt to create an object that it was previously unable to create. Use RETRY if the error was transient or has since been rectified; for example, if you previously had insufficient resources. If RETRY is TRUE, Oracle creates the object only at master sites whose object status is not VALID. |
| copy_rows | Indicate TRUE if you want the initial contents of a newly replicated object to match the contents of the object at the master definition site. See Table 9–116 for more information. |
| gname | The name of the object group in which you want to create the replicated object. The schema name is used as the default object group name if none is specified. |

***Table 9–114 Exceptions for CREATE_MASTER_REPOBJECT***

| Exceptions | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| notquiesced | The replicated object group has not been suspended. |
| duplicateobject | The given object already exists in the replicated object group and retry is FALSE, or if a name conflict occurs. |
| missingobject | The object identified by SNAME and ONAME does not exist and appropriate DDL has not been provided. |
| typefailure | Objects of the given type cannot be replicated. |
| ddlfailure | DDL at the master definition site did not succeed. |
| commfailure | At least one master site is not accessible. |
| notcompat | Not all remote masters in 7.3 compatibility mode. |

***Table 9–115 Object Creation at Master Sites***

| Object Already Exists? | COPY_ROWS | USE_EXISTING_OBJECTS | Result |
|---|---|---|---|
| yes | TRUE | TRUE | *duplicatedobject* message if objects do not match. For tables, use data from master definition site. |
| yes | FALSE | TRUE | *duplicatedobject* message if objects do not match. For tables, Admin must ensure contents are identical. |
| yes | TRUE / FALSE | FALSE | *duplicatedobject* message. |
| no | TRUE | TRUE / FALSE | Object is created. Tables populated using data from master definition site. |
| no | FALSE | TRUE / FALSE | Object is created. DBA must populate tables and ensure consistency of tables at all sites. |

# DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP

## Purpose

To create a new, empty snapshot replication object group in your local database.

## Syntax

The parameters for the CREATE_SNAPSHOT_REPGROUP procedure are described in Table 9–116, and the procedures are listed in Table 9–117. The syntax for this procedure is shown below:

```
DBMS_REPCAT.CREATE_SNAPSHOT_REPGROUP(
      gname              IN   VARCHAR2,
      master             IN   VARCHAR2,
      comment            IN   VARCHAR2     := '',
      propagation_mode   IN   VARCHAR2     := 'ASYNCHRONOUS')
```

**Note:** CREATE_SNAPSHOT_REPGROUP automatically calls DBMS_REPCAT.REGISTER_SNAPSHOT_REPGROUP, but ignores any errors that may have happened during registration.

***Table 9–116 Parameters for CREATE_SNAPSHOT_REPGROUP***

| Parameter | Description |
|---|---|
| gname | The name of the replicated object group. This object group must exist at the given master site. |
| master | The fully qualified database name of the database in the replicated environment to use as the master. |
| comment | This comment is added to the GROUP_COMMENT field of the RepCat view. |
| propagation_mode | The method of propagation for all updatable snapshots in the object group. Acceptable values are SYNCHRONOUS and ASYNCHRONOUS. |

***Table 9–117 Exceptions for CREATE_SNAPSHOT_REPGROUP***

| Exception | Description |
|---|---|
| duplicaterepgroup | The object group already exists at the invocation site. |
| nonmaster | The given database is not a master site. |
| commfailure | The given database is not accessible. |
| norepopt | The advanced replication option is not installed. |
| typefailure | The propagation mode was specified incorrectly. |
| missingrepgroup | If replicated object group not at master site. |
| notcompatible | Must be 7.3 compatible. |

# DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT

## Purpose

To add a replicated object to your snapshot site.

## Syntax

The parameters for the CREATE_SNAPSHOT_REPOBJECT procedure are shown in Table 9–118, and the exceptions are listed in Table 9–119. The syntax for this procedure is shown below:

```
DBMS_REPCAT.CREATE_SNAPSHOT_REPOBJECT(
     sname          IN   VARCHAR2,
     oname          IN   VARCHAR2,
     type           IN   VARCHAR2,
     ddl_text       IN   VARCHAR2  := '',
     comment        IN   VARCHAR2  := '',
     gname          IN   VARCHAR2  := '',
     gen_objs_owner  IN   VARCHAR2 := '',
     min_communication IN BOOLEAN  := TRUE )
```

**Note**: If the DDL is supplied without specifying a schema, the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema.

***Table 9–118 Parameters for CREATE_SNAPSHOT_REPOBJECT***

| Parameter | Description |
|---|---|
| sname | The name of the schema in which the object is located. |
| oname | The name of the object that you want to add to the replicated snapshot object group. ONAME must exist at the associated master site. |
| type | The type of the object that you are replicating. The types supported for snapshot sites are: PACKAGE, PACKAGE BODY, PROCEDURE, FUNCTION, SNAPSHOT, SYNONYM, and VIEW. |
| ddl_text | For objects of type SNAPSHOT, the DDL text needed to create the object; for other types, use the default, '' (an empty string). If a snapshot with the same name already exists, Oracle ignores the DDL and registers the existing snapshot as a replicated object. If the master table for a snapshot does not exist in the replicated object group of the master site designated for this schema, Oracle raises a *missingobject error.* |
| comment | This comment is added to the OBJECT_COMMENT field of the RepObject view. |
| gname | The name of the replicated object group to which you are adding an object. The schema name is used as the default group name if none is specified. |
| gen_objs_owner | The name of the user you want to assign as owner of the transaction. |
| min_communication | Set to FALSE if any master site is running Oracle7 release 7.3. Set to TRUE to minimize new and old values of propagation. The default is TRUE. For more information, see "Minimizing Data Propagation for Update Conflict Resolution" on page 5-40. |

***Table 9–119 Exceptions for CREATE_SNAPSHOT_REPOBJECT***

| Exception | Description |
|---|---|
| nonsnapshot | The invocation site is not a snapshot site. |
| nonmaster | The master is no longer a master site. |
| missingobject | The given object does not exist in the master's replicated object group. |
| duplicateobject | The given object already exists with a different shape. |
| typefailure | The type is not an allowable type. |
| ddlfailure | The DDL did not succeed. |
| commfailure | The master site is not accessible. |
| missingschema | The schema does not exist as a database schema. |
| badsnapddl | DDL was executed but snapshot does not exist. |
| onlyonesnap | Only one snapshot for master table can be created. |
| badsnapname | Snapshot base table differs from master table. |
| missingrepgroup | Replicated object group does not exist. |

# DBMS_REPCAT.DEFINE_COLUMN_GROUP

## Purpose

To create an empty column group. You must call this procedure from the master definition site. For more information, see "Update Conflict Resolution and Column Groups" on page 5-7.

## Syntax

The parameters for the DEFINE_COLUMN_GROUP procedure are described in Table 9–120, and the exceptions are listed in Table 9–121. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DEFINE_COLUMN_GROUP(
     sname            IN   VARCHAR2,
     oname            IN   VARCHAR2,
     column_group     IN   VARCHAR2,
     comment          IN   VARCHAR2 := NULL)
```

***Table 9–120 Parameters for DEFINE_COLUMN_GROUP***

| Parameter | Description |
|---|---|
| sname | The schema in which the replicated table is located. |
| oname | The name of the replicated table for which you are creating a column group. |
| column_group | The name of the column group that you want to create. |
| comment | This user text is displayed in the RepColumnGroup view. |

***Table 9–121 Exceptions for DEFINE_COLUMN_GROUP***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingobject | The given table does not exist. |
| duplicategroup | The given column group already exists for the table. |
| notquiesced | The object group that the given table belongs to is not quiesced. |

# DBMS_REPCAT.DEFINE_PRIORITY_GROUP

## Purpose

To create a new priority group for a replicated object group. You must call this procedure from the master definition site. See "Priority Groups and Site Priority" on page 5-20.

## Syntax

The parameters for the DEFINE_PRIORITY_GROUP procedure are described in Table 9–122, and the exceptions are listed in Table 9–123. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DEFINE_PRIORITY_GROUP(
     gname          IN   VARCHAR2,
     pgroup         IN   VARCHAR2,
     datatype       IN   VARCHAR2,
     fixed_length   IN   INTEGER := NULL,
     comment        IN   VARCHAR2 := NULL)
```

***Table 9–122 Parameters for DEFINE_PRIORITY_GROUP***

| Parameter | Description |
|---|---|
| gname | The replicated object group for which you are creating a priority group. |
| pgroup | The name of the priority group that you are creating. |
| datatype | The datatype of the priority group members. The datatypes supported are: CHAR, VARCHAR2, NUMBER, DATE, RAW, NCHAR, and NVARCHAR2. |
| fixed_length | You must provide a column length for the CHAR datatype. All other types can use the default, NULL. |
| comment | This user comment is added to the RepPriority view. |

***Table 9–123 Exceptions for DEFINE_PRIORITY_GROUP***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingrepgroup | The given replicated object group does not exist. |
| duplicatepriritygroup | The given priority group already exists in the replicated object group. |
| typefailure | The given datatype is not supported. |
| notquiesced | The replicated object group is not quiesced. |

# DBMS_REPCAT.DEFINE_SITE_PRIORITY

## Purpose

To create a new site priority group for a replicated object group. You must call this procedure from the master definition site. See "Priority Groups and Site Priority" on page 5-20.

## Syntax

The parameters for the DEFINE_SITE_PRIORITY procedure are described in Table 9–124, and the exceptions are listed in Table 9–125. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DEFINE_SITE_PRIORITY(
      gname          IN   VARCHAR2,
      name           IN   VARCHAR2,
      comment        IN   VARCHAR2 := NULL)
```

***Table 9–124 Parameters for DEFINE_SITE_PRIORITY***

| Parameter | Description |
|---|---|
| gname | The replicated object group for which you are creating a site priority group. |
| name | The name of the site priority group that you are creating. |
| comment | This user comment is added to the RepPriority view. |

***Table 9–125 Exceptions for DEFINE_SITE_PRIORITY***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingrepgroup | The given replicated object group does not exist. |
| duplicateprioritygroup | The given site priority group already exists in the replicated object group. |
| notquiesced | The replicated object group is not quiesced. |

# DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN

## Purpose

To execute the local outstanding deferred administrative procedures for the given replicated object group at the current master site, or (with assistance from job queues) for all master sites.

**Note**: DO_DEFERRED_REPCAT_ADMIN executes only those administrative requests submitted by the connected user that called DO_DEFERRED_ REPCAT_ADMIN. Requests submitted by other users are ignored.

## Syntax

The parameters for the DO_DEFERRED_REPCAT_ADMIN procedure are described in Table 9–126, and the exceptions are listed in Table 9–127. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN(
      gname          IN   VARCHAR2,
      all_sites      IN   BOOLEAN := FALSE)
```

***Table 9–126 Parameters for DO_DEFERRED_REPCAT_ADMIN***

| Parameter | Description |
|---|---|
| gname | The name of the replicated object group. |
| all_sites | If ALL_SITES is TRUE, use a job to execute the local administrative procedures at each master. |

***Table 9–127 Exceptions for DO_DEFERRED_REPCAT_ADMIN***

| Exception | Description |
|---|---|
| nonmaster | The invocation site is not a master site. |
| commfailure | At least one master site is not accessible and all_sites is TRUE. |

# DBMS_REPCAT.DROP_COLUMN_GROUP

## Purpose

To drop a column group. You must call this procedure from the master definition site. See "Using Priority Groups for Update Conflict Resolution" on page 5-22.

## Syntax

The parameters for the DROP_COLUMN_GROUP procedure are described in Table 9–128, and the exceptions are listed in Table 9–129. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DROP_COLUMN_GROUP(
      sname        IN   VARCHAR2,
      oname        IN   VARCHAR2,
      column_group IN   VARCHAR2)
```

***Table 9–128 Parameters for DROP_COLUMN_GROUP***

| Parameter | Description |
|---|---|
| sname | The schema in which the replicated table is located. |
| oname | The name of the replicated table whose column group you are dropping. |
| column_group | The name of the column group that you want to drop. |

***Table 9–129 Exceptions for DROP_COLUMN_GROUP***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| referenced | The given column group is being used in conflict detection and resolution. |
| missingobject | The given table does not exist. |
| missinggroup | The given column group does not exist. |
| notquiesced | The replicated object group that the table belongs to is not quiesced. |

# DBMS_REPCAT.DROP_GROUPED_COLUMN

## Purpose

To remove members from a column group. You must call this procedure from the master definition site. For more information, see "Adding and Removing Columns in a Column Group" on page 5-14.

## Syntax

The parameters for the DROP_GROUPED_COLUMN procedure are described in Table 9–130, and the exceptions are listed in Table 9–131. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DROP_GROUPED_COLUMN(
      sname                 IN   VARCHAR2,
      oname                 IN   VARCHAR2,
      column_group          IN   VARCHAR2,
      list_of_column_names  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S)
```

***Table 9–130 Parameters for DROP_GROUPED_COLUMN***

| Parameter | Description |
|---|---|
| sname | The schema in which the replicated table is located. |
| oname | The name of the replicated table in which the column group is located. |
| column_group | The name of the column group from which you are removing members. |
| list_of_column_names | The names of the columns that you are removing from the designated column group. This can either be a comma-separated list or a PL/SQL table of column names. The PL/SQL table must be of type dbms_repcat.varchar2s. |

***Table 9–131 Exceptions for DROP_GROUPED_COLUMN***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingobject | The given table does not exist. |

***Table 9–131 Exceptions for DROP_GROUPED_COLUMN***

| Exception | Description |
|---|---|
| notquiesced | The replicated object group that the table belongs to is not quiesced. |

# DBMS_REPCAT.DROP_MASTER_REPGROUP

## Purpose

To drop a replicated object group from your current site. To drop the replicated object group from all master sites, including the master definition site, you can call this procedure at the master definition site, and set the final argument to TRUE.

## Syntax

The parameters for the DROP_MASTER_REPGROUP procedure are described in Table 9–132, and the exceptions are listed in Table 9–133. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DROP_MASTER_REPGROUP(
      gname             IN VARCHAR2,
      drop_contents     IN BOOLEAN    := FALSE,
      all_sites         IN BOOLEAN     := FALSE)
```

***Table 9–132 Parameters for DROP_MASTER_REPGROUP***

| Parameter | Description |
|---|---|
| gname | The name of the replicated object group that you want to drop from the current master site. |
| drop_contents | By default, when you drop the object group at a master site, all of the objects remain in the schema. They simply are no longer replicated; that is, the replicated objects in the object group no longer send changes to, or receive changes from, other master sites. If you set this argument to TRUE, any replicated objects in the replicated object group are dropped from their associated schemas. |
| all_sites | If ALL_SITES is TRUE and the invocation site is the master definition site, the procedure synchronously multicasts the request to all masters. In this case, execution is immediate at the master definition site and may be deferred at all other master sites. |

***Table 9–133 Exceptions for DROP_MASTER_REPGROUP***

| Exception | Description |
|---|---|
| nonmaster | The invocation site is not a master site. |
| nonmasterdef | The invocation site is not the master definition site and ALL_SITES is TRUE. |
| commfailure | At least one master site is not accessible and ALL_SITES is TRUE. |
| fullqueue | The deferred RPC queue has entries for the replicated object group. |
| masternotremoved | Master does not recognize the masterdef. |

# DBMS_REPCAT.DROP_MASTER_REPOBJECT

## Purpose

To drop a replicated object from a replicated object group. You must call this procedure from the master definition site.

## Syntax

The parameters for the DROP_MASTER_REPOBJECT procedure are described in Table 9–134, and the exceptions are listed in Table 9–135. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DROP_MASTER_REPOBJECT(
      sname          IN   VARCHAR2,
      oname          IN   VARCHAR2,
      type           IN   VARCHAR2,
      drop_objects   IN   BOOLEAN     := FALSE)
```

***Table 9–134 Parameters for DROP_MASTER_REPOBJECT***

| Parameter | Description |
|---|---|
| sname | The name of the schema in which the object is located. |
| oname | The name of the object that you want to remove from the replicated object group. |
| type | The type of object that you want to drop. |
| drop_objects | By default, the object remains in the schema, but is dropped from the replicated object group; that is, any changes to the object are no longer replicated to other master and snapshot sites. To completely remove the object from the replicated environment, set this argument to TRUE. |

***Table 9–135 Exceptions for DROP_MASTER_REPOBJECT***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| missingobject | The given object does not exist. |
| typefailure | The given type parameter is not supported. |
| commfailure | At least one master site is not accessible. |

# DBMS_REPCAT.DROP_PRIORITY

## Purpose

To drop a member of a priority group by priority level. You must call this procedure from the master definition site. See "Dropping a Member by Priority" on page 5-27.

## Syntax

The parameters for the DROP_PRIORITY procedure are described in Table 9–136, and the exceptions are listed in Table 9–137. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DROP_PRIORITY(
      gname          IN   VARCHAR2,
      pgroup         IN   VARCHAR2,
      priority_num   IN   NUMBER)
```

***Table 9–136 Parameters for DROP_PRIORITY***

| Parameter | Description |
|---|---|
| gname | The replicated object group with which the priority group is associated. |
| pgroup | The name of the priority group containing the member that you want to drop. |
| priority_num | The priority level of the priority group member that you want to remove from the group. |

***Table 9–137 Exceptions for DROP_PRIORITY***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingrepgroup | The given replicated object group does not exist. |
| missingprioritygroup | The given priority group does not exist. |
| notquiesced | The replicated object group is not quiesced. |

# DBMS_REPCAT.DROP_PRIORITY_GROUP

## Purpose

To drop a priority group for a given replicated object group. You must call this procedure from the master definition site. See "Dropping a Priority Group" on page 5-27.

## Syntax

The parameters for the DROP_PRIORITY_GROUP procedure are described in Table 9–138, and the exceptions are listed in Table 9–139. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DROP_PRIORITY_GROUP(
      gname      IN   VARCHAR2,
      pgroup     IN   VARCHAR2)
```

***Table 9–138 Parameters for DROP_PRIORITY_GROUP***

| Parameter | Description |
|---|---|
| gname | The replicated object group with which the priority group is associated. |
| pgroup | The name of the priority group that you want to drop. |

***Table 9–139 Exceptions for DROP_PRIORITY_GROUP***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingrepgroup | The given replicated object group does not exist. |
| referenced | The given priority group is being used in conflict resolution. |
| notquiesced | The given replicated object group is not quiesced. |

# DBMS_REPCAT.DROP_PRIORITY_*datatype*

## Purpose

To drop a member of a priority group by value. You must call this procedure from the master definition site. The procedure that you must call is determined by the datatype of your "priority" column. See "Dropping a Member by Value" on page 5-26.

## Syntax

The parameters for the DROP_PRIORITY_*datatype* procedure are described in Table 9–140, and the exceptions are listed in Table 9–141. The syntax for the DROP_PRIORITY_*datatype* procedure is shown below.

```
DBMS_REPCAT.DROP_PRIORITY_datatype(
      gname    IN   VARCHAR2,
      pgroup   IN   VARCHAR2,
      value    IN   datatype)
```

where *datatype*:

```
{ NUMBER
| VARCHAR2
| CHAR
| DATE
| RAW
| NCHAR
| NVARCHAR2 }
```

***Table 9–140 Parameters for DROP_PRIORITY_datatype***

| Parameter | Description |
|---|---|
| gname | The replicated object group with which the priority group is associated. |
| pgroup | The name of the priority group containing the member that you want to drop. |
| value | The value of the priority group member that you want to remove from the group. |

***Table 9–141 Exceptions for DROP_PRIORITY_datatype***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingrepgroup | The given replicated object group does not exist. |
| missingprioritygroup | The given priority group does not exist. |
| paramtype, typefailure | The value has the incorrect datatype for the priority group. |
| notquiesced | The given replicated object group is not quiesced |

# DBMS_REPCAT.DROP_SITE_PRIORITY

## Purpose

To drop a site priority group for a given replicated object group. You must call this procedure from the master definition site. See "Dropping a Site Priority Group" on page 5-30.

## Syntax

The parameters for the DROP_SITE_PRIORITY procedure are described in Table 9–142, and the exceptions are listed in Table 9–143. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DROP_SITE_PRIORITY(
      gname       IN   VARCHAR2,
      name        IN   VARCHAR2)
```

***Table 9–142 Parameters for DROP_SITE_PRIORITY***

| Parameter | Description |
|---|---|
| gname | The replicated object group with which the site priority group is associated. |
| name | The name of the site priority group that you want to drop. |

***Table 9–143 Exceptions for DROP_SITE_PRIORITY***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingrepgroup | The given replicated object group does not exist. |
| referenced | The given site priority group is being used in conflict resolution. |
| notquiesced | The given replicated object group is not quiesced |

# DBMS_REPCAT.DROP_SITE_PRIORITY_SITE

## Purpose

To drop a given site, by name, from a site priority group. You must call this procedure from the master definition site. See "Dropping a Site by Site Name" on page 5-30.

## Syntax

The parameters for the DROP_SITE_PRIORITY_SITE procedure are described in Table 9–144, and the exceptions are listed in Table 9–145. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DROP_SITE_PRIORITY_SITE(
      gname       IN   VARCHAR2,
      name        IN   VARCHAR2,
      site        IN   VARCHAR2)
```

***Table 9–144 Parameters for DROP_SITE_PRIORITY_SITE***

| Parameter | Description |
|---|---|
| gname | The replicated object group with which the site priority group is associated. |
| name | The name of the site priority group whose member you are dropping. |
| site | The global database name of the site you are removing from the group. |

***Table 9–145 Exceptions for DROP_SITE_PRIORITY_SITE***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingrepgroup | The given replicated object group does not exist. |
| missingpriority | The given site priority group does not exist. |
| missingsite | The given site does not exist. |
| notquiesced | The given replicated object group is not quiesced. |

# DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP

## Purpose

To drop a snapshot site from your replicated environment.

## Syntax

The parameters for the DROP_SNAPSHOT_REPGROUP procedure are described in Table 9–146, and the exceptions are listed in Table 9–147. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP(
    gname                IN   VARCHAR2,
    drop_contents        IN   BOOLEAN   := FALSE)
```

**Note:** DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP automatically calls DBMS_REPCAT.REGISTER_SNAPSHOT_ REPGROUP to unregister the snapshot, but ignores any errors that may have occurred during unregistration.

***Table 9–146 Parameters for DROP_SNAPSHOT_REPGROUP***

| Parameter | Description |
|---|---|
| gname | The name of the replicated object group that you want to drop from the current snapshot site. All objects generated to support replication, such as triggers and packages, are dropped. |
| drop_contents | By default, when you drop the replicated object group at a snapshot site, all of the objects remain in their associated schemas; they simply are no longer replicated. If you set this argument to TRUE, any replicated objects in the replicated object group are dropped from their schemas. |

***Table 9–147 Exceptions for DROP_SNAPSHOT_REPGROUP***

| Exception | Description |
|---|---|
| nonsnapshot | The invocation site is not a snapshot site. |
| missrepgrp | The specified object group does not exist. |

# DBMS_REPCAT.DROP_SNAPSHOT_REPOBJECT

## Purpose

To drop a replicated object from a snapshot site.

## Syntax

The parameters for the DROP_SNAPSHOT_REPOBJECT procedure are described in Table 9–148, and the exceptions are listed in Table 9–149. The syntax for this procedure is shown below:

```
DBMS_REPCAT.DROP_SNAPSHOT_REPOBJECT(
     sname          IN   VARCHAR2,
     oname          IN   VARCHAR2,
     type           IN   VARCHAR2,
     drop_objects   IN   BOOLEAN  := FALSE)
```

***Table 9–148 Parameters for DROP_SNAPSHOT_REPOBJECT***

| Parameter | Description |
|---|---|
| sname | The name of the schema in which the object is located. |
| oname | The name of the object that you want to drop from the replicated object group. |
| type | The type of the object that you want to drop. |
| drop_objects | By default, the object remains in its associated schema, but is dropped from its associated object group. To completely remove the object from its schema at the current snapshot site, set this argument to TRUE. |

***Table 9–149 Exceptions for DROP_SNAPSHOT_REPOBJECT***

| Exception | Description |
|---|---|
| nonsnapshot | The invocation site is not a snapshot site. |
| missingobject | The given object does not exist. |
| typefailure | The given type parameter is not supported. |

# DBMS_REPCAT.DROP_*conflicttype*_RESOLUTION

## Purpose

To drop an update, delete, or uniqueness conflict resolution routine. You must call these procedures from the master definition site. The procedure that you must call is determined by the type of conflict that the routine resolves.

| Conflict Type | Procedure Name |
|---|---|
| update | DROP_UPDATE_RESOLUTION |
| uniqueness | DROP_UNIQUE_RESOLUTION |
| delete | DROP_DELETE_RESOLUTION |

## Syntax

The parameters for the DROP_*conflicttype*_RESOLUTION procedure are described in Table 9–150, and the exceptions are listed in Table 9–151. The syntax for the DROP_UPDATE_RESOLUTION procedure is shown below:

```
DBMS_REPCAT.DROP_UPDATE_RESOLUTION(
     sname              IN   VARCHAR2,
     oname              IN   VARCHAR2,
     column_group       IN   VARCHAR2,
     sequence_no        IN   NUMBER)
```

The syntax for the DROP_DELETE_RESOLUTION procedure is shown below:

```
DBMS_REPCAT.DROP_DELETE_RESOLUTION(
     sname              IN   VARCHAR2,
     oname              IN   VARCHAR2,
     sequence_no        IN   NUMBER)
```

The syntax for the DROP_UNIQUE_RESOLUTION procedure is shown below:

```
DBMS_REPCAT.DROP_UNIQUE_RESOLUTION(
     sname              IN   VARCHAR2,
     oname              IN   VARCHAR2,
     constraint_name    IN  VARCHAR2,
     sequence_no        IN  NUMBER)
```

***Table 9–150 Parameters for DROP_conflicttype_RESOLUTION***

| Parameter | Description |
|---|---|
| sname | The schema in which the table is located. |
| oname | The name of the table for which you want to drop a conflict resolution routine. |
| column_group | The name of the column group for which you want to drop an update conflict resolution routine. |
| constraint_name | The name of the Unique constraint for which you want to drop a unique conflict resolution routine. |
| sequence_no | The sequence number assigned to the conflict resolution method that you want to drop. This number uniquely identifies the routine. |

***Table 9–151 Exceptions for DROP_conflicttype_RESOLUTION***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| missingobject | The given object does not exist as a table in the given schema, or a conflict resolution routine with the given sequence number is not registered. |
| referenced | The conflict resolution routine is being used in conflict resolution. |
| notquiesced | The replicated object group is not quiesced. |

# DBMS_REPCAT.EXECUTE_DDL

## Purpose

To supply DDL that you want to have executed at each master site. You can call this procedure only from the master definition site.

## Syntax

The parameters for the EXECUTE_DDL procedure are described in Table 9–152, and the exceptions are listed in Table 9–153. The syntax for this procedure is shown below:

```
DBMS_REPCAT.EXECUTE_DDL(
     gname           IN   VARCHAR2,
     { master_list   IN   VARCHAR2     := NULL,
      | master_table  IN   DBMS_UTILITY.DBLINK_ARRAY,}
     DDL_TEXT        IN   VARCHAR2)
```

**Note**: If the DDL is supplied without specifying a schema, the default schema is the replication administrator's schema. Be sure to specify the schema if it is other than the replication administrator's schema. This procedure is overloaded. The MASTER_LIST and MASTER_TABLE parameters are mutually exclusive.

***Table 9–152 Parameters for EXECUTE_DDL***

| Parameter | Description |
|---|---|
| gname | The name of the replicated object group. |
| master_list | A comma-separated list of master sites at which you want to execute the supplied DDL. There must be no extra white space between site names. The default value, NULL, indicates that the DDL should be executed at all sites, including the master definition site. |
| master_table | A table of master sites at which you want to execute the supplied DDL. The first master should be at offset 1, the second at offset 2, and so on. |
| ddl_text | The DDL that you want to have executed at each of the given master sites. |

*Table 9–153 Exceptions for EXECUTE_DDL*

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| nonmaster | At least one site is not a master site. |
| ddlfailure | DDL at the master definition site did not succeed. |
| commfailure | At least one master site is not accessible. |

# DBMS_REPCAT.GENERATE_REPLICATION_PACKAGE

## Purpose

To provide more fine-grained control of replication support generation. Primarily used for environments that include Oracle7 Release 7.3 sites. Generates the packages needed to support replication for a given table at all master sites. You must call this procedure from the master definition site.

## Syntax

The parameters for the GENERATE_REPLICATION_PACKAGE procedure are described in Table 9–154, and the exceptions are listed in Table 9–155. The syntax for this procedure is shown below:

```
DBMS_REPCAT.GENERATE_REPLICATION_PACKAGE(
      sname  IN    VARCHAR2,
      oname  IN    VARCHAR2)
```

***Table 9–154 Parameters for GENERATE_REPLICATION_PACKAGE***

| Parameter | Description |
|---|---|
| sname | The schema in which the table is located. |
| oname | The name of the table for which you are generating replication support. |

***Table 9–155 Exceptions for GENERATE_REPLICATION_PACKAGE***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| missingobject | The given object does not exist as a table in the given schema awaiting row-level replication information or as a procedure or package (body) awaiting wrapper generation. |
| commfailure | At least one master site is not accessible. |
| notcompat | This procedure requires release 7.3 or greater. |
| notquiesced | The replicated object group was not quiesced. |

# DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT

## Purpose

To generate the triggers, packages, and procedures needed to support replication. You must call this procedure from the master definition site.

## Syntax

The parameters for the GENERATE_REPLICATION_SUPPORT procedure are described in Table 9–156, and the exceptions are listed in Table 9–157. The syntax for this procedure is shown below:

```
DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT(
  sname                 IN    VARCHAR2,
  oname                 IN    VARCHAR2,
  type                  IN    VARCHAR2,
  package_prefix        IN    VARCHAR2   := NULL,
  procedure_prefix      IN    VARCHAR2   := NULL,
  distributed           IN    BOOLEAN    := TRUE,
  gen_objs_owner        IN    VARCHAR2   := NULL,
  min_communication     IN    BOOLEAN    := TRUE )
```

***Table 9–156 Parameters for GENERATE_REPLICATION_SUPPORT***

| Parameter | Description |
|---|---|
| sname | The schema in which the object is located. |
| oname | The name of the object for which you are generating replication support. |
| type | The type of the object. The types supported are: TABLE, PACKAGE, and PACKAGE BODY. |
| package_prefix | For objects of type PACKAGE or PACKAGE BODY this value is prepended to the generated wrapper package name. The default is DEFER_. |
| procedure_prefix | For objects of type PROCEDURE, PACKAGE or PACKAGE BODY, this value is prepended to the generated wrapper procedure names. By default, no prefix is assigned. The default is DEFER_. |
| distributed | This parameter must be set to TRUE if your COMPATIBLE parameter is set to 7.3.0 or greater. |

***Table 9–156 Parameters for GENERATE_REPLICATION_SUPPORT***

| Parameter | Description |
|---|---|
| gen_objs_owner | The name of the user you want to use as owner of the transaction. |
| min_communication | Set to FALSE if any master site is running Oracle7 release 7.3. Set to TRUE when you want propagation of new and old values to be minimized. The default is TRUE. For more information, see "Minimizing Data Propagation for Update Conflict Resolution" on page 5-40. |

***Table 9–157 Exceptions for GENERATE_REPLICATION_SUPPORT***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| missingobject | The given object does not exist as a table in the given schema awaiting row-level replication information or as a procedure or package (body) awaiting wrapper generation. |
| typefailure | The given type parameter is not supported. |
| notquiesced | The replicated object group has not been suspended. |
| commfailure | At least one master site is not accessible. |
| missschema | Schema does not exist. |
| dbnotcompatible | One of the masters is not 7.3 compatible. |
| duplicateobject | Object already exists. |

# DBMS_REPCAT.GENERATE_REPLICATION_TRIGGER

## Purpose

To provide more fine-grained control of replication support generation. Primarily used for environments that include Oracle7 Release 7.3 sites. Generates the triggers and their associated packages needed to support replication for a given object at all master sites, or to generate the triggers and their associated packages needed to support replication for all of the objects in a given object group at a list of master sites. You must call this procedure from the master definition site. The associated object group must be quiesced.

## Syntax

The parameters for the GENERATE_REPLICATION_TRIGGER procedure are described in Table 9–158, and the exceptions are listed in Table 9–159. This syntax for this procedure is shown below:

```
DBMS_REPCAT.GENERATE_REPLICATION_TRIGGER(
      sname                       IN  VARCHAR2,
      oname                       IN  VARCHAR2,
      gen_objs_owner              IN  VARCHAR2 := NULL
      min_communication           IN  BOOLEAN  := TRUE)
DBMS_REPCAT.GENERATE_REPLICATION_TRIGGER(
      gname                       IN  VARCHAR2,
      gen_objs_owner              IN  VARCHAR2 := NULL
      min_communication           IN  BOOLEAN  := NULL)
```

**Attention**: The GENERATE_REPLICATION_TRIGGER procedure is overloaded to allow you to generate support for a single object at all master sites or for an object group at a list of sites. Because the parameter types are the same for both calls, you may need to use named notation to indicate whether you are calling the procedure for a single object or for an object group.

**Attention**: If you want to generate support for a list of master sites (that is, if you will not be using the default, NULL), you must use either an array or named notation.

***Table 9–158 Parameters for GENERATE_REPLICATION_TRIGGER***

| Parameter | Description |
|---|---|
| sname | The schema in which the object is located. |
| oname | The name of the object for which you are generating replication support. |
| gname | The name of the object group for which you want to generate support. |
| gen_objs_owner | This parameter is provided for compatibility with previous releases. If you have any pre-release 7.3 snapshot sites, you must set this parameter to TRUE. |
| min_communication | Set to FALSE if any master site is running Oracle7 release 7.3. Set to TRUE when you want propagation of new and old values to be minimized. The default is varies. For more information, see "Minimizing Data Propagation for Update Conflict Resolution" on page 5-40. |

***Table 9–159 Exceptions for GENERATE_REPLICATION_TRIGGER***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| missingobject | The given object does not exist as a table in the given schema awaiting row-level replication information or as a procedure or package (body) awaiting wrapper generation. |
| notquiesced | The replicated object group has not been suspended. |
| commfailure | At least one master site is not accessible. |
| notcompat | One of the masters is not 7.3 compatible. |
| missingschema | The given schema does not exist. |

## Altering Propagation Mode

After altering the propagation mode of an object group, you need to regenerate the supporting PL/SQL triggers for these objects at each Oracle7 Release 7.3 site in the replicated environment. Internal triggers automatically change propagation mode as directed by ALTER_MASTER_PROPAGATION and are not affected by attempts to regenerate triggers.

To generate the supporting PL/SQL triggers and their associated packages for all Oracle7 release 7.3 members of an object group for a given set of master sites, call the DBMS_REPCAT.GENERATE_REPLICATION_TRIGGER procedure, as shown in the following example:

```
DBMS_REPCAT.GENERATE_REPLICATION_TRIGGER( gname => 'acct');
```

Because no list of master sites is specified in this example, Oracle regenerates the supporting triggers and their associated packages for the objects in the GNAME object group at all Oracle7 release 7.3 master sites. You must call this procedure from the master definition site for the given replicated object group. Oracle must successfully create the necessary triggers at the master definition site for this procedure to complete successfully. These objects are asynchronously created at the other master sites as described

# DBMS_REPCAT.GENERATE_SNAPSHOT_SUPPORT

## PURPOSE

To activate triggers and generate packages needed to support the replication of updatable snapshots or procedural replication.You must call this procedure from the snapshot site.

## SYNTAX

The parameters for the GENERATE_SNAPSHOT_SUPPORT procedure are described in Table 9–160, and the exceptions are listed in Table 9–161. The syntax for this procedure is shown below:

```
DBMS_REPCAT.GENERATE_SNAPSHOT_SUPPORT
      sname              IN VARCHAR2,
      oname              IN VARCHAR2,
      type               IN VARCHAR2,
      gen_objs_owner     IN VARCHAR2 := '',
      min_communication  IN BOOLEAN  := TRUE)
```

**Note:** CREATE_SNAPSHOT_REPOBJECT automatically generates snapshot support for updatable snapshots.

***Table 9–160 Parameters for GENERATE_SNAPSHOT_SUPPORT***

| Parameter | Description |
|---|---|
| sname | The schema in which the object is located. |
| oname | If the object exists in the replicated snapshot object group as an updatable snapshot using row/column level replication, generate the row-level replication trigger and stored packages. |
| type | Type of the object. The types supported are SNAPSHOT, PACKAGE, and PACKAGE BODY. |
| gen_objs_owner | Specifies the schema in which the generated replication trigger and trigger package or wrapper is installed. If NULL, the generated trigger and trigger package or wrapper are installed in the schema specified by the sname parameter. |

*Table 9–160 Parameters for GENERATE_SNAPSHOT_SUPPORT*

| Parameter | Description |
|---|---|
| min_communication | If TRUE, the update trigger sends the new value of a column only if the update statement modifies the column. The update trigger sends the old value of the column only if it is a key column or a column in a modified column group. |

*Table 9–161 Exceptions for GENERATE_SNAPSHOT_SUPPORT*

| Exceptions | Descriptions |
|---|---|
| nonsnapshot | The invocation site is not a snapshot site. |
| missingobject | The given object does not exist as a snapshot in the replicated schema awaiting row/column-level replication information or as a procedure or package (body) awaiting wrapper generation. |
| typefailure | The given type parameter is not supported. |
| missingschema | The specified owner of generated objects does not exist. |
| missingremoteobject | The master object has not yet generated replication support. |
| commfailure | The master is not accessible. |

# DBMS_REPCAT.MAKE_COLUMN_GROUP

## Purpose

To create a new column group with one or more members. You must call this procedure from the master definition site. For more information, see “Creating a Column Group” on page 5-13.

## Syntax

The parameters for the MAKE_COLUMN_GROUP procedure are described in Table 9–162, and the exceptions are listed in Table 9–163. The syntax for this procedure is shown below:

```
DBMS_REPCAT.MAKE_COLUMN_GROUP(
     sname                 IN   VARCHAR2,
     oname                 IN   VARCHAR2,
     column_group          IN   VARCHAR2,
     list_of_column_names  IN   VARCHAR2 | DBMS_REPCAT.VARCHAR2S)
```

***Table 9–162 Parameters for MAKE_COLUMN_GROUP***

| Parameter | Description |
|---|---|
| sname | The schema in which the replicated table is located. |
| oname | The name of the replicated table for which you are creating a new column group. |
| column_group | The name that you want assigned to the column group that you are creating. |
| list_of_column_names | The names of the columns that you are grouping. This can either be a comma-separated list or a PL/SQL table of column names. The PL/SQL table must be of type dbms_repcat.varchar2s. Use the single value ‘*’ to create a column group that contains all of the columns in your table. |

*Table 9–163 Exceptions for MAKE_COLUMN_GROUP*

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the masterdef site. |
| duplicategroup | The given column group already exists for the table. |
| missingobject | The given table does not exist. |
| missingcolumn | The given column does not exist in the designated table. |
| duplicatecolumn | The given column is already a member of another column group. |
| notquiesced | The replicated object group is not quiesced. |

# DBMS_REPCAT.PURGE_MASTER_LOG

## Purpose

To remove local messages in the RepCatLog associated with a given identification number, source, or replicated object group.

## Syntax

The parameters for the PURGE_MASTER_LOG procedure are described in Table 9–164, and the exception is listed in Table 9–165. If any parameter is NULL, Oracle treats it as a wildcard. The syntax for this procedure is shown below:

```
DBMS_REPCAT.PURGE_MASTER_LOG(
      id      IN   NATURAL,
      source  IN   VARCHAR2,
      gname   IN   VARCHAR2)
```

***Table 9–164 Parameters for PURGE_MASTER_LOG***

| Parameter | Description |
|---|---|
| id | The identification number of the request, as it appears in the RepCatLog view. |
| source | The master site from which the request originated. |
| gname | The name of the replicated object group for which the request was made. |

***Table 9–165 Exception for PURGE_MASTER_LOG***

| Exception | Description |
|---|---|
| nonmaster | GNAME is not NULL and the invocation site is not a master site. |

# DBMS_REPCAT.PURGE_STATISTICS

## Purpose

To remove information from the RepResolution_Statistics view.

## Syntax

The parameters for the PURGE_STATISTICS procedure are described in Table 9–166, and the exceptions are listed in Table 9–167. The syntax for this procedure is shown below:

```
DBMS_REPCAT.PURGE_STATISTICS(
      sname      IN   VARCHAR2,
      oname      IN   VARCHAR2,
      start_date IN   DATE,
      end_date   IN   DATE)
```

***Table 9–166 Parameters for PURGE_STATISTICS***

| Parameter | Description |
|---|---|
| sname | The name of the schema in which the replicated table is located. |
| oname | The name of the table whose conflict resolution statistics you want to purge. |
| start_date/end_date | The range of dates for which you want to purge statistics. If START_DATE is NULL, purge all statistics up to the END_DATE. If END_DATE is NULL, purge all statistics after the START_DATE. |

***Table 9–167 Exceptions for PURGE_STATISTICS***

| Exception | Description |
|---|---|
| missingschema | The given schema does not exist. |
| missingobject | The given table does not exist. |
| statnotreg | Table not registered to collect statistics. |

# DBMS_REPCAT.REFRESH_SNAPSHOT_REPGROUP

## Purpose

To refresh a snapshot site object group with the most recent data from its associated master site.

## Syntax

The parameters for the REFRESH_SNAPSHOT_REPGROUP procedure are described in Table 9–168, and the exceptions are listed in Table 9–169. The syntax for this procedure is shown below:

```
DBMS_REPCAT.REFRESH_SNAPSHOT_REPGROUP(
     gname                 IN   VARCHAR2,
     drop_missing_contents IN   BOOLEAN    := FALSE,
     refresh_snapshots     IN   BOOLEAN    := FALSE,
     refresh_other_objects IN   BOOLEAN    := FALSE)
```

***Table 9–168 Parameters for REFRESH_SNAPSHOT_REPGROUP***

| Parameter | Description |
|---|---|
| gname | The name of the replicated object group. |
| drop_missing_contents | If an object was dropped from the replicated object group, it is not automatically dropped from the schema at the snapshot site. It is simply no longer replicated; that is, changes to this object are no longer sent to its associated master site. Snapshots can continue to be refreshed from their associated master tables; however, any changes to an updatable snapshot will be lost. When an object is dropped from the object group, you can choose to have it dropped from the schema entirely by setting this argument to TRUE. |
| refresh_snapshots | Set this parameter to TRUE to refresh the contents of the snapshots in the replicated object group. |
| refresh_other_objects | Set this parameter to TRUE to refresh the contents of the non-snapshot objects in the replicated object group. |

***Table 9–169 Exceptions for REFRESH_SNAPSHOT_REPGROUP***

| Exception | Description |
|---|---|
| nonsnapshot | The invocation site is not a snapshot site. |
| nonmaster | The master is no longer a master site. |
| commfailure | The master is not accessible. |
| missrepgroup | Object group name not specified. |

# DBMS_REPCAT.REGISTER_SNAPSHOT_REPGROUP

## Purpose

To facilitate the administration of snapshots at their respective master sites by inserting / modifying / deleting from repcat_repsite.

## Syntax

The parameters for REGISTER_SNAPSHOT_REPGROUP are described in Table 9–170. The syntax for the REGISTER_SNAPSHOT_REPGROUP procedure is shown below.

```
DBMS_REPCAT.REGISTER_SNAPGROUP_REPGROUP(
     gname            IN   VARCHAR2,
     snapsite         IN   VARCHAR2,
     comment          IN   VARCHAR2  := NULL,
     rep_type         IN   NUMBER    := reg_unknown)
```

***Table 9–170 Parameters for REGISTER_SNAPSHOT_REPGROUP***

| Parameter | Description |
|---|---|
| gname | The name of the snapshot object group to be registered. |
| snapsite | Global name of the snapshot site. |
| comment | Comment for the snapshot site or update for an existing comment. |
| rep_type | Version of the snapshot group. Valid constants that can be assigned include reg_uknown (the default), reg_v7_group, reg_v8_group, and reg_repapi_group. |

***Table 9–171 Exceptions for REGISTER_SNAPSHOT_REPGROUP***

| Exception | Description |
|---|---|
| missrepgroup | Object group name not specified. |
| nullsitename | A snapshot site was not specified. |
| nonmaster | The procedure must be executed at the snapshot's master site. |
| duplrepgrp | The object already exists. |

# DBMS_REPCAT.REGISTER_STATISTICS

## Purpose

To collect information about the successful resolution of update, delete and uniqueness conflicts for a table.

## Syntax

The parameters for the REGISTER_STATISTICS procedure are described in Table 9–172, and the exceptions are listed in Table 9–173. The syntax for this procedure is shown below:

```
DBMS_REPCAT.REGISTER_STATISTICS(
      sname IN   VARCHAR2,
      oname IN   VARCHAR2)
```

***Table 9–172 Parameters for REGISTER_STATISTICS***

| Parameter | Description |
|---|---|
| sname | The name of the schema in which the table is located. |
| oname | The name of the table for which you want to gather conflict resolution statistics. |

***Table 9–173 Exceptions for REGISTER_STATISTICS***

| Exception | Description |
|---|---|
| missingschema | The given schema does not exist. |
| missingobject | The given table does not exist. |

# DBMS_REPCAT.RELOCATE_MASTERDEF

## Purpose

To change your master definition site to another master site in your replicated environment.

## Syntax

The parameters for the RELOCATE_MASTERDEF procedure are described in Table 9–174, and the exceptions are listed in Table 9–175. The syntax for this procedure is shown below:

```
DBMS_REPCAT.RELOCATE_MASTERDEF(
     gname                  IN   VARCHAR2,
     old_masterdef          IN   VARCHAR2,
     new_masterdef          IN   VARCHAR2,
     notify_masters         IN   BOOLEAN    := TRUE,
     include_old_masterdef  IN   BOOLEAN    := TRUE)
```

***Table 9–174 Parameters for RELOCATE_MASTERDEF***

| Parameter | Description |
|---|---|
| gname | The name of the object group whose master definition you want to relocate. |
| old_masterdef | The fully qualified database name of the current master definition site. |
| new_masterdef | The fully qualified database name of the existing master site that you want to make the new master definition site. |
| notify_masters | If NOTIFY_MASTERS is TRUE, the procedure synchronously multicasts the change to all masters (including OLD_MASTERDEF only if INCLUDE_OLD_MASTERDEF is TRUE). If any master does not make the change, roll back the changes at all masters. |
| include_old_masterdef | If NOTIFY_MASTERS is TRUE and INCLUDE_OLD_MASTERDEF is also TRUE, the old master definition site is also notified of the change. |

*Table 9–175 Exceptions for RELOCATE_MASTERDEF*

| Exception | Description |
|---|---|
| nonmaster | NEW_MASTERDEF is not a master site or the invocation site is not a master site. |
| nonmasterdef | OLD_MASTERDEF is not the master definition site. |
| commfailure | At least one master site is not accessible and NOTIFY_MASTERS is TRUE. |

## Usage Notes

It is not necessary for either the old or new master definition site to be available when you call RELOCATE_MASTERDEF. In a planned reconfiguration, you should invoke RELOCATE_MASTERDEF with NOTIFY_MASTERS TRUE and INCLUDE_OLD_MASTERDEF TRUE. If just the master definition site fails, you should invoke RELOCATE_MASTERDEF with NOTIFY_MASTERS TRUE and INCLUDE_OLD_MASTERDEF FALSE. If several master sites and the master definition site fail, the administrator should invoke RELOCATE_MASTERDEF at each operational master with NOTIFY_MASTERS FALSE.

# DBMS_REPCAT.REMOVE_MASTER_DATABASES

## Purpose

To remove one or more master databases from a replicated environment. This procedure regenerates the triggers and their associated packages at the remaining master sites. You must call this procedure from the master definition site.

## Syntax

The parameters for the REMOVE_MASTER_DATABASES procedure are described in Table 9–176, and the exceptions are listed in Table 9–177. The syntax for this procedure is shown below:

```
DBMS_REPCAT.REMOVE_MASTER_DATABASES(
     gname           IN   VARCHAR2,
     master_list     IN   VARCHAR2 |
     master_table    IN   DBMS_UTILITY.DBLINK_ARRAY)
```

***Table 9–176 Parameters for REMOVE_MASTER_DATABASES***

| Parameter | Description |
|---|---|
| gname | The name of the object group associated with the replicated environment. This prevents confusion if a master database is involved in more than one replicated environment. |
| master_list | A comma-separated list of fully qualified master database names that you want to remove from the replicated environment. There must be no extra white space between names in the list. |
| master_table | In place of a list, you may also specify the database names in a PL/SQL table of type DBMS_UTILITY.DBLINK_ARRAY. |

*Table 9–177 Exceptions for REMOVE_MASTER_DATABASES*

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| nonmaster | At least one of the given databases is not a master site. |
| reconfigerror | One of the given databases is the master definition site. |
| commfailure | At least one remaining master site is not accessible. |

# DBMS_REPCAT.REPCAT_IMPORT_CHECK

## Purpose

To ensure that the objects in the replicated object group have the appropriate object identifiers and status values after you perform an export/import of a replicated object or an object used by the advanced replication facility.

## Syntax

The parameters for the REPCAT_IMPORT_CHECK procedure are described in Table 9–178, and the exceptions are listed in Table 9–179. The syntax for this procedure is shown below:

```
DBMS_REPCAT.REPCAT_IMPORT_CHECK(
     gname      IN   VARCHAR2,
     master     IN   BOOLEAN)
```

***Table 9–178 Parameters for REPCAT_IMPORT_CHECK***

| Parameter | Description |
|---|---|
| gname | The name of the replicated object group. If you omit both parameters, the procedure checks all replicated object groups at your current site. |
| master | Set this flag to TRUE if you are checking a master site or FALSE if you are checking a snapshot site. |

***Table 9–179 Exceptions for REPCAT_IMPORT_CHECK***

| Exception | Description |
|---|---|
| nonmaster | MASTER is TRUE and either the database is not a master site for the schema or the database is not the expected database. |
| nonsnapshot | MASTER is FALSE and the database is not a snapshot site for the schema. |
| missingobject | A valid replicated object in the schema does not exist. |
| missingschema | The given group name does not exist. |

# DBMS_REPCAT.RESUME_MASTER_ACTIVITY

## Purpose

To resume normal replication activity after quiescing a replicated environment.

## Syntax

The parameters for the RESUME_MASTER_ACTIVITY procedure are described in Table 9–180, and the exceptions are listed in Table 9–181. The syntax for this procedure is shown below:

```
DBMS_REPCAT.RESUME_MASTER_ACTIVITY(
     gname       IN  VARCHAR2,
     override    IN  BOOLEAN := FALSE)
```

***Table 9–180 Parameters for RESUME_MASTER_ACTIVITY***

| Parameter | Description |
|---|---|
| gname | The name of the replicated object group. |
| override | If override is TRUE, it ignores any pending RepCat administration requests and restores normal replication activity at each master as quickly as possible. This should be considered only in emergency situations. If override is FALSE, it restores normal replication activity at each master only when there is no pending RepCat administration request for GNAME at that master. |

***Table 9–181 Exceptions for RESUME_MASTER_ACTIVITY***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| notquiesced | The replicated object group is not quiescing or quiesced. |
| commfailure | At least one master site is not accessible. |

# DBMS_REPCAT.SEND_AND_COMPARE_OLD_VALUES

## Purpose

You have the option of sending old column values for each non-key column of a replicated table for updates and deletes. The default is to send old values for all columns. You can change this behavior at all master and snapshot sites by invoking DBMS_REPCAT.SEND_AND_COMPARE_OLD_VALUES at the master definition site.

**Note:** The OPERATION parameter allows you to decide whether or not to transmit old values for non-key columns when rows are deleted or when non-key columns are updated. If you do not send the old value, 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.

**Caution:** Read “Minimizing Data Propagation for Update Conflict Resolution” on page 5-40.before changing the default behavior of Oracle.

## Syntax

The parameters for the SEND_AND_COMPARE_OLD_VALUES procedure are described in Table 9–182, and the exceptions are listed in Table 9–183. The syntax for this procedure is shown below:

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

**Note:** This procedure is overloaded. The COLUMN_LIST and COLUMN_TABLE parameters are mutually exclusive.

*Table 9–182 Parameters for SEND_AND_COMPARE_OLD_VALUES*

| Parameter | Description |
|---|---|
| sname | The schema in which the table is located. |
| oname | The name of the table. |
| column_list | A comma-separated list of the columns in the table. There must be no white space between entries. |
| column_table | Instead of a list, you can use a PL/SQL table of type DBMS_REPCAT.VARCHAR2S to contain the column names. The first column name should be at offset 1, the second at offset 2, and so on. |
| operation | Possible values are: UPDATE, DELETE, or the asterisk wildcard '*', which means update and delete. |
| send | If TRUE, the old values of the specified columns are sent. If FALSE, the old values of the specified columns are not sent. Unspecified columns and unspecified operations are not affected. The specified change takes effect at the master definition site as soon as min_communication is TRUE for the table. The change takes effect at a master site or at a snapshot site the next time replication support is generated at that site with min_communication TRUE. |

*Table 9–183 Exceptions for SEND_AND_COMPARE_OLD_VALUES*

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| missingobject | The given object does not exist as a table in the given schema awaiting row-level replication information. |
| missingcolumn | At least one column is not in the table. |
| notquiesced | The replicated object group has not been suspended. |
| typefailure | An illegal operation is given. |

# DBMS_REPCAT.SET_COLUMNS

## Purpose

To use an alternate column or group of columns, instead of the primary key, to determine which columns of a table to compare when using row-level replication. You must call this procedure from the master definition site. See "Designating an Alternate Key for a Replicated Table" on page 3-20

## Syntax

The parameters for the SET_COLUMNS procedure are described in Table 9–184, and the exceptions are listed in Table 9–185. The syntax for this procedure is shown below:

```
DBMS_REPCAT.SET_COLUMNS(
     sname           IN    VARCHAR2,
     oname           IN    VARCHAR2,
     { column_list   IN    VARCHAR2
     | column_table  IN    DBMS_UTILITY.NAME_ARRAY } )
```

**Note:** This procedure is overloaded. The COLUMN_LIST and COLUMN_TABLE parameters are mutually exclusive.

***Table 9–184 Parameters for SET_COLUMNS***

| Parameter | Description |
|---|---|
| sname | The schema in which the table is located. |
| oname | The name of the table. |
| column_list | A comma-separated list of the columns in the table that you want to use as a "primary key". There must be no white space between entries. |
| column_table | Instead of a list, you can use a PL/SQL table of type DBMS_UTILITY.NAME_ARRAY to contain the column names. The first column name should be at offset 1, the second at offset 2, and so on. |

*Table 9–185 Exceptions for SET_COLUMNS*

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| missingobject | The given object does not exist as a table in the given schema awaiting row-level replication information. |
| missingcolumn | At least one column is not in the table. |

# DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY

## Purpose

To suspend replication activity for an object group. You must call this procedure from the master definition site.

**Note**: The current implementation of SUSPEND_MASTER_ACTIVITY quiesces all replicated object groups at each master site.

## Syntax

The parameter for the SUSPEND_MASTER_ACTIVITY procedure is described in Table 9–186, and the exceptions are listed in Table 9–187. The syntax for this procedure is shown below:

```
DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY(gname   IN   VARCHAR2)
```

***Table 9–186 Parameter for SUSPEND_MASTER_ACTIVITY***

| Parameter | Description |
|---|---|
| gname | The name of the object group for which you want to suspend activity. |

***Table 9–187 Exceptions for SUSPEND_MASTER_ACTIVITY***

| Exception | Description |
|---|---|
| nonmasterdef | The invocation site is not the master definition site. |
| notnormal | The replicated object group is not in normal operation. |
| commfailure | At least one master site is not accessible. |
| missingobjectgroup | The given object group does not exist. |

# DBMS_REPCAT.SWITCH_SNAPSHOT_MASTER

## Purpose

To change the master database of a snapshot replicated object group to another master site. This procedure does a full refresh of the affected snapshots and regenerates the triggers and their associated packages as needed. This procedure does not push the queue to the old master site before changing masters.

## Syntax

The parameters for the SWITCH_SNAPSHOT_MASTER procedure are described in Table 9–188, and the exceptions are listed in Table 9–189. The syntax for this procedure is shown below:

```
DBMS_REPCAT.SWITCH_SNAPSHOT_MASTER(
     gname                IN   VARCHAR2,
     master               IN   VARCHAR2)
```

***Table 9–188 Parameters for SWITCH_SNAPSHOT_MASTER***

| Parameter | Description |
|---|---|
| gname | The name of the snapshot object group for which you want to change master sites. |
| master | The fully qualified database name of the new master database to use for the snapshot site. |

***Table 9–189 Exceptions for SWITCH_SNAPSHOT_MASTER***

| Exception | Description |
|---|---|
| nonsnapshot | The invocation site is not a snapshot site. |
| nonmaster | The given database is not a master site. |
| commfailure | The given database is not accessible. |

# DBMS_REPCAT.UNREGISTER_SNAPSHOT_REPGROUP

## Purpose

To facilitate the administration of snapshots at their respective master sites by inserting / modifying / deleting from repcat$_repsite.

## Syntax

The parameters for UNREGISTER_SNAPSHOT_REPGROUP are described in Table 9–190. The syntax for the UNREGISTER_SNAPSHOT_REPGROUP procedure is shown below.

```
DBMS_REPCAT.UNREGISTER_SNAPGROUP_REPGROUP(
     gname     IN   VARCHAR2,
     snapsite  IN   VARCHAR2)
```

***Table 9–190 Parameters for UNREGISTER_SNAPSHOT_REPGROUP***

| Parameter | Description |
|---|---|
| gname | The name of the snapshot object group to be unregistered. |
| snapsite | Global name of the snapshot site. |

# DBMS_REPCAT.VALIDATE

## Purpose

To validate the correctness of key conditions of a multiple master replication environment, you can execute the VALIDATE procedure, which is overloaded.

## Syntax

The parameters for the VALIDATE function are described in Table 9–191 and the exceptions are described in Table 9–192. The syntax for this procedure is shown below:

```
DBMS_REPCAT.VALIDATE (
      gname               IN VARCHAR2,
      check_genflags      IN BOOLEAN := FALSE,
      check_valid_objs    IN BOOLEAN := FALSE,
      check_links_sched   IN BOOLEAN := FALSE,
      check_links         IN BOOLEAN := FALSE,
      error_table         OUT dbms_repcat.validate_err_table )
  RETURN BINARY_INTEGER

DBMS_REPCAT.VALIDATE (
      gname               IN VARCHAR2,
      check_genflags      IN BOOLEAN := FALSE,
      check_valid_objs    IN BOOLEAN := FALSE,
      check_links_sched   IN BOOLEAN := FALSE,
      check_links         IN BOOLEAN := FALSE,
      error_msg_table     OUT DBMS_UTILITY.UNCL_ARRAY,
      error_num_table     OUT DBMS_UTILITY.NUMBER_ARRAY )
  RETURN BINARY_INTEGER
```

***Table 9–191 Parameters for VALIDATE***

| Parameter | Description |
|---|---|
| gname | The name of the master group to validate. |
| check_genflags | Check whether all the objects in the group are generated. This must be done at the masterdef site only. |
| check_valid_objs | Check that the underlying objects for objects in the group valid. This must be done at the masterdef site only. The masterdef site goes to all other sites and checks that the underlying objects are valid. The validity of the objects is checked within the schema of the connected user. |

*Table 9–191 Parameters for VALIDATE*

| Parameter | Description |
|---|---|
| check_links_sched | Check whether the links are scheduled for execution. This should be invoked at each master site. |
| check_links | Check whether the connected user (repadmin), as well as the propagator, have correct links for replication to work properly. Checks that the links exist in the database and are accessible. This should be invoked at each master site. |
| error_table | Returns the message and numbers of all errors that are found. |
| error_msg_table | Returns the messages of all errors that are found. |
| error_num_table | Returns the numbers of all errors that are found. |

The return value of VALIDATE is the number of errors found. The function's OUT parameter(s) returns any errors that are found. In the first interface function, the ERROR_TABLE consists of an array of records. Each record has a VARCHAR2 and a NUMBER in it. The string field contains the error message and the number field contains the Oracle error number.

The second interface is similar except that there are two OUT arrays. A VARCHAR2 array with the error messages and a NUMBER array with the error numbers.

*Table 9–192 Exceptions for VALIDATE*

| Exception | Description |
|---|---|
| missingdblink | The database link does not exist in the schema of the replication propagator or has not been scheduled. Ensure that the database link exists in the database, is accessible, and is scheduled for execution. |
| dblinkmismatch | The database link name at the local node does not match the global name of the database that the link accesses. Ensure that global names is set to true and the link name matches the global name. |

*Table 9–192 Exceptions for VALIDATE*

| Exception | Description |
|---|---|
| dblinkuidmismatch | The user name of the replication administration user at the local node and the user name at the node corresponding to the database link are not the same. Advanced replication expects the two users to be the same. Ensure that the user ID of the replication administration user at the local node and the user ID at the node corresponding to the database link are the same. |
| objectnotgenerated | The object has not been generated at other master sites or is still being generated. Ensure that the object is generated by calling generate_replication_support and do_deferred_repcat_admin for the object at the masterdef site. |
| opnotsupported | Operation is not supported if the object group is replicated at a pre-V8 node. Ensure that all nodes of the replicated object group are V8. |

# DBMS_REPCAT.WAIT_MASTER_LOG

## Purpose

To determine whether changes that were asynchronously propagated to a master site have been applied.

## Syntax

The parameters for the WAIT_MASTER_LOG procedure are described in Table 9–193, and the exception is listed in Table 9–194. The syntax for this procedure is shown below:

```
DBMS_REPCAT.WAIT_MASTER_LOG(
     gname          IN   VARCHAR2,
     record_count   IN   NATURAL,
     timeout        IN   NATURAL,
     true_count     OUT   NATURAL)
```

***Table 9–193 Parameters for WAIT_MASTER_LOG***

| Parameter | Description |
|---|---|
| gname | The name of the replicated object group. |
| record_count | The procedure returns whenever the number of incomplete activities is at or below this threshold. |
| timeout | The maximum number of seconds to wait before the procedure returns. |
| true_count (out parameter) | Returns the number of incomplete activities. |

***Table 9–194 Exception for WAIT_MASTER_LOG***

| Exception | Description |
|---|---|
| nonmaster | The invocation site is not a master site. |

# DBMS_REPCAT_ADMIN Package

The DBMS_REPCAT_ADMIN package contains the following procedures:

- DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA
- DBMS_REPCAT_ADMIN.GRANT_ADMIN_SCHEMA
- DBMS_REPCAT_ADMIN.REVOKE_ADMIN_ANY_SCHEMA
- DBMS_REPCAT_ADMIN.REVOKE_ADMIN_SCHEMA

The following pages discuss each procedure.

# DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA

## Purpose

To grant the necessary privileges to the replication administrator to administer any replicated object group at the current site.

## Syntax

The parameter for the GRANT_ADMIN_ANY_SCHEMA procedure is described in Table 9–195, and the exception is described in Table 9–196. The syntax for this procedure is shown below:

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA(username  IN   VARCHAR2)
```

***Table 9–195 Parameter for GRANT_ADMIN_ANY_SCHEMA***

| Parameter | Description |
|---|---|
| username | The name of the replication administrator to whom you want to grant the necessary privileges and roles to administer any replicated object groups at the current site. |

***Table 9–196 Exception for GRANT_ADMIN_ANY_REPGROUP***

| Exception | Description |
|---|---|
| ORA-01917 | The user does not exist. |

# DBMS_REPCAT_ADMIN.GRANT_ADMIN_SCHEMA

## Purpose

To grant the necessary privileges to the replication administrator to administer a schema at the current site. This procedure is most useful if your object group does not span schemas.

## Syntax

The parameter for the GRANT_ADMIN_REPSCHEMA procedure is described in Table 9–197, and the exception is described in Table 9–198. The syntax for this procedure is shown below:

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_SCHEMA(username IN VARCHAR2)
```

***Table 9–197 Parameter for GRANT_ADMIN_REPSCHEMA***

| Parameter | Description |
|---|---|
| username | The name of the replication administrator. This user is then granted the necessary privileges and roles to administer the schema of the same name within a replicated object group at the current site. |

***Table 9–198 Exception for GRANT_ADMIN_REPSCHEMA***

| Exception | Description |
|---|---|
| ORA-01917 | The user does not exist. |

# DBMS_REPCAT_ADMIN.REVOKE_ADMIN_ANY_SCHEMA

## Purpose

To revoke the privileges and roles from the replication administrator that would be granted by GRANT_ADMIN_ANY_SCHEMA.

**Attention**: Identical privileges and roles that were granted independently of GRANT_ADMIN_ANY_SCHEMA are also revoked.

## Syntax

The parameter for the REVOKE_ADMIN_ANY_SCHEMA procedure is described in Table 9–199, and the exception is described in Table 9–200. The syntax for this procedure is shown below:

```
DBMS_REPCAT_ADMIN.REVOKE_ADMIN_ANY_SCHEMA(username IN   VARCHAR2)
```

***Table 9–199 Parameter for REVOKE_ADMIN_ANY_SCHEMA***

| Parameter | Description |
|---|---|
| username | The name of the replication administrator whose privileges you want to revoke. |

***Table 9–200 Exception for REVOKE_ADMIN_ANY_SCHEMA***

| Exception | Description |
|---|---|
| ORA-01917 | The user does not exist. |

# DBMS_REPCAT_ADMIN.REVOKE_ADMIN_SCHEMA

## Purpose

To revoke the privileges and roles from the replication administrator that would be granted by GRANT_ADMIN_SCHEMA.

**Attention**: Identical privileges and roles that were granted independently of GRANT_ADMIN_SCHEMA are also revoked.

## Syntax

The parameter for the REVOKE_ADMIN_SCHEMA procedure is described in Table 9–201, and the exception is described in Table 9–202. The syntax for this procedure is shown below:

```
DBMS_REPCAT_ADMIN.REVOKE_ADMIN_SCHEMA(username IN VARCHAR2)
```

***Table 9–201 Parameter for REVOKE_ADMIN_SCHEMA***

| Parameter | Description |
|---|---|
| username | The name of the replication administrator whose privileges you want to revoke. |

***Table 9–202 Exception for REVOKE_ADMIN_SCHEMA***

| Exception | Description |
|---|---|
| ORA-01917 | The user does not exist. |

# DBMS_REPCAT_AUTH Package

The DBMS_REPCAT_AUTH package contains the following procedures:

- DBMS_REPCAT_AUTH.GRANT_SURROGATE_REPCAT
- DBMS_REPCAT_AUTH.REVOKE_SURROGATE_REPCAT

The following pages discuss each procedure.

# DBMS_REPCAT_AUTH.GRANT_SURROGATE_REPCAT

## Purpose

To grant the privileges needed by the advanced replication facility to a user.

## Syntax

The parameter for the GRANT_SURROGATE_REPCAT procedure is described in Table 9–203, and the exception is described in Table 9–204. The syntax for this procedure is shown below:

```
DBMS_REPCAT_AUTH.GRANT_SURROGATE_REPCAT(userid IN VARCHAR2)
```

***Table 9–203 Parameter for GRANT_SURROGATE_REPCAT***

| Parameter | Description |
|---|---|
| userid | The name of the user to whom you wish to grant the necessary privileges. |

***Table 9–204 Exception for GRANT_SURROGATE_REPCAT***

| Exception | Description |
|---|---|
| ORA-01917 | The user does not exist. |

# DBMS_REPCAT_AUTH.REVOKE_SURROGATE_REPCAT

## Purpose

To revoke the privileges granted to the surrogate repcat user.

## Syntax

The parameters for the REVOKE_SURROGATE_REPCAT procedure is described in Table 9–205, and the exception is described in Table 9–206. The syntax for this procedure is shown below:

```
DBMS_REPCAT_AUTH.REVOKE_SURROGATE_REPCAT(userid IN VARCHAR2)
```

***Table 9–205 Parameter for REVOKE_SURROGATE_REPCAT***

| Parameter | Description |
|---|---|
| userid | The name of the user from whom you wish to revoke the necessary privileges. |

***Table 9–206 Exception for REVOKE_SURROGATE_REPCAT***

| Exception | Description |
|---|---|
| ORA-01917 | The user does not exist. |

# DBMS_REPUTIL Package

The DBMS_REPUTIL package contains the following procedures:

- DBMS_REPUTIL.REPLICATION_OFF
- DBMS_REPUTIL.REPLICATION_ON

The following pages discuss each procedure.

# DBMS_REPUTIL.REPLICATION_OFF

## Purpose

To modify tables without replicating the modifications to any other sites in the replicated environment, or to disable row-level replication when using procedural replication. In general, you should suspend replication activity for all master groups in your replicated environment before setting this flag.

## Syntax

The syntax for the REPLICATION_OFF procedure is shown below. This procedure takes no arguments.

```
DBMS_REPUTIL.REPLICATION_OFF
```

# DBMS_REPUTIL.REPLICATION_ON

## Purpose

To reenable replication of changes after replication has been temporarily suspended by calling REPLICATION_OFF.

## Syntax

The syntax for the REPLICATION_ON procedure is shown below. This procedure takes no arguments.

```
DBMS_REPUTIL.REPLICATION_ON
```

# DBMS_SNAPSHOT Package

The DBMS_SNAPSHOT package contains the following procedures and one function:

- DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION
- DBMS_SNAPSHOT.END_TABLE_REORGANIZATION
- DBMS_SNAPSHOT.I_AM_A_REFRESH
- DBMS_SNAPSHOT.PURGE_LOG
- DBMS_SNAPSHOT.REFRESH
- DBMS_SNAPSHOT.REGISTER_SNAPSHOT
- DBMS_SNAPSHOT.SET_I_AM_A_REFRESH
- DBMS_SNAPSHOT.UNREGISTER_SNAPSHOT

# DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION

## Purpose

This procedure must be called before a master table is reorganized. It performs process to preserve snapshot data needed for refresh.

**Additional Information:** See "Reorganizing Master Tables that Have Snapshot Logs" on page 2-30.

## Syntax

The parameters for the BEGIN_TABLE_REORGANIZATION procedure are described in Table 9–207. The syntax for BEGIN_TABLE_REORGANIZATION is shown below.

```
DBMS_SNAPSHOT.BEGIN_TABLE_REORGANIZATION(
      tabowner     IN   VARCHAR2
      tabname      IN   VARCHAR2)
```

***Table 9–207 Parameters for BEGIN_TABLE_REORGANIZATION***

| Parameter | Description |
|---|---|
| tabowner | The owner of the table being reorganized. |
| tabname | The name of the table being reorganized. |

# DBMS_SNAPSHOT.END_TABLE_REORGANIZATION

## Purpose

This procedure must be called after a master table is reorganized. It ensures that the snapshot data for the master table is valid and that the master table is in the proper state.

**Additional Information:** See "Reorganizing Master Tables that Have Snapshot Logs" on page 2-30.

## Syntax

The parameters for END_TABLE_REORGANIZATION are described in Table 9–208. The syntax for END_TABLE_REORGANIZATION is shown below.

```
DBMS_SNAPSHOT.END_TABLE_REORGANIZATION(
      tabowner     IN   VARCHAR2
      tabname      IN   VARCHAR2)
```

***Table 9–208 Parameters for END_TABLE_REORGANIZATION***

| Parameter | Description |
|---|---|
| tabowner | The owner of the table being reorganized. |
| tabname | The name of the table being reorganized. |

# DBMS_SNAPSHOT.I_AM_A_REFRESH

## Purpose

To return the value of the I_AM_REFRESH package state.

## Syntax

The I_AM_A_REFRESH function takes no arguments. A return value of TRUE indicates that all local replication triggers for snapshots will be effectively disabled in this session because each replication trigger first checks this state. A return value of FALSE indicates that these triggers are enabled. The syntax for this procedure is shown below.

```
DBMS_SNAPSHOT.I_AM_A_REFRESH RETURN BOOLEAN
```

# DBMS_SNAPSHOT.PURGE_LOG

## Purpose

To purge rows from the snapshot log.

## Syntax

The parameters for the PURGE_LOG procedure are described in Table 9–209. The syntax for this procedure is shown below:

```
DBMS_SNAPSHOT.PURGE_LOG(
     master        IN   VARCHAR2,
     num           IN   BINARY_INTEGER := 1,
     flag          IN   VARCHAR2       := 'NOP')
```

***Table 9–209 Parameters for PURGE_LOG***

| Parameter | Description |
|---|---|
| master | Name of the master table. |
| num | Number of least recently refreshed snapshots whose rows you want to remove from snapshot log. For example, the following statement deletes rows needed to refresh the two least recently refreshed snapshots:<br>`dbms_snapshot.purge_log('master_table', 2);`<br>To delete all rows in the snapshot log, indicate a high number of snapshots to disregard, as in this example:<br>`dbms_snapshot.purge_log('master_table',9999);`<br>This statement completely purges the snapshot log that corresponds to MASTER_TABLE if fewer than 9999 snapshots are based on MASTER_TABLE. A simple snapshot whose rows have been purged from the snapshot log must be completely refreshed the next time it is refreshed. |
| flag | Specify DELETE to guarantee that rows are deleted from the snapshot log for at least one snapshot. This argument can override the setting for the argument NUM. For example, the following statement deletes rows from the least recently refreshed snapshot that actually has dependent rows in the snapshot log:<br>`dbms_snapshot.purge_log('master_table',1,'DELETE');` |

# DBMS_SNAPSHOT.REFRESH

## Purpose

To consistently refresh one or more snapshots that are not members of the same refresh group. For additional information, see page Table 9–210.

## Syntax

The parameters for the REFRESH procedure are described in Table 9–210. The syntax for this procedure is shown below:

```
DBMS_SNAPSHOT.REFRESH(
      { list                IN VARCHAR2,
      | tab                 IN OUT DBMS_UTILITY.UNCL_ARRAY,}
      method                IN VARCHAR2       := NULL,
      rollback_seg          IN VARCHAR2       := NULL,
      push_deferred_rpc     IN BOOLEAN        := TRUE,
      refresh_after_errors  IN BOOLEAN        := FALSE,
      purge_option          IN BINARY_INTEGER := 1,
      parallelism           IN BINARY_INTEGER := 0,
      heap_size             IN BINARY_INTEGER := 0)
```

***Table 9–210 Parameters for REFRESH***

| Parameter | Description |
|---|---|
| list<br>tab | Comma-separated list of snapshots that you want to refresh. (Synonyms are not supported.) These snapshots can be located in different schemas and have different master tables; however, all of the listed snapshots must be in your current database. Alternatively, you may pass in a PL/SQL table of type DBMS_UTILITY.UNCL_ARRAY, where each element is the name of a snapshot. |

***Table 9–210 Parameters for REFRESH***

| Parameter | Description |
|---|---|
| method | Type of refresh to perform for each snapshot listed; 'F' or 'f' indicates a fast refresh, 'C' or 'c' indicates a complete refresh, and '?' indicates a default refresh. If you specified a refresh mode when you created the snapshot, that mode is used when you specify a default refresh. If no mode was specified, Oracle performs a fast refresh if possible; otherwise, it performs a complete refresh. If the METHOD list contains fewer elements than the snapshot LIST, the trailing elements in the snapshot list are refreshed using a default refresh. For example, the following EXECUTE statement within SQL*Plus:<br>`dbms_snapshot.refresh`<br>`('emp,dept,scott.salary','CF');`<br>performs a complete refresh of the EMP snapshot, a fast refresh of the DEPT snapshot, and a default refresh of the SCOTT.SALARY snapshot. |
| rollback_seg | Name of the snapshot site rollback segment to use while refreshing snapshots.<br>When you call REFRESH, all of the listed snapshots are updated to a single point in time. If the refresh fails for any of the snapshots, none of the snapshots are updated. |
| push_deferred_rpc | Used by updatable snapshots only. Use the default value, TRUE, if you want to push changes from the snapshot to its associated master before refreshing the snapshot. Otherwise, these changes may appear to be temporarily lost. |
| refresh_after_errors | Used by updatable snapshots only. Set this parameter to TRUE if you want the refresh to proceed even if there are outstanding conflicts logged in the DEFERROR view for the snapshot's master. |

***Table 9–210 Parameters for REFRESH***

| Parameter | Description |
|---|---|
| purge_option | If you are using the parallel propagation mechanism (in other words, parallelism is set to 1 or greater), 0 = don't purge; 1 = lazy (default); 2 = aggressive. In most cases, *lazy* purge is the optimal setting. Set purge to *aggressive* to trim back the queue if multiple master replication groups are pushed to different target sites, and updates to one or more replication groups are infrequent and infrequently pushed. If all replication groups are infrequently updated and pushed, set purge to *don't purge* and occasionally execute PUSH with purge set to *aggressive* to reduce the queue. |
| parallelism | 0 = serial propagation; *n* > 0 = parallel propagation with *n* parallel server processes; 1 = parallel propagation using only one parallel server process. |
| heap_size | Maximum number of transactions to be examined simultaneously for parallel propagation scheduling. Oracle automatically calculates the default setting for optimal performance. Do not set the parameter unless so directed by Oracle Worldwide Support. |

# DBMS_SNAPSHOT.REGISTER_SNAPSHOT

## Purpose

To enable the administration of individual snapshots.

## Syntax

The parameters for the REGISTER_SNAPSHOT procedure are described in Table 9–211. The syntax for the REGISTER_SNAPSHOT procedure is shown below.

```
DBMS_SNAPSHOT.REGISTER_SNAPSHOT(
      snapowner   IN   VARCHAR2,
      snapname    IN   VARCHAR2,
      snapsite    IN   VARCHAR2,
      snapshot_id IN   DATE | BINARY_INTEGER,
      flag        IN   BINARY_INTEGER,
      qry_txt     IN   VARCHAR2,
      rep_type    IN   BINARY_INTEGER := DBMS_SNAPSHOT.REG_UNKNOWN)
```

**Note:** This procedure is overloaded. The SNAPSHOT_ID and FLAG parameters are mutually exclusive.

***Table 9–211 Parameters for REGISTER_SNAPSHOT***

| Parameter | Description |
|---|---|
| sowner | The owner of the snapshot. |
| snapname | The name of the snapshot. |
| snapsite | The name of the snapshot site for a snapshot registering at an Oracle8 master. This parameter should not contain any double quotes. |
| snapshot_id | The identification number of the snapshot. Specify an Oracle8 snapshot as a BINARY_INTEGER; specify an Oracle7 snapshot registering at an Oracle8 master sites as a DATE. |
| flag | PL/SQL package variable indicating whether subsequent create or move commands are registered in the query text. |
| query_txt | The first 32,000 bytes of the query. |

*Table 9–211 Parameters for REGISTER_SNAPSHOT*

| Parameter | Description |
|---|---|
| rep_type | Version of the snapshot. Valid constants that can be assigned include reg_uknown (the default), reg_v7_group, reg_v8_group, and reg_repapi_group. |

**Note**: This procedure is executed at the master site, and can be done by a remote procedure call. If REGISTER_SNAPSHOT is called multiple times with the same SNAPOWNER, SNAPNAME, and SNAPSITE, the most recent values for SNAPSHOT_ID, FLAG, and QUERY_TXT are stored. If a query exceeds the maximum VARCHAR2 size, QUERY_TXT contains the first 32000 characters of the query and the remainder is truncated. When invoked manually, the values of SNAPSHOT_ID and FLAG have to be looked up in the snapshot views by the person who calls the procedure.

**Note:** If you do NOT want the snapshot query registered at the master site, call the SET_REGISTER_QUERY_TEXT procedure with the option set to FALSE. To see the most recent setting of the option, call the GET_REG_QUERY_TEXT_FLAG function at the snapshot site before issuing the DDL.

# DBMS_SNAPSHOT.SET_I_AM_A_REFRESH

## Purpose

To set the I_AM_REFRESH package state to the appropriate value.

## Syntax

The parameter for the SET_I_AM_A_REFRESH procedure is described in Table 9–212. The syntax for this procedure is shown below.

```
DBMS_SNAPSHOT.SET_I_AM_A_REFRESH(value IN BOOLEAN)
```

***Table 9–212 Parameter for SET_I_AM_A_REFRESH***

| Parameter | Description |
|---|---|
| value | Value that you want to set the I_AM_A_REFRESH package state to. If this state is set to TRUE, all local replication triggers for snapshots will be effectively disabled in this session because each replication trigger first checks this state. If this state is set to FALSE, these triggers will be enabled. |

# DBMS_SNAPSHOT.UNREGISTER_SNAPSHOT

## Purpose

To enable the administration of individual snapshots. Invoked at a master site to unregister a snapshot.

## Syntax

The parameters for the UNREGISTER_SNAPSHOT procedure are described in Table 9–213. The syntax for the UNREGISTER_SNAPSHOT procedure is shown below.

```
DBMS_SNAPSHOT.UNREGISTER_SNAPSHOT(
      snapowner      IN   VARCHAR2,
      snapname       IN   VARCHAR2,
      snapsite       IN   VARCHAR2)
```

***Table 9–213 Parameters for UNREGISTER_SNAPSHOT***

| Parameters | Description |
|---|---|
| snapowner | The owner of the snapshot. |
| snapname | The name of the snapshot. |
| snapsite | The name of the snapshot site. |

# Package Variables

Table 9–214 describes the package variables that are used by the advanced replication facility. You may need to check the value of one or more of these variables in your own packages or triggers.

***Table 9–214 Replication Package Variables***

| Variable | Type | Description |
|---|---|---|
| dbms_reputil.replication_is_on | BOOLEAN | TRUE indicates that the generated replication triggers are enabled. FALSE indicates that replication is disabled at the current site for the replicated object group. This variable is set by calling the REPLICATION_ON or REPLICATION_OFF procedures in the DBMS_REPUTIL package. |
| dbms_reputil.from_remote | BOOLEAN | This variable is set to TRUE at the beginning of procedures in the $RP replication packages, and is set to FALSE at the end of these procedures. You may need to check this variable if you have any triggers that could be fired as the result of an update by a $RP package. |
| dbms_reputil.global_name | VARCHAR2 (128) | This variable contains the global database name of the local database. |

# 10

# Data Dictionary Views

This chapter describes data dictionary views that can be useful to users of the advanced replication facility. The views are alphabetized within the following categories:

- Replication Catalog Views.
- Deferred Transaction Views.
- Snapshots and Snapshot Refresh Group Views.

# Replication Catalog Views

Whenever you install advanced replication capabilities at a site, Oracle installs the replication catalog, which consists of tables and views, at that site. As shown in Table 10–1, the views are used by master and snapshot sites to determine such information as what objects are being replicated, where they are being replicated, and if any errors have occurred during replication. *You should not modify the replication catalog tables directly; use the procedures provided in the DBMS_REPCAT package.*

Each view has three versions, which have different prefixes: USER_*, ALL_*, and SYS.DBA_* unless otherwise stated. This section ignores any differences among these views.

This section contains information about the following views:

- REPGROUP View
- REPCATLOG View
- REPCOLUMN View
- REPCOLUMN_GROUP View
- REPCONFLICT View
- REPDDL View
- REPGENERATED View
- REPGROUPED_COLUMN View
- REPKEY_COLUMNS View
- REPOBJECT View
- REPPARAMETER_COLUMN View
- REPPRIORITY View
- REPPRIORITY_GROUP View
- REPPROP View
- REPRESOLUTION View
- REPRESOL_STATS_CONTROL View
- REPRESOLUTION_METHOD View
- REPRESOLUTION_STATISTICS View

- REPSITES View
- REPGENOBJECTS View

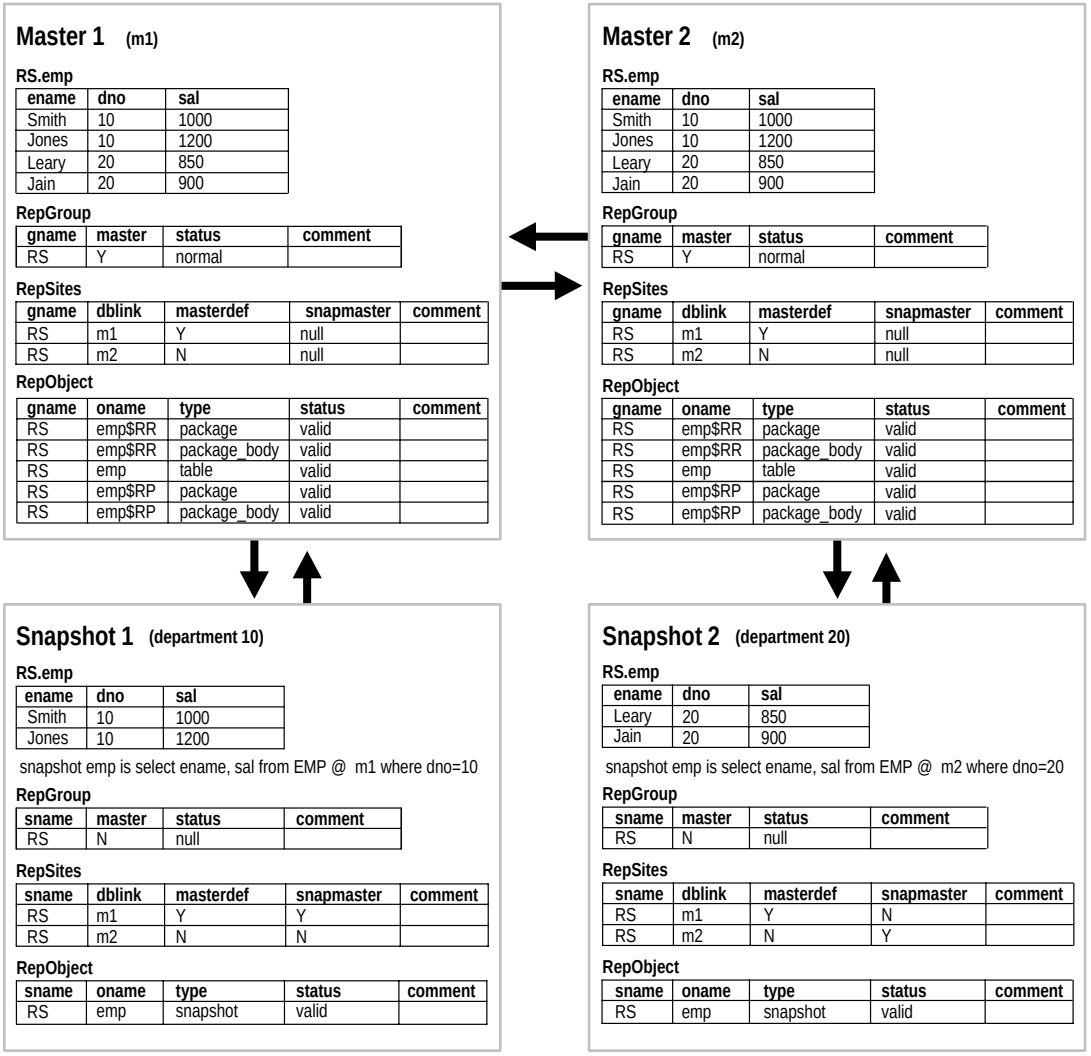
## REPGROUP View

The REPGROUP view lists all of the object groups that are being replicated. The members of each object group are listed in a different view, REPOBJECT.

***Table 10–1 REPGROUP View***

| Column | Description |
|---|---|
| sname | The name of the replicated schema. Obsolete with release 7.3 or later. |
| gname | The name of the replicated object group. |
| master | 'Y' indicates that this is a master site. 'N' indicates the current site is a snapshot site. |
| status | Used at master sites only. Status can be: normal, quiescing, or quiesced. |
| schema_Comment | Any user-supplied Comments. |

***Figure 10–1 Replication Catalog Views***



## REPCATLOG View

The REPCATLOG at each master site contains the interim status of any asynchronous administrative requests and any error messages generated. All messages encountered while executing a request are eventually transferred to the REPCATLOG at the master that originated the request. If an administrative request completes without error, ultimately all traces of this request are removed from the REPCATLOG view.

***Table 10–2 REPCATLOG View***

| Column | Description |
|---|---|
| id | A sequence number. Together, the ID and SOURCE columns identify all log records at all master sites that pertain to a single administrative request. |
| source | Location that the request originated. |
| userid | Userid of person making the request. |
| timestamp | When the request was made.7 |
| role | Indicates if site is the 'masterdef' or a 'master' site. |
| master | If the role is 'masterdef' and the task is remote, indicates which master is performing the task. |
| sname | The name of the schema for the replicated object, if applicable. |
| request | The name of the DBMS_REPCAT administrative procedure that was run. |
| oname | The name of the replicated object, if applicable. |
| type | The type of replicated object. |
| status | The status of the administrative request: ready, do_callback, await_callback, or error. |
| message | Any error message that has been returned. |
| errnum | The Oracle error number for the message. |
| gname | The name of the replicated object group. |

## REPCOLUMN View

The REPCOLUMN view lists the replicated columns for a table.

*Table 10–3 REPCOLUMN View*

| Column | Description |
|---|---|
| sname | The name of the object owner. |
| oname | The name of the object. |
| type | The type of the object. |
| cname | The name of the replicated column. |
| id | The id of the replicated column. |
| pos | The ordering of the replicated column. |
| compare_old_on_delete | Indicates whether Oracle sends and compares the old value of the column in replicated deletes. |
| compare_old_on_update | Indicates whether Oracle sends and compares the old value of the column in replicated updates. |

## REPCOLUMN_GROUP View

The REPCOLUMN_GROUP view lists all of the column groups that you have defined for each replicated table.

*Table 10–4 REPCOLUMN_GROUP View*

| Column | Description |
|---|---|
| sname | The name of the schema containing the replicated table. |
| oname | The name of the replicated table. |
| group_name | The column group name. |
| group_Comment | Any user-supplied Comments. |

**Note:** The sname column is not present in the USER_ view.

## REPCONFLICT View

The REPCONFLICT view displays the name of the table for which you have defined a conflict resolution method and the type of conflict that the method is used to resolve.

***Table 10–5 REPCONFLICT View***

| Column | Description |
|---|---|
| sname | The name of the schema containing the replicated table. |
| oname | The name of the table for which you have defined a conflict resolution method. |
| conflict_type | The type of conflict that the conflict resolution method is used to resolve: delete, uniqueness, or update. |
| reference_name | The object to which the routine applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name. |

**Note:** The sname column is not present in the USER_ view.

## REPDDL View

The REPDDL holds DDL for replication objects.

***Table 10–6 REPDDL View***

| Column | Description |
|---|---|
| log_id | Identifying number of the REPCAT log record. |
| source | Name of the database at which the request originated. |
| role | 'Y' if this database is the masterdef for the request; 'N' if this database is a master. |
| master | Name of the database that processes this request. |
| line | Ordering of records within a single request. |
| text | Portion of an argument or DDL text. |

## REPGENERATED View

The REPGENERATED view lists information about system-generated objects.

***Table 10–7 REPGENERATED View***

| Column | Description |
|---|---|
| sname | Owner of the object. |
| oname | Object name. |
| type | Object type. |
| base_sname | Owner of the base object. |
| base_oname | Object name of the base object. |
| base_type | Object type of the base object. |
| package_prefix | Prefix for package wrapper name. |
| procedure_prefix | prefix for procedure wrapper for procedures within a package wrapper. |
| distributed | 'Y' if generation is distributed;<br>'N' if generated objects are cloned.<br>Should always be 'Y' for Rep3 environments. |
| reason | The reason why this object was generated. |

**Note:** The REPGENERATED view is obsolete and is kept only for backward compatibility.

## REPGROUPED_COLUMN View

The REPGROUPED_COLUMN view lists all of the columns that make up the column groups for each table.

*Table 10–8 REPGROUPED_COLUMN View*

| Column | Description |
|---|---|
| sname | The name of the schema containing the replicated table. |
| oname | The name of the replicated table. |
| group_name | The name of the column group. |
| column_name | The name of the column in the column group. |

**Note:** The sname column is not present in the USER_ version of the view.

## REPKEY_COLUMNS View

The REPKEY_COLUMNS view lists information relating to the primary key column.

*Table 10–9 REPKEY_COLUMNS View*

| Column | Description |
|---|---|
| sname | Owner of the replicated table. |
| oname | Name of the replicated table. |
| col | ”Primary Key” column name in the table. |

## REPOBJECT View

The REPOBJECT view provides information about the objects in each replicated object group. An object can belong to only one object group. A replicated object group can span multiple schemas.

***Table 10–10 REPOBJECT View***

| Column | Description |
|---|---|
| sname | The name of the schema containing the replicated object. |
| oname | The name of the replicated object. |
| type | The type of replicated object: table, view, package, package body, procedure, function, index, synonym, trigger, or snapshot. |
| status | CREATE indicates that Oracle is applying user supplied or Oracle-generated DDL to the local database in an attempt to create the object locally. When a local replica exists, Oracle COMPAREs the replica with the master definition to ensure that they are consistent. When creation or comparison complete successfully, Oracle updates the status to VALID; otherwise, it updates the status to ERROR. If you drop an object, Oracle updates its status to DROPPED before deleting the row from the REPOBJECT view. |
| id | The identifier of the local database object, if one exists. |
| object_Comment | Any user supplied Comments. |
| gname | The name of the replicated object group to which the object belongs. |
| generation_status | Status of whether the object needs to generate replication packages. |
| min_communication | N = send both OLD and NEW values for an updated view. |
| trigflag | Inline trigger flag. Y= trigger activated; N = trigger disabled. |

## REPPARAMETER_COLUMN View

In addition to the information contained in the REPRESOLUTION view, the REPPARAMETER_COLUMN view also contains information about the columns that you indicated should be used to resolve the conflict. These are the column values that are passed as the LIST_OF_COLUMN_NAMES argument to the ADD_*_RESOLUTION procedures in the DBMS_REPCAT package.

***Table 10–11 REPPARAMETER_COLUMN View***

| Column | Description |
|---|---|
| sname | The name of the schema containing the replicated table. |
| oname | The name of the replicated table. |
| conflict_type | The type of conflict that the routine is used to resolve: delete, uniqueness, or update. |
| reference_name | The object to which the routine applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name. |
| sequence_no | The order that resolution methods are applied, with 1 being applied first. |
| method_name | The name of an Oracle-supplied conflict resolution method. For user-supplied methods, this value is 'user function'. |
| function_name | For methods of type 'user function', the name of the user-supplied conflict resolution routine. |
| priority_group | For methods of type 'priority group', the name of the priority group. |
| parameter_table_ name | Defaults to object name of PL/SQL table containing columns passed to conflict resolution function. |
| parameter_ column_name | The name of the column used as the IN parameter for the conflict resolution routine. |
| parameter_ sequence_no | Ordering of column used as IN parameter. |

**Note:** The SNAME column is not present in the USER_ view.

## REPPRIORITY View

The REPPRIORITY view displays the value and priority level of each priority group member. Priority group names must be unique within a replicated object group. Priority levels and values must each be unique within a given priority group.

***Table 10–12 REPPRIORITY View***

| Column | Description |
|---|---|
| sname | The name of the replicated schema. Obsolete with release 7.3 or later. |
| gname | The name of the replicated object group. |
| priority_group | The name of the priority group or site priority group. |
| priority | The priority level of the member. The highest number has the highest priority. |
| data_type | The datatype of the values in the priority group. |
| fixed_data_length | The maximum length of values of datatype CHAR. |
| char_value | The value of the priority group member, if data_type = char. |
| varchar2_value | The value of the priority group member, if data_type = varchar2. |
| number_value | The value of the priority group member, if data_type = number. |
| date_value | The value of the priority group member, if data_type = date. |
| raw_value | The value of the priority group member, if data_type = raw. |
| nchar_value | The value of the priority group member, if data_type = nchar. |
| nvarchar2_value | The value of the priority group member, if data_type = nvarchar2. |
| large_char_value | The value of the priority group member, for blank-padded character strings over 255 characters. |

**Note:** The SNAME and GNAME columns are not present in the USER_ view.

## REPPRIORITY_GROUP View

The REPPRIORITY_GROUP view lists the priority and site priority groups that you have defined for a replicated object group.

***Table 10–13 REPPRIORITY_GROUP View***

| Column | Description |
|---|---|
| sname | The name of the replicated schema. Obsolete with release 7.3 or later. Not shown in USER views. |
| gname | The name of the replicated object group. Not shown in USER views. |
| priority_group | The name of the priority group or site priority group. |
| data_type | The datatype of the values in the priority group. |
| fixed_data_length | The maximum length for values of datatype CHAR. |
| priority_Comment | Any user-supplied Comments. |

## REPPROP View

The REPPROP view indicates the technique used to propagate operations on an object to the same object at another master site. These operations may have resulted from a call to a stored procedure or procedure wrapper, or may have been issued against a table directly.

***Table 10–14 REPPROP View***

| Column | Description |
|---|---|
| sname | The name of the schema containing the replicated object. |
| oname | The name of the replicated object. |
| type | The type of object being replicated. |
| dblink | The fully qualified database name of the master site to which changes are being propagated. |
| how | How propagation is performed. Values recognized are 'none' for the local master site, and 'synchronous' or 'asynchronous' for all others. |
| propagate_ Comment | Any user-supplied Comments. |

## REPRESOLUTION View

The REPRESOLUTION view indicates the routines used to resolve update, unique or delete conflicts for each table replicated using row-level replication for a given schema.

***Table 10–15 REPRESOLUTION View***

| Column | Description |
|---|---|
| sname | The name of the replicated schema. |
| oname | The name of the replicated table. |
| conflict_type | The type of conflict that the routine is used to resolve: delete, uniqueness, or update. |
| reference_name | The object to which the routine applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name. |
| sequence_no | The order that resolution methods are applied, with 1 being applied first. |
| method_name | The name of an Oracle-supplied conflict resolution method. For user-supplied methods, this value is 'user function'. |
| function_name | For methods of type 'user function', the name of the user-supplied conflict resolution routine. |
| priority_group | For methods of type 'priority group', the name of the priority group. |
| resolution_ Comment | Any user-supplied Comments. |

**Note:** The SNAME column is not present in the USER_ view.

## REPRESOL_STATS_CONTROL View

The REPRESOL_STATS_CONTROL view lists information about statistics collection for conflict resolutions for all replicated tables in the database.

*Table 10–16 REPRESOL_STATS_CONTROL View*

| Column | Description |
|---|---|
| sname | Owner of the table. |
| oname | Table name. |
| created | Timestamp for which statistics collection was first started. |
| status | Status of statistics collection: ACTIVE, CANCELLED. |
| status_update_date | Timestamp for which the status was last updated. |
| purged_date | Timestamp for the last purge of statistics data. |
| last_purged_start_date | The last start date of the statistics purging date range. |
| last_purged_end_date | The last end date of the statistics purging date range. |

**Note:** The SNAME column is not present in the USER_ view.

## REPRESOLUTION_METHOD View

The REPRESOLUTION_METHOD view lists all of the conflict resolution routines available in your current database. Initially, this view lists the standard routines provided with the advanced replication facility. As you create new user functions and add them as conflict resolution methods for an object in the database, these functions are added to this view.

*Table 10–17 REPRESOLUTION_METHOD View*

| Column | Description |
|---|---|
| conflict_type | The type of conflict that the resolution routine is designed to resolve: update, uniqueness, or delete. |
| method_name | The name of the Oracle-supplied method, or the name of the user-supplied routine. |

## REPRESOLUTION_STATISTICS View

The REPRESOLUTION_STATISTICS view lists information about successfully resolved update, uniqueness, and delete conflicts for all replicated tables. These statistics are only gathered for a table if you have called DBMS_REPCAT.REGISTER_STATISTICS.

***Table 10–18 REPRESOLUTION_STATISTICS View***

| Column | Description |
|---|---|
| sname | The name of the replicated schema. |
| oname | The name of the replicated table. |
| conflict_type | The type of conflict that was successfully resolved: delete, uniqueness, or update. |
| reference_name | The object to which the conflict resolution routine applies. For delete conflicts, this is the table name. For uniqueness conflicts, this is the constraint name. For update conflicts, this is the column group name. |
| method_name | The name of an Oracle-supplied conflict resolution method. For user-supplied methods, this value is 'user function'. |
| function_name | For methods of type 'user function', the name of the user supplied conflict resolution routine. |
| priority_group | For methods of type 'priority group', the name of the priority group. |
| primary_key_value | A concatenated representation of the row's primary key. |
| resolved_date | Date on which the conflict for this row was resolved. |

**Note:** The SNAME column is not present in the USER_ view.

## REPSITES View

The REPSITES view lists the members of each replicated object group.

***Table 10–19 Columns in USER, ALL, and DBA REPSITES View***

| Column | Description |
|---|---|
| gname | The name of the replicated object group. |
| dblink | The database link to the master site for this object group. |
| masterdef | Indicates which of the dblinks is the master definition site. |
| snapmaster | Used by snapshot sites to indicate which of the dblinks to use when refreshing. |
| master_Comment | User-supplied Comments. |
| master | Is the site a master site for the replicated group? Y or N. |

The DBA_REPSITES view has the following additional columns:

| | |
|---|---|
| prop_updates | Number of requested updates for master. |
| my_dblink | Used to detect problem after import. If Y, the dblink is the global name. |

## REPGENOBJECTS View

The REPGENOBJECTS view describes objects generated to support replication.

***Table 10–20 REPGENOBJECTS View***

| Column | Description |
|---|---|
| Base_oname | The object for which this object was generated. |
| Base_sname | The base object's owner. |
| Base type | The type of the base object. |
| Distributed | Obsolete. |
| Oname | The name of the generated object. |
| Package-prefix | The prefix for the package wrapper. |
| Procedure-prefix | Not used. |
| Reason | The reason the object was generated. |
| Sname | The name of the replicated schema. |
| Type | The type of the generated object. |

## Deferred Transaction Views

Oracle provides several views for you to use in administering deferred transactions. These views provide information about each deferred transaction, such as the transaction destinations, the deferred calls that make up the transactions, and any errors encountered during attempted execution of the transaction. *You should not modify the tables directly; use the procedures provided in the DBMS_DEFER and DBMS_DEFER_SYS packages.*

- DEFCALL View
- DEFDEFAULTDEST View
- DEFERRCOUNT View
- DEFCALLDEST View
- DEFERROR View
- DEFLOB View
- DEFPROPAGATOR View
- DEFSCHEDULE View
- DEFTRAN View
- DEFTRANDEST View

## DEFCALL View

The DEFCALL view records all deferred remote procedure calls.

***Table 10–21 DEFCALL View***

| Column | Description |
|---|---|
| callno | Unique ID of call within a transaction. |
| deferred_tran_id | The unique ID of the associated transaction. |
| schemaname | The schema name of the deferred call. |
| packagename | The package name of the deferred call. |
| procname | The procedure name of the deferred call. |
| argcount | The number of arguments to the deferred call. |

## DEFCALLDEST View

The DEFCALLDEST view lists the destinations for each deferred remote procedure call.

***Table 10–22 DEFCALLDEST View***

| Column | Description |
|---|---|
| callno | Unique ID of call within a transaction. |
| deferred_tran_id | Corresponds to the deferred_tran_id in the DEFTRAN view. Each deferred transaction is made up of one or more deferred calls. |
| dblink | The fully qualified database name of the destination database. |

## DEFDEFAULTDEST View

If you are not using Oracle's replication facility and do not supply a destination for a deferred transaction or the calls within that transaction, Oracle uses the DEFDEFAULTDEST view to determine the destination databases to which you want to defer a remote procedure call.

***Table 10–23 DEFDEFAULTDEST View***

| Column | Description |
|---|---|
| dblink | The fully qualified database name to which to replicate a transaction. |

## DEFERRCOUNT View

The DEFERRCOUNT view provides information about the error transactions for a given destination.

***Table 10–24 DEFERRCOUNT View***

| Column | Description |
|---|---|
| errcount | Number of existing transactions that caused an error for the destination. |
| destination | Database link used to address destination. |

## DEFERROR View

The DEFERROR view provides the ID of each transaction that could not be applied. You can use this ID to locate the queued calls associated with this transaction. These calls are stored in the DEFCALL view. You can use the procedures in the DBMS_DEFER_QUERY package to determine the arguments to the procedures listed in the DEFCALL view.

***Table 10–25 DEFERROR View***

| Column | Description |
|---|---|
| deferred_tran_id | The ID of the transaction causing the error. |
| callno | Unique ID of call at deferred_tran_db. |
| destination | Database link used to address destination. |
| error_number | Oracle error number. |
| error_msg | Error message text. |
| receiver | Original receiver of the deferred transaction. |
| origin_tran_db | The database originating the deferred transaction. |
| origin_tran_id | The original ID of the transaction. |
| start_time | Time the original transaction was enqueued. |

## DEFLOB View

The DEFLOB view stores the LOB parameters to deferred RPCs.

***Table 10–26 DEFLOB View***

| Column | Description |
|---|---|
| id | Identifier of the LOB parameter. |
| deferred_tran_id | Trnsaction ID for deferred RPC with this LOB parameter. |
| blob_col | The binary LOB parameter. |
| clob_col | The character LOB parameter. |
| nclob_col | The national character LOB parameter. |

## DEFPROPAGATOR View

The DEFPROPAGATOR view displays information about the local propagator.

*Table 10–27 DEFPROPAGATOR View*

| Column | Description |
|---|---|
| username | Username of the propagator. |
| userid | User ID of the propagator. |
| status | Status of the propagator. |
| created | Time when the propagator was registered. |

## DEFSCHEDULE View

The DEFSCHEDULE view displays information about when a job is next scheduled to be executed.

*Table 10–28 DEFSCHEDULE View*

| Column | Description |
|---|---|
| dblink | Fully qualified pathname to master database site for which you have scheduled periodic execution of deferred remote procedure calls. |
| job | Number assigned to job when you created it by calling DBMS_DEFER_SYS.SCHEDULE_EXECUTION. Query the WHAT column of USER_JOBS view to determine what is executed when the job is run. |
| interval | Function used to calculate the next time to push the deferred transaction queue to destination. |
| next_date | Next date that job is scheduled to be executed. |
| last_date | Last time the queue was pushed (or attempted to push) remote procedure calls to this destination. |
| disabled | Is propagation to destination disabled? |
| last_txn_count | Number of transactions pushed during last attempt. |
| last_error_number | Oracle error number from last push. |
| last_error_message | Error message from last push. |

## DEFTRAN View

The DEFTRAN view records all deferred transactions.

***Table 10–29 DEFTRAN View***

| Column | Description |
|---|---|
| deferred_tran_id | The transaction ID that enqueued the calls. |
| delivery_order | An identifier that determines the order of deferred transactions in the queue. The identifier is derived from the system commit number of the originating or copying transaction. |
| destination_list | 'R' or 'D'. 'R' indicates that the destinations are determined by the REPSCHEMA view. 'D' indicates that the destinations were determined by the DEFDEFAULTDEST view or the NODE_LIST argument to the TRANSACTION, CALL, or COPY procedures. |
| start_time | The time that the original transaction was enqueued. |

## DEFTRANDEST View

The DEFTRANDEST view lists the destinations for a deferred transaction.

***Table 10–30 DEFTRANDEST View***

| Column | Description |
|---|---|
| deferred_tran_id | The transaction ID of the transaction to replicate to the given database link. |
| dblink | The fully qualified database name of the destination database. |
| delivery_order | An identifier that determines the order of deferred transactions in the queue. The identifier is derived from the system commit number of the originating or copying transaction. |

# Snapshots and Snapshot Refresh Group Views

The following views provide information about snapshots and snapshot refresh groups.

- SNAPSHOTS View
- REGISTERED_SNAPSHOTS View
- SNAPSHOTS_LOGS View
- SNAPSHOT_REFRESH_TIMES View
- REFRESH View
- REFRESH_CHILDREN View

## SNAPSHOTS View

The SNAPSHOTS catalog view lists information about all of the snapshots in a database.

***Table 10–31 SNAPSHOTS View***

| Column | Description |
|---|---|
| OWNER | Owner of the snapshot. |
| NAME | Name of the view used by users and applications for querying and updating the snapshot. |
| TABLE_NAME | Table in which the snapshot is stored (it has an extra column for the master rowid). |
| MASTER_VIEW | View of the master table, owned by the snapshot owner, used for refreshes. |
| MASTER_OWNER | Owner of the master table. |
| MASTER | Master table that this snapshot copies. |
| MASTER_LINK | Database link name to the master site. |
| CAN_USE_LOG | YES if this snapshot can use a snapshot log, NO if this snapshot is too complex to use a log. |
| UPDATABLE | 'YES' indicates snapshot is updatable; 'NO' indicates read-only. |
| LAST_REFRESH | Date and time at the master site of the last refresh. |
| ERROR | Number of failed attempts since last successful attempt. |
| TYPE | Type of refresh for all automatic refreshes: COMPLETE, FAST, FORCE. |
| NEXT | Date function used to compute next refresh dates. |
| START_WITH | Date function used to compute next refresh dates. |
| REFRESH_METHOD | Values used to drive a fast or complete refresh of the snapshot. |
| FR_OPERATIONS | If 'REGENERATE', then fast refresh operations have not been generated. |
| CR_OPERATIONS | If 'REGENERATE', then complete refresh operations have not been generated. |

*Table 10–31 SNAPSHOTS View*

| Column | Description |
|---|---|
| MASTER_ROLLBACK_SEG | Rollback segment to be used during refresh at the master. |
| REFRESH_GROUP | Group identifier for consistent refresh. |
| UPDATE_TRIG | Name of the trigger that fills in the UPDATE_LOG for an updatable snapshot. |
| UPDATE_LOG | Name of the table that logs changes to an updatable snapshot. |
| QUERY | Query used to create the snapshot. |

**Note:** UPDATE_TRIG: NULL in Oracle8 because of internalized triggers; MASTER_VIEW: NULL in Oracle8, now obsolete.

## REGISTERED_SNAPSHOTS View

This view describes local or remote snapshots of local tables.

*Table 10–32 REGISTERED_SNAPSHOTS View*

| Column | Description |
|---|---|
| OWNER | Owner of the snapshot. |
| NAME | The name of the snapshot. |
| SNAPSHOT_SITE | Global name of the snapshot site. |
| CAN_USE_LOG | If NO, the snapshot is complex and cannot fast refresh. |
| UPDATABLE | If NO, the snapshot is read only. |
| REFRESH_METHOD | Whether the snapshot uses ROWIDs or primary key for fast refresh. |
| SNAPSHOT_ID | Identifier for the snapshot used by the master for fast refresh. |
| QUERY_TXT | Query defining the snapshot, if registered. |
| VERSION | Version of the snapshot. |

## SNAPSHOTS_LOGS View

The SNAPSHOT_LOGS view describes all the snapshot logs in the database.

***Table 10–33 SNAPSHOT_LOGS View***

| Column | Description |
|---|---|
| LOG_OWNER | Owner of the snapshot log. |
| MASTER | Name of the master table for which changes are logged. |
| LOG_TABLE | Log table; with ROWIDs and timestamps of rows which changed in the master. |
| LOG_TRIGGER | An after-row trigger on the master which inserts rows into the log. NULL in Oracle 8 because of internalized triggers. |
| ROWIDS | If YES, the snapshot log records rowid information. |
| PRIMARY_KEY | If YES, the snapshot log records primary key information. |
| FILTER_COLUMNS | If YES, the snapshot log records filter column information. |
| CURRENT_SNAPSHOTS | One date per snapshot -- the date the snapshot of the master last refreshed. |
| SNAPSHOT_ID | Unique identifier of the snapshot. |

**Note:** The view shows one row for each snapshot using the log. However, there is only one for all the snapshots located at the master site. To find out which logs are used, query USER_SNAPSHOT_LOGS using unique and not selecting SNAPSHOT_ID and CURRENT_SNAPSHOTS.

## SNAPSHOT_REFRESH_TIMES View

The REFRESH_TIMES view lists the date and time of the last refresh.

***Table 10–34 SNAPSHOT_REFRESH _TIMES View***

| Column | Description |
|---|---|
| OWNER | Owner of the snapshot. |
| NAME | Name of the snapshot view. |
| MASTER_OWNER | Owner of the master table. |
| MASTER | Name of the master table. |
| LAST_REFRESH | The date and time of the last refresh. |

## REFRESH View

The REFRESH view lists each refresh group in the database, and describes refresh intervals for each group.

***Table 10–35 REFRESH View***

| Column | Description |
|---|---|
| ROWNER | Owner of the refresh group. |
| RNAME | Name of the refresh group. |
| REFGROUP | ID number of the refresh group. |
| IMPLICIT_DESTROY | Implicit delete flag. If this value is Y, Oracle deletes the refresh group after you have subtracted the last member from the group. |
| JOB | ID number of the job used to execute the automatic refresh of the snapshot group. You can use this information to query the USER_JOBS view for more information about the job. |
| NEXT_DATE | Date when the members of the group will next be refreshed. |
| INTERVAL | The function used to calculate the interval between refreshes. |
| BROKEN | Flag used to indicate a problem with the refresh group. If the value of the broken flag is Y, Oracle will not refresh the group, even if it is scheduled to be refreshed. |
| PUSH_DEFERRED_RPC | If Y, push changes to master before refresh. |
| REFRESH_AFTER_ERRORS | If Y, proceed with refresh despite errors when pushing deferred RPCs. |
| ROLLBACK_SEG | Name of rollback segment used at the snapshot site during refresh. |
| PURGE_OPTION | The method for purging the transaction queue after each push. |
| PARALLELISM | The level of parallelism for transaction propagation. |
| HEAP_SIZE | The heap size used for transaction propagation. |

## REFRESH_CHILDREN View

The REFRESH_CHILDREN view lists the members of each refresh group owned by the user, and includes information about the refresh interval for each member.

***Table 10–36 REFRESH_CHILDREN View***

| Column | Description |
|---|---|
| OWNER | Owner of the refresh group member. |
| NAME | Name of the refresh group member. |
| TYPE | Type of the refresh group member; for example, SNAPSHOT. |
| ROWNER | Owner of the refresh group. |
| RNAME | Name of the refresh group. |
| REFGROUP | ID number of the refresh group. |
| IMPLICIT_DESTROY | Implicit delete flag. If this value is Y, Oracle deletes the refresh group after you have subtracted the last member from the group. |
| JOB | ID number of the job used to execute the automatic refresh of the snapshot refresh group. You can use this information to query the USER_JOBS view for more information about the job. |
| NEXT_DATE | Date when the members of the group will next be refreshed. |
| INTERVAL | The function used to calculate the interval between refreshes. |
| BROKEN | Flag used to indicate a problem with the refresh group. If the value of the broken flag is Y, Oracle will not refresh the group, even if it is scheduled to be refreshed. |
| PUSH_DEFERRED_RPC | If Y, push changes to master before refresh. |
| REFRESH_AFTER_ERRORS | If Y, proceed with refresh despite errors when pushing deferred RPCs. |
| ROLLBACK_SEG | Name of rollback segment used at the snapshot site during refresh. |
| PURGE_OPTION | The method for purging the transaction queue after each push. |
| PARALLELISM | The level of parallelism for transaction propagation. |
| HEAP_SIZE | The heap size used for transaction propagation. |

# A

# New Features

This Appendix briefly describes the new replication features of the Oracle8 Server and provides pointers to other chapters in this document where you can get additional information.

New features include:

- Performance Enhancements.
- Data Subsetting Based on Subqueries.
- Enhanced, System-Based Security Model.
- New Replication Manager Features.

# Performance Enhancements

Oracle8 provides significant performance improvements based on the following new features.

## Parallel Propagation of Deferred Transactions

Dramatically improves throughput performance by parallelizing the propagation of a replication transaction stream while maintaining consistency and transaction dependencies.

**Additional Information:** See "Deciding between Serial and Parallel Propagation" on page 3-11.

## Internalized Replication Triggers

Internal triggers:

- Improve response time performance.
- Reduce processing overhead.
- Require less administration.

**Additional Information:** See "Oracle's Advanced Replication Architecture" on page 1-20.

## Reduced Data Propagation

Oracle8 reduces the amount of replicated data propagated over the network. Propagation can be reduced to only the following: new values of columns updated; old values of columns needed for conflict detection and resolution; and primary key values. This feature is important for performance when replicating LOBs.

**Additional Information:** See "Minimizing Data Propagation for Update Conflict Resolution" on page 5-40.

# Data Subsetting Based on Subqueries

Snapshots defined with certain types of subqueries can now be fast refreshed. This enables subsets of data to be easily defined and maintained. This feature is important for mass deployment applications, such as salesforce automation and branch automation.

**Additional Information**: See "Creating Snapshots with Subqueries" on page 2-16.

# Large Object Datatypes (LOBs) Support

Oracle8 supports the replication of the following types of large objects:

- Binary LOBs (BLOBs).
- Character LOBs (CLOBs).
- National language support character LOBs (NCLOBs).

**Additional Information**: See the following sections: “Datatype Considerations for Replicated Tables” on page 3-20, “Datatype Considerations for Snapshots” on page 4-18.

# Improved Management and Ease of Use

Oracle8 facilitates database management with the following features.

## Fine grained Quiesce

Replication master groups can now be individually quiesced without impacting other replication groups. Master groups can continue to replication updates while other master groups are quiesced.

**Additional Information:** See “Suspending Replication Activity for a Master Group” on page 3-17.

## Primary Key Snapshots

Primary key snapshots allow you to reorganize master tables while preserving fast refresh capability. Oracle8 adds primary key snapshots as the default, and continues to support ROWID snapshots.

**Additional Information:** See “Creating Simple Snapshots” on page 2-10 and Appendix B, “Migration and Compatibility”.

## Snapshot Registration at Master Sites

Oracle8 automatically registers information about a snapshot s at master sites. This facilitates monitoring and distributed administration.

**Additional Information:** See “Registering a Snapshot at its Master Site” on page 2-35.

### Reorganizing Tables With Capability of Fast Refresh

Oracle8 provides utilities to enable you to "reorg" master tables while preserving the consistency of master snapshot logs.

**Additional Information:** See "Master Table Reorganization and ROWID Snapshots" on page 2-43.

### Support for Offline Instantiation

Offline instantiation of schemas and database using Export/Import is now more automatic.

**Additional Information:** See "Snapshot Cloning and Offline Instantiation" on page 7-14.

### Deferred Constraints for Updatable Snapshots

Updatable snapshots now support declarative referential and uniqueness constraints.

**Additional Information:** See "Replicating Object Definitions to Master Sites" on page 3-21.

### Validate Procedure

A new procedure validates the correctness of multiple master environments to help avoid errors.

**Additional Information:** See "Validating a Master Group" on page 6-2.

### Partitioned Tables and Indexes

Oracle8 supports the replication of partitioned tables and indexes. You can use this feature if you want the replicated table to have the same partitions as the table at the master definition site.

## Enhanced, System-Based Security Model

Oracle8 system-based security model:

- Improves consistency: a single system-level model operates the same in both synchronous and asynchronous environments.
- Improves reliability: transactions are less likely to fail due to the lack of privileges at the receiving sites.

- Simplifies links: a single user can act as repadmin and repsys.
- Eliminates the need for a "user-level" model.
- One or more snapshot owners can perform snapshot refresh.

**Additional Information:** See "Preparing for Multimaster Replication" on page 3-4, "Preparing for Snapshot Site Replication" on page 4-5, and Appendix B, "Migration and Compatibility".

## New Replication Manager Features

Replication Manager now includes several wizards to help you configure your system quickly.

- Setup of multimaster configurations, including account, schema, and link creation.
- Setup of snapshot site configurations, including account, schema, and link creation.
- Setup of snapshot groups.
- Setup of snapshots.

Oracle Replication Manager includes support for most new features of Oracle8 Server replication functionality, including:

- Automatic recognition of each site's database version and dynamic configuration to manage both 8.x and 7.3 databases.
- Support for managing snapshot logs.
- Easy entry of date and interval expressions for jobs, refresh groups, and scheduled links.
- For Oracle8 databases, a flag to indicate that a table requires regeneration of replication support.
- Support for registering a site's replication administrator and propagator.
- A database objects folder to display database objects for master groups.
- Drag and drop functionality for master groups and snapshot refresh groups.
- Support for the validation of a master group at all master sites.

# B

# Migration and Compatibility

This Appendix explains the steps that need to be taken to migrate a replication environment from Oracle7 to Oracle8. Topics covered include:

- Migration Overview.
- Migrating All Sites at Once.
- Incremental Migration.
- Upgrading to Primary Key Snapshots.
- Features Requiring Migration to Oracle8.
- Obsolete procedures.

# Migration Overview

In some cases you may find it easiest to migrate your environment, particularly the multimaster component of your environment, in one step. Typically, this will only be possible for small configurations. Instead, you may wish to migrate an existing Oracle7 replication environment to Oracle8 incrementally. Replication and administrative operations can be run successfully in a mixed Oracle7 and Oracle8 replication environment.

To successfully interoperate, however, you must observe the following restrictions:

- Oracle8 snapshot sites can only interact with Oracle7 Release 7.3.3 or greater master sites.
- Oracle8 master sites can only interact with Oracle7 Release 7.3.4 or greater snapshot sites and with Oracle7 Release 7.3.3 or greater master sites.

After migrating a master site to Oracle8, perform a full refresh of all of associated snapshot sites.

Downgrading from Oracle8 to Oracle7 is not supported.

Certain Oracle8 replication features require that all sites be successfully migrated to Oracle8 before the features can be used. For example, before you can use primary key snapshots, both the snapshot site and its associated master site must be migrated to Oracle8. The Oracle8 simple snapshots with subqueries feature and the master table reorganization procedures require that you first upgrade from Rowid snapshots to primary key snapshots.

Migration using a full database export from Oracle7 and import to Oracle8 is also supported.

# Migrating All Sites at Once

This section describes how to migrate your entire multimaster environment at once to Oracle8. Note that any snapshot sites that you do not also migrate to Oracle8, must be upgraded to Oracle7 Release 7.3.4 or greater.

Follow these steps to migrate all master sites and (optionally) snapshot sites at one time:

1. Quiesce the replication environment by executing DBMS_REPCAT. SUSPEND_MASTER_ACTIVITY at the master definition site for all master replication groups, and stopping all propagation and refreshing from snapshot sites to the master, for example, by temporarily suspending or “breaking” entries in the job queue that control automated propagation and refreshing at

the snapshot sites. You must also resolve and re-execute any errors in the local error queue until it is empty. For more information see the following sections in *Oracle 7 Server Distributed Systems, Volume II: Replicated Data:* Chapter 4, "Asynchronous Propagation of DML Changes", and "Suspending Replication Activity", as well as Chapter 7, "Resolving an Error Manually".

2. Migrate all master sites using the Oracle7 to Oracle8 Migration Utility and replication CATREP8M.SQL script as documented in *Oracle8 Migration*.

3. Using the Replication Manager setup wizard, create a primary master replication administrator account granting this user Oracle8 Replication Administrator, Propagator, and Receiver privileges on all master sites, and set up the appropriate links connecting all sites. See "Preparing for Multimaster Replication" on page 3-4.

4. Using Replication Manager or the replication management API, regenerate replication support for each replication base object. See "Generating Replication Support for Master Group Objects" on page 3-26 for more information. Among other activities, generating replication support will establish the registered propagator as the owner of generated objects

5. Using Replication Manager or the replication management API, resume replication activity by unquiescing the environment. See "Resuming Replication Activity for a Master Group" on page 3-19 for more information.

6. At a minimum, you must now upgrade all associated snapshot sites to Oracle7 Release 7.3.4. For instructions on migrating your snapshot sites to Oracle8, see "Incremental Migration of Snapshot Sites" on page B-5.

7. All snapshots at all snapshot sites will need a full refresh after their master sites have been migrated to Oracle8. Before the refresh, be certain that you have "unbroken" any jobs that you may have "broken" during migration of your snapshot sites by calling the DBMS_JOB.BROKEN procedure.

   If your snapshots have been defined with the refresh "FORCE" option, their next attempted refresh will full refresh automatically. Snapshots defined with the refresh "FAST" option will need to be manually refreshed using dbms_refresh.refresh or other refresh procedures.

   If you are using procedural replication at snapshot sites, also regenerate snapshot support on all packages and package bodies used for procedural replication.

   **Note:** If you are migrating all of the master's snapshot sites to Oracle8 when the master site is migrated to Oracle8, in other words, you do not need to migrate the snapshot sites incrementally, you can alternatively drop the

snapshot logs for the master and recreate them as primary key snapshot logs. The snapshots at each snapshot site should be altered to convert them to primary key snapshots. You can then do a full refresh for each primary key snapshot. See "Upgrading to Primary Key Snapshots" on page B-10 for additional details.

8. Drop any administrative accounts and links that you were using to maintain your Oracle7 multimaster replication environment that are not needed in your Oracle8 environment. Unnecessary privileges may also be revoked. Be careful not to drop accounts that are needed to maintain any Oracle7 snapshot sites.

## Incremental Migration

It is possible to incrementally migrate your replication environment. However, you must carefully analyze the interdependencies between sites to ensure that they will continue to interoperate throughout your migration. Table B–1 describes the conditions that must be met to allow Oracle7 and Oracle8 replication sites to interoperate.

***Table B–1 Interoperability in a Replication Environment***

| Environment | Action | Pre-Requisite |
|---|---|---|
| Multimaster | Migrate master site from Oracle7 to Oracle8. | All other master sites must be Oracle7 Release 7.3.3 or greater. |
| Master with dependent snapshots | Migrate master site from Oracle7 to Oracle8. | All dependent snapshot sites must be Oracle7 Release 7.3.4 or greater. |
| Master with dependent snapshots | Migrate snapshot site from Oracle7 to Oracle8. | Associated master site must be Oracle7 Release 7.3.3 or greater. |

To avoid interoperability problems within a replication environment, it is strongly recommended that if you must perform an incremental migration that you perform it in the following order:

1. Upgrade all of your master sites to Oracle7 Release 7.3.3 or greater and follow the steps in "Preparing Oracle7 Master Sites for Incremental Migration" on page 5 to prepare your Oracle7 master sites for incremental migration.
2. Incrementally migrate all snapshot sites to Oracle8.
3. Incrementally migrate all master sites to Oracle8.

## Preparing Oracle7 Master Sites for Incremental Migration

Before beginning incremental migration of Oracle7 master or snapshot sites, your Oracle7 Release 7.3.3 or greater master sites must be configured so that all replication administration and propagation is done within the security context of a single user at each site. Additionally, this primary master replication administrator must have the same username and password at all Oracle7 and Oracle8 sites. Your Oracle7 master sites may already be configured in this manner. If not, you must complete the following steps:

1. Choose a primary master replication administrator for your replication environment. You may select your current replication administrator or create a new user.
2. At each master site, grant the required privileges to the primary master replication administrator using both DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_REPGROUP and DBMS_REPCAT_AUTH.GRANT_SURROGATE_REPCAT.
3. If they do not already exist, you must create the following links for from each master site to all other master sites in the multimaster environment (for a total of 3N(N - 1) links):
   - A public database link, created as SYS, that includes a valid global database name, as well as a USING clause with a valid SQL*Net 2.3 TNS alias.
   - A private database link, created as SYS, that includes a valid global database name, as well as a CONNECT TO clause with the username and password of the primary master replication administrator.
   - A private database link, create as the primary replication administrator, that includes a valid global database name, as well as a CONNECT TO clause with the username and password of the primary master replication administrator.

## Incremental Migration of Snapshot Sites

Before you can migrate a snapshot site to Oracle8, its associated master site must have been upgraded to Oracle7 Release 7.3.3 or greater and the master site must have been fully prepared for incremental migration.

To incrementally migrate your Oracle7 snapshot sites to Oracle8, complete the following steps:

1. Isolate the snapshot site from the replication environment by stopping all local updates to updatable snapshots at the snapshot site (in a separate session you may lock each snapshot's base table to prevent further transactions). Empty the local deferred transaction queue by pushing the queue to the snapshot's master. For more information refer to the *Oracle7 Server Distributed Systems, Volume II: Replicated Data*, Chapter 4, "Asynchronous Propagation of DML Changes". Stop all propagation from the snapshot site to its master, for example, by temporarily suspending or "breaking" entries in the job queue that control automated propagation and refreshing at the snapshot sites.
2. Run the Oracle7 to Oracle8 Migration Utility and replication CATREP8M.SQL script as documented in *Oracle8 Server Migration*.
3. Use the Replication Manager setup wizard or execute the appropriate replication management API calls to configure the primary snapshot replication administrator as the replication administrator and propagator for the snapshot site, to configure a receiver account at the associated master, and to create the appropriate links to the master. For Oracle7 master sites your receiver at the master site must be the primary master replication administrator that you prepared in the previous section. If you are using the Replication Manager setup wizard select the customize option to specify this receiver.
4. Using Replication Manager or the appropriate replication management API calls, regenerate snapshot replication support. See "Regenerating Replication Support for an Updatable Snapshot" on page 4-20 for more information. Among other activities, generating replication support establishes the registered propagator as the owner of generated objects
5. Using Replication Manager or the appropriate replication management API calls, reschedule propagation and/or refresh intervals with the master and enable local updates where appropriate. If you used the DBMS_JOB.BROKEN procedure to help isolate your master site in Step 1, you need to "unbreak" your jobs to resume your replication activity from your snapshot sites.
6. Drop any administrative accounts and links that you were using to maintain your Oracle7 replication environment that are not needed in your Oracle8 environment. Unnecessary privileges may also be revoked.

## Incremental Migration of Master Sites

Before upgrading a master site from Oracle7 to Oracle8, you must meet the following conditions:

- All other master sites in a multimaster environment must be running Oracle7 Release 7.3.3 or greater.

- You must have completed the instructions in "Preparing Oracle7 Master Sites for Incremental Migration" on page B-5.
- Any dependent snapshot sites must be running Oracle8 or Oracle7 Release 7.3.4 or greater.

To incrementally migrate your Oracle7 master sites to Oracle8, complete the following steps:

1. Pick a master site to migrate. You should migrate your master definition site first.
2. If you are using procedural replication, record the configuration information and locations (schemas) of existing procedure wrappers. This information will be used later.
3. Isolate the master site from the replication environment. To do this you must:
   - Stop updates to the master site by either:

     calling DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY at the master definition site for all master replication groups,

     or by calling DBMS_DEFER_SYS.UNSCHEDULE_EXECUTION (for Oracle7 sites) or DBMS_DEFER_SYS.UNSCHEDULE_PUSH (for Oracle8 sites) at every remote master site and dependent snapshot site, and by preventing update activity at the master site being migrated. You should also refrain from executing any administrative operations at the master definition site that may affect the master site being migrated.
   - Resolve and re-execute any errors in the local error queue until it is empty.
   - Stop any refreshes of the dependent snapshot sites from occurring by "breaking" entries in the job queue at each snapshot site that control automated propagation and refreshing at the snapshot sites.
   - For more information on completing the tasks in Step 1 refer to the following sections in *Oracle7 Server Distributed Systems, Volume II: Replicated Data:* Chapter 4, "Asynchronous Propagation of DML Changes", "Suspending Replication Activity", "Removing a Master Site from the Deferred Push List", and "Forcing the Deferred Transaction Queue to Push List". Also see Chapter 7, "Resolving an Error Manually".
4. Migrate the master site using the Oracle7 to Oracle8 Migration Utility and replication CATREP8M.SQL script as documented in *Oracle8 Migration*.
5. Using the Replication Manager setup wizard or the replication management API, grant your primary master replication administrator Oracle8 Primary

Replication Administrator, Propagator, and Receiver privileges for the master site. See “Preparing for Multimaster Replication” on page 3-4 for more information. Database links from the primary replication administrator to the primary master replication administrator at all other Oracle7 and Oracle8 master sites should already exist if you prepared your Oracle 7 master site for compatibility with Oracle8 using the directions in “Preparing Oracle7 Master Sites for Incremental Migration” on page B-5.

6. If you are not already in a quiesced state, use Replication Manager **o**r the replication management API to suspend all replication activity for all master groups. See “Suspending Replication Activity for a Master Group” on page 3-17 for more information.

7. Using Replication Manager or the replication management API, regenerate replication support for each replicated object. See “Generating Replication Support for Master Group Objects” on page 3-26 for more information. If any sites in the replication environment are still running Oracle7, you must set the “min_communication” parameter to FALSE. The “min_communication” parameter should only be set to TRUE (the default) once all sites have been migrated to Oracle8. For more information, see “Minimizing Data Propagation for Update Conflict Resolution” on page 5-40. Among other activities, generating replication support will establish the registered propagator as the owner of generated objects

8. If you are using procedural replication, check your remaining Oracle7 master sites to determine whether the wrappers have been moved (list created from Step 2). If they have been moved, create a synonym in their old location (in the schema of either the replication administrator or the table owner, depending on whether the site previously used the system-based or user-based model) pointing to the new location in the schema of the primary replication administrator. Confirm necessary object privileges have been granted to access the new owner and locations. If you are also using procedural replication at snapshot sites, regenerate snapshot support on all packages and package bodies used for procedural replication.

9. Using Replication Manager or the replication management API, resume replication activity and unquiesce the environment for each master group. See “Resuming Replication Activity for a Master Group” on page 3-19 for more information. If you have isolated the master by unscheduling propagation to other masters and from other masters then reschedule propagation by executing DBMS_DEFER_SYS.SCHEDULE_EXECUTION (for Oracle7 sites) or following the instructions in“Editing a Scheduled Link” on page 3-11(for Oracle8 sites) for all master sites.

10. All snapshots at both Oracle7 and Oracle8 snapshot sites will need a full refresh after their master site has been migrated to Oracle8. Because of the new Oracle8 rowid format, the Oracle7 to Oracle8 migration utility truncates all master snapshot logs. If you used the DBMS_JOB.BROKEN procedure to help isolate you master site in Step 3, "unbreak" your jobs to resume your replication activity from your snapshot sites.

    If your snapshots have been defined with the refresh "FORCE" option, their next attempted refresh will full refresh automatically. Snapshots defined with the refresh "FAST" option will need to be manually refreshed using dbms_refresh.refresh or other refresh procedures.

    **Note:** If you are able to migrate all of the master's snapshot sites to Oracle8 when the master site is migrated to Oracle8, (that is, you do not need to migrate the snapshot sites incrementally) you can alternatively drop the snapshot logs for the master and recreate them as primary key snapshot logs. The snapshots at each snapshot site should be altered to convert them to primary key snapshots. You can then do a full refresh for each primary key snapshot. See "Upgrading to Primary Key Snapshots" on page B-10 for additional details.

11. Drop any administrative accounts and links that you were using to maintain your Oracle7 multimaster replication environment that are not needed in your Oracle8 environment. Unnecessary privileges may also be revoked. Be careful not to drop accounts that are needed to maintain any Oracle7 snapshot sites or master sites.

## Migration Using Export/ Import

Full database export from Oracle7 Release 7.3.3 or greater and import to Oracle8 is supported for both masters and snapshots. You may use export/import as an alternative to the Oracle7 to Oracle8 Migration Utility and replication CATREP8M.SQL script in the procedures described above. Be sure that you follow all the steps, both before and after the actual migration from Oracle7 to Oracle8, in the above procedures however.

To export a full database from Oracle7 Release 7.3.3 or greater and import to Oracle8, follow these steps:

1. Export the Oracle7 Release 7.3.3 or greater database to a dump file using the Release 7.3 export utility under the SYSTEM schema with FULL=y.
2. Start up an Oracle8 database with advanced replication installed. Connect as SYSTEM and execute DBMS_REPCAT_MIG.PRE_IMPORT. This procedure

temporarily disables referential constraints on the replication data dictionary tables.

3. Import the dump file to the Oracle8 database using the Oracle8 import utility under the SYSTEM schema with FULL=y. The referential constraints on the replication data dictionary tables will be automatically enabled.

You may also export data from individual Oracle7 tables, import the data to Oracle8 tables, and then configure those tables as masters in an Oracle8 replication environment using standard advanced replication procedures.

See the *Oracle8 Utilities* reference guide for more information.

# Upgrading to Primary Key Snapshots

Once a snapshot site and its master have been migrated to Oracle8, you can upgrade your rowid snapshots to Oracle8 primary key snapshots. To do this you must first alter the snapshot logs for each master table to log primary key information, as well as rowid information, when master rows are updated. Once this is completed at your master site(s), you can incrementally convert your Oracle8 snapshots sites by altering the snapshots to convert them to primary key snapshots. Oracle8 masters that have been altered to log primary key as well as rowid information can support Oracle7 rowid snapshots as well as Oracle8 rowid and primary key snapshots simultaneously to allow for incremental migration.

**Note:** A primary key snapshot cannot be converted or downgraded to a rowid snapshot.

## Primary Key Snapshots Conversion at Master Site(s)

To support primary key snapshots, do the following at the Oracle8 master site:

1. Define and enable a primary key constraint on each master table that does not already have a primary key constraint enabled.
2. Alter the snapshot log for each master table supporting fast refresh to include primary key information using the ALTER SNAPSHOT LOG command. See ALTER SNAPSHOT LOG in the *Oracle8 SQL Reference* manual for additional information.

**Note:** If the above conditions are not met an error will be raised when you execute the ALTER SNAPSHOT command at the snapshot sites to convert to primary key snapshots.

### Primary Key Snapshot Conversion at Snapshot Site(s)

After the Oracle8 master site has been configured to support primary key snapshots, do the following at the Oracle8 snapshot sites:

1. Isolate the snapshot site from the replication environment by stopping all local updates to updatable snapshots at the snapshot site.
2. If any read-only ROWID snapshots being converted to primary key snapshots do not include all the columns of the primary key, drop and recreate them with all the primary key columns. See "Creating Simple Snapshots" on page 2-10 for more information.

**Note:** Constraints should not be defined on Rowid snapshots.

3. Perform a fast refresh of all snapshots to remove the need for any remaining rowid references in the master snapshot log.
4. Use the ALTER SNAPSHOT command to convert rowid snapshots to primary key snapshots. For complete syntax information, see the book *Oracle8 SQL Reference*.
5. Resume replication by rescheduling propagation and / or snapshot refresh with the master, enabling local updates where appropriate. If you used the DBMS_JOB.BROKEN procedure to help isolate you master site in Step 1, you need to "unbreak" your jobs to resume your replication activity from your snapshot sites.

## Features Requiring Migration to Oracle8

The following features require that all the sites involved be successfully migrated to Oracle8:

1. Replication of LOB data types.
2. Reduced data propagation.
   - Use the min_communication parameter and,
   - the send_and_compare_old_values procedure.
3. Parallel propagation of deferred transactions.
4. Global authentication and privileged database links.
5. Validate procedure.

**Additional Information:** See Appendix A, "New Features".

The following features require that all the sites involved must be successfully migrated to Oracle8 and primary key snapshots:

1. Simple snapshots with subqueries.
2. Master table reorganization procedures.

The following features will automatically work in mixed Oracle7 and Oracle8 environments, but only affect Oracle8 sites:

1. Fine grained quiesce.
2. Snapshot registration.

**Note:** All master groups at Oracle7 sites will be quiesced if any master group at that site is quiesced.

**Note:** Oracle7 snapshots will not be automatically registered at Oracle8 sites but can be manually registered using the DBMS_SNAPSHOT.REGISTER_ SNAPSHOT and DBMS_SNAPSHOT.UNREGISTER_SNAPSHOT procedures at the master site(s). See “Registering a Snapshot at its Master Site” on page 2-35 for more information.

## Obsolete procedures

Procedures that are obsoleted in Oracle8 include:

DBMS_REPCAT_ADMIN.GRANT_ADMIN_REPGROUP

DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_REPGROUP

DBMS_REPCAT_ADMIN.REVOKE_ADMIN_REPGROUP

DBMS_REPCAT_ADMIN.REVOKE_ADMIN_ANY_REPGROUP

DBMS_REPCAT_AUTH.GRANT_SURROGATE_REPCAT

DBMS_REPCAT_AUTH.REVOKE_SURROGATE_REPCAT

DBMS_DEFER_SYS.EXECUTE

# Index

## A


accounts
  creating for snapshots, 2-4
Add Objects to Group dialog
  Replication Manager, 3-19
ADD procedure
  DBMS_REFRESH package, 2-38
ADD_MASTER_DATABASE procedure
  DBMS_REPCAT package, 3-25
adding
  columns in column groups, 5-14
  priority group members, 5-24
  site priority group members, 5-29
  sites
    to an advanced replication environment, 7-8
additive prebuilt conflict resolution method, 5-16
Admin Requests folder, 6-5
Administration folder, 6-9, 6-11
administration requests, 1-19, 6-4 to 6-8
  applying, 6-5
  deleting, 6-6
  diagnosing problems with, 6-8
  displaying, 6-5
administrator
  account
    snapshot sites, 4-5, 4-7
administrator accounts
  master sites, 3-5
advanced replication, 3-1 to 7-35
  asynchronous propagation, 1-21
  deferred transactions and, 1-22
  disabling, 7-33
  hybrid configurations, 1-16
  job queues and, 1-22
  multimaster, 3-1 to 3-35
  multimaster configuration, 1-14
  overview (see also "basic replication"), 1-12 to 1-30
  procedural replication, 1-28
  row-level replication, 1-20
  RPCs and, 1-22
  snapshot sites
    offline instantiation of, 7-17
    replication, 4-1 to 4-24
    security setup, 7-24
  synchronous propagation, 1-28, 7-6
  techniques, 7-1 to 7-35
  transaction propagation protection, 3-33 to 3-34
  updatable snapshots, 1-15
  uses for, 1-13
advanced replication systems
  monitoring, 6-4
Alter Replication Object dialog
  Replication Manager, 3-23
ALTER SNAPSHOT command
  REFRESH clause, 2-44
ALTER SNAPSHOT LOG command, 2-28
ALTER_MASTER_REPOBJECT procedure
  DBMS_REPCAT package, 3-23
ALTER_SNAPSHOT_PROPAGATION procedure
  DBMS_REPCAT package, 4-16
altering
  priority group members
    priorities, 5-25
    values, 5-25
  priority levels, 9-91
  propagation method, 9-87, 9-99

replicated objects, 9-89
site priority group members
priorities, 5-29
values, 5-30
snapshot definitions, 4-21
snapshots
required privileges, 2-36
alternate keys
detecting conflicts and, 1-26
in replicated tables, 3-20
AND expression
for simple subquery snapshots, 2-24
append sequences
conflict resolution methods, 5-34
append site names
conflict resolution methods, 5-34
assigning
update conflict resolution methods, 5-15
asynchronous
DDL, 9-139
replication, 1-21
RPCs, 8-3
auditing, 6-16
conflict resolution, 6-16
automatic conflict resolution
versus manual, 5-7
automatic refreshes
refresh group, 1-10
refresh interval, 1-10
average
conflict resolution method
use of, 5-38
average prebuilt conflict resolution method, 5-16

## B

backups
for replication, 6-15
using multiple conflict resolution methods
for, 5-11
basic replication, 1-2, 1-4 to 1-12, 2-1 to 2-44
configuring environment, 2-2 to 2-27
example, 2-2
designing environment, 2-2
environment
read-only snapshots, 2-34 to 2-37
managing environment, 2-27 to 2-44
monitoring environment, 2-44
privileges in, 2-5
SNP background processes for, 2-6
uses of, 1-4
Batch Size setting
Create New Scheduled Link property
sheet, 3-10
BEGIN_INSTANTIATION procedure
DBMS_OFFLINE_OG package, 7-16
BEGIN_LOAD procedure
DBMS_OFFLINE_OG package, 7-17
BEGIN_TABLE_REORGANIZATION procedure
DBMS snapshot package, 2-31
BLOBs support, A-3
branch automation, A-2
bulk updates, 6-24

## C

catalog, replication, 1-19
CHANGE procedure
DBMS_REFRESH package, 2-38
checking imported data, 6-15
circular dependencies in tables
restrictions on adding sites with to master
groups, 3-25
CLOBs support, A-3
Cluster page
Create Snapshot property sheet, 2-15
clustering snapshots, 2-15
column groups, 1-26
adding members to
syntax, 9-75
and update conflict resolution, 5-7
creating
syntax, 9-118, 9-149
dropping, 5-14
syntax, 9-123
removing members from
syntax, 9-124
shadow, 1-27, 5-7
columns
adding and removing in column groups, 5-14

commands, SQL
ALTER SNAPSHOT LOG, 2-28
CREATE SNAPSHOT, 2-7, 2-10, 2-19 to 2-21
CREATE SNAPSHOT LOG, 2-7, 2-17
CREATE TABLE, 2-16
DROP SNAPSHOT, 2-36
DROP SNAPSHOT LOG, 2-33
TRUNCATE, 2-30
TRUNCATE TABLE, 2-31
comments
on Oracle documentation, vi
updating, 6-28
comments field
in views
updating, 6-28
comparing
tables, 9-55
complete refreshes, 1-9
complex snapshots, 1-11, 2-41
value for PCTFREE, 2-43
value for PCTUSED, 2-43
configuration
basic replication environment, 2-2 to 2-27
example, 2-2
Configuration folder, 3-29, 6-5
conflicts
additive resolution method, 5-16
avoidance
dynamic ownership, 7-27
avoiding, 5-4
column groups and, 1-26
data models and, 1-24
delete, 1-24, 5-3
avoiding, 5-5
detecting, 1-26, 5-5, 7-8
master sites, avoiding at, 5-5
detection
identifying rows during, 5-6
how Oracle detects, 5-6
maximum value conflict resolution method, 5-17
minimum value conflict resolution method, 5-17
notification
package, sample, 5-52
notification log
creating, 5-51
notification log table
sample, 5-51
notification methods
user-defined, 5-50
notification package
creating, 5-51
procedural replication, 1-28
replication, 1-24
resolution, 1-26
adding method of, 9-83
auditing, 6-16
automatic versus manual, 5-7
configuration
overview, 5-12
declarative methods, update conflicts, 5-16
delete, 5-8
strategy, 7-25
delete, configuring, 5-36
design and preparation guidelines, 5-12
detecting conflicts, 5-5
gathering statistics, 6-16
highest priority, 5-29
in synchronous propagation, 7-7
information
viewing, 5-54
methods, 1-27
procedural replication and, 7-3
RepResolution_Statistics table, deleting, 6-17
statistics, 9-100, 9-156
canceling, 6-17
uniqueness, 5-8
configuring, 5-33
update
and column groups, 5-7
user-defined routines
parameters, 5-46
user-defined routines, example, 5-49
user-defined routines, restrictions, 5-48
viewing information, 5-54
resolution and notification methods
custom, 5-10

resolution methods, 5-8
average prebuilt, 5-16
delete, assigning, 5-36
delete, removing, 5-36
discard, 5-19
managing, 5-15
multiple
using, 5-10
overwrite, 5-19
prebuilt, restrictions for, 5-9
uniqueness, assigning, 5-33
uniqueness, removing, 5-33
update
prebuilt, 5-8
user-defined, 5-46
using, for notification, 5-11
resolution methods for
removing, 5-15
resolution methods for column groups
ordering, 5-15
row-level replication, 1-26
uniqueness, 1-24, 5-3
avoiding, 5-5
update, 1-24, 5-3
avoiding, 5-5
update resolution
using site priority for, 5-28
connection qualifiers
master groups, 3-16
constraint violations, 6-24
continuous pushes
scheduling, 3-11, 3-13
Create, 3-5
Create Master Sites dialog
setup wizard, 3-24
Create New Master Group property sheet
Destinations page, 3-15
General page, 3-15
Objects page, 3-15
Replication Manager, 3-15
Create New Scheduled Link property sheet, 3-9
General page, 3-9
Options page, 3-9
Create Refresh Group property sheet
General page, 4-22
Next Date field, 2-26
Replication Manager, 4-3
Scheduling page, 4-22
Snapshots page, 4-22
Create Scheduled Link property sheet
setup wizard, 3-11, 3-13
CREATE SNAPSHOT command, 2-7, 2-10, 2-19 to 2-21
REFRESH clause, 2-44
CREATE SNAPSHOT LOG command, 2-17
Create Snapshot Log property sheet
Filter Columns page, 4-11
General page, 4-11
Options page, 2-8
Replication Manager, 4-3, 4-11
Storage page, 2-8
Tablespace and Extents page, 4-11
Create Snapshot property sheet
Cluster page, 2-15
Options page, 2-14
Storage page, 2-14
CREATE TABLE command, 2-16
CREATE_MASTER_REPGROUP procedure
DBMS_REPCAT package, 3-16
creating
basic replication environment, 2-2 to 2-27
example, 2-2
column groups
syntax, 9-118, 9-149
database links for snapshot, 2-2
deferred transactions, 8-3
master sites, 3-5
priority groups, 5-23, 9-119
refresh groups, 9-67
replicated object groups
syntax, 9-109
replicated objects
generating support for, 9-141, 9-142, 9-144
snapshot sites, 9-115
syntax, 9-110
site priority groups, 5-28
syntax, 9-121
snapshot logs, 2-3, 2-6
snapshot refresh groups, 2-3

snapshot sites
syntax, 9-113
snapshots, 2-3, 4-17
simple, 2-10
Customize Master Site property sheet
setup wizard, 3-7
Customize Snapshot Site property sheet
setup wizard, 4-8

## D

data
convergence, 5-6
guaranteeing, 5-37
inconsistencies
removing after deleting master sites, 3-26
integrity
ensuring with multiple column groups, 5-7
parallel propagation, 3-34
serial propagation, 3-34
data cluster
creating snapshots as part of, 2-15
data definition language (DDL)
supplying asynchronous, 9-139
data dictionary
DBA_REGISTERED_SNAPSHOTS view, 2-35
data dictionary views, 10-2
DEFCALL, 10-20
DEFCALLDEST, 10-20
DEFDEFAULTDEST, 10-21
DEFERRCOUNT, 10-21
deferred transactions, 10-19
DEFERROR, 10-22
DEFLOB, 10-22
DEFPROPAGATOR, 10-23
DEFSCHEDULE, 10-23
DEFTRAN, 10-24
DEFTRANDEST, 10-24
REFRESH, 10-30
REFRESH_CHILDREN, 10-31
REGISTERED_SNAPSHOTS, 10-27
REPCATLOG, 10-5
REPCOLUMN, 10-6
REPCOLUMN_GROUP, 10-6
REPCONFLICT, 10-7
REPDDL, 10-7
REPGENERATED, 10-8
REPGENOBJECTS, 10-18
REPGROUP, 10-3
REPGROUPED_COLUMN, 10-9
REPKEY_COLUMNS, 10-9
REPOBJECT, 10-10
REPPARAMETER_COLUMN, 10-11
REPPRIORITY, 10-12
REPPRIORITY_GROUP, 10-13
REPPROP, 10-13
REPRESOL_STATS_CONTROL, 10-15
REPRESOLUTION, 10-14
REPRESOLUTION_METHOD, 10-15
REPRESOLUTION_STATISTICS, 10-16
REPSITES, 10-17
SNAPSHOT_REFRESH_TIMES, 10-29
SNAPSHOTS, 10-26
snapshots, 10-25
SNAPSHOTS_LOGS, 10-28
data manipulation language (DML)
minimizing propagation, 5-40
data ownership models, 1-24
dynamic ownership, 1-25
primary ownership, 1-25
shared ownership, 1-25
static ownership, 1-25
data propagation
and dependency maintenance, 3-33
minimizing, 3-28
minimizing for update conflict resolution, 5-40
mode of
altering in a master site, 7-10
reduced, A-2
synchronous, 7-6
data replication
real-time, 7-6
data requirements
evaluating and understanding, 5-2
data subsetting, A-2
data warehouse
basic replication for, 1-5
database
links
using different paths, 3-16

preparing for multimaster replication, 3-4
Database Link setting
Create New Scheduled Link property sheet, 3-9
database link specifications
incomplete, 6-25
database links, 6-22
creating for snapshot site, 2-2
snapshot sites, for, 2-5
database objects
for read-only snapshot, 1-7
Database Objects folder
master definition sites, 3-19
datatypes
allowed in replicated tables, 3-20
allowed in snapshots, 2-12, 4-18
support for, A-3
DBA_REFRESH view, 2-44
DBA_REFRESH_CHILDREN view, 2-44
DBA_REGISTERED_SNAPSHOTS view, 2-35, 2-44
DBA_RGROUP view, 2-44
DBA_SNAPSHOT_LOGS view, 2-35, 2-44
DBA_SNAPSHOTS view, 2-44
DBMS_DEFER package
ADD_DEFAULT_DEST procedure, 8-6
CALL procedure, 9-5
CHAR_ARG procedure, 8-5, 9-8
COMMIT_WORK procedure, 9-7
datatype_ARG procedure, 9-8
DATE_ARG procedure, 8-5, 9-8
NUMBER_ARG procedure, 8-5, 9-8
RAW_ARG procedure, 8-5, 9-8
ROWID_ARG procedure, 8-5, 9-8
TRANSACTION procedure, 8-4, 9-9
VARCHAR2_ARG procedure, 8-5, 9-8
DBMS_DEFER_QUERY package
GET_ARG_TYPE procedure, 9-12
GET_CALL_ARGS procedure, 9-14
GET_CHAR_ARG procedure, 9-16
GET_DATE_ARG procedure, 9-16
GET_NUMBER_ARG procedure, 9-16
GET_RAW_ARG procedure, 9-16
GET_ROWID_ARG procedure, 9-16
GET_VARCHAR2_ARG procedure, 9-16
DBMS_DEFER_SYS package
ADD_DEFAULT_DEST procedure, 9-19
DELETE_DEF_DESTINATION procedure, 9-21
DELETE_DEFAULT_DEST procedure, 8-6, 9-20
DELETE_ERROR procedure, 9-22
DELETE_TRAN procedure, 9-23
DISABLED function, 9-24
DISABLED procedure, 3-12
EXCLUDE_PUSH function, 9-25
EXECUTE_ERROR procedure, 9-26
PURGE procedure, 9-28
PUSH procedure, 9-31
REGISTER PROPAGATOR procedure, 9-34
SCHEDULE_EXECUTION procedure, 3-10, 3-11, 3-13, 3-14, 9-37
SCHEDULE_PURGE procedure, 9-35
SCHEDULE_PUSH procedure, 3-10, 3-11, 3-13, 3-14, 9-37
SET_DISABLED procedure, 3-11, 3-13, 3-14, 9-40
UNSCHEDULE_PURGE procedure, 9-42
UNSCHEDULE_PUSH procedure, 3-12, 9-43
DBMS_OFFLINE_OG package
BEGIN_INSTANTIATION procedure, 7-16, 9-45
BEGIN_LOAD procedure, 7-17, 9-46
END_INSTANTIATION procedure, 7-17, 9-47
END_LOAD procedure, 7-17, 9-48
RESUME_SUBSET_OF_MASTERS procedure, 7-16, 9-49
DBMS_OFFLINE_SNAPSHOT package
BEGIN_LOAD procedure, 9-51
END_LOAD procedure, 9-53
DBMS_RECTIFIER_DIFF package, 3-26, 6-17
DIFFERENCES procedure, 9-55
RECTIFY procedure, 9-59
DBMS_REFRESH package
ADD procedure, 2-38, 9-63
CHANGE procedure, 2-38, 9-64
DESTROY procedure, 2-38, 9-66
MAKE procedure, 4-22, 9-67
REFRESH procedure, 2-39, 9-70
SUBTRACT procedure, 2-38, 9-71
DBMS_REPCAT package, 6-6
ADD_DELETE_RESOLUTION procedure, 9-83
ADD_GROUPED_COLUMN procedure, 9-75
ADD_MASTER_DATABASE procedure, 3-25,

9-77
ADD_PRIORITY_CHAR procedure, 5-24, 9-79
ADD_PRIORITY_DATE procedure, 5-24, 9-79
ADD_PRIORITY_NUMBER procedure, 5-24, 9-79
ADD_PRIORITY_RAW procedure, 5-24, 9-79
ADD_PRIORITY_VARCHAR2 procedure, 5-24, 9-79
ADD_SITE_PRIORITY_SITE procedure, 5-29, 9-81
ADD_UNIQUE_RESOLUTION procedure, 9-83
ADD_UPDATE_RESOLUTION procedure, 9-83
ALTER_MASTER_PROPAGATION procedure, 9-87
ALTER_MASTER_REPOBJECT procedure, 3-23, 9-89
ALTER_PRIORITY procedure, 5-25, 9-91
ALTER_PRIORITY_CHAR procedure, 5-25, 9-93
ALTER_PRIORITY_DATE procedure, 5-25, 9-93
ALTER_PRIORITY_NUMBER procedure, 5-25, 9-93
ALTER_PRIORITY_RAW procedure, 5-25, 9-93
ALTER_PRIORITY_VARCHAR2 procedure, 5-25, 9-93
ALTER_SITE_PRIORITY procedure, 5-29, 9-95
ALTER_SITE_PRIORITY_SITE procedure, 5-30, 9-97
ALTER_SNAPSHOT_PROPAGATION procedure, 9-99
CANCEL_STATISTICS procedure, 6-17, 9-100
COMMENT_ON_COLUMN_GROUP procedure, 6-28, 9-101
COMMENT_ON_DELETE resolution package, 6-28
COMMENT_ON_DELETE_RESOLUTION procedure, 9-107
COMMENT_ON_PRIORITY_GROUP procedure, 6-28, 9-102
COMMENT_ON_REPCAT procedure, 6-28
COMMENT_ON_REPGROUP procedure, 9-104
COMMENT_ON_REPOBJECT procedure, 6-28, 9-106
COMMENT_ON_REPSCHEMA procedure, 6-28
COMMENT_ON_REPSITES procedure, 9-105
COMMENT_ON_SITE_PRIORITY procedure, 9-102
COMMENT_ON_UNIQUE resolution package, 6-28
COMMENT_ON_UNIQUE_RESOLUTION procedure, 9-107
COMMENT_ON_UPDATE resolution package, 6-28
COMMENT_ON_UPDATE_RESOLUTION procedure, 9-107
CREATE_MASTER_REPGROUP procedure, 3-16, 9-109
CREATE_MASTER_REPOBJECT procedure, 9-110
CREATE_SNAPSHOT_REPGROUP procedure, 9-113
CREATE_SNAPSHOT_REPOBJECT procedure, 9-115
DEFINE_COLUMN_GROUP procedure, 9-118
DEFINE_PRIORITY_GROUP procedure, 5-23, 9-119
DEFINE_SITE_PRIORITY procedure, 5-28, 9-121
DO_DEFERRED_REPCAT_ADMIN procedure, 6-5, 6-6 to 6-7, 9-122
DROP_COLUMN_GROUP procedure, 9-123
DROP_GROUPED_COLUMN procedure, 9-124
DROP_MASTER_REPGROUP procedure, 3-17, 9-126
DROP_MASTER_REPOBJECT procedure, 3-24, 9-128
DROP_PRIORITY procedure, 5-27, 9-129
DROP_PRIORITY_CHAR procedure, 5-26, 9-131
DROP_PRIORITY_DATE procedure, 5-26, 9-131
DROP_PRIORITY_GROUP procedure, 5-27, 9-130
DROP_PRIORITY_NUMBER procedure, 5-26, 9-131
DROP_PRIORITY_RAW procedure, 5-26, 9-131
DROP_PRIORITY_VARCHAR2 procedure, 5-26, 9-131
DROP_SITE_PRIORITY procedure, 5-30, 9-133

DROP_SITE_PRIORITY_SITE procedure, 5-30, 9-134
DROP_SNAPSHOT_REPGROUP procedure, 9-135
DROP_SNAPSHOT_REPOBJECT procedure, 4-21, 9-136
EXECUTE_DDL procedure, 6-2, 9-139
GENERATE_REPLICATION_PACKAGE procedure, 9-141
GENERATE_REPLICATION_SUPPORT procedure, 3-28, 9-142
GENERATE_REPLICATION_TRIGGER procedure, 9-144
GENERATE_SNAPSHOT_SUPPORT, 9-147
MAKE_COLUMN_GROUP procedure, 9-149
PURGE_MASTER_LOG procedure, 9-151
PURGE_STATISTICS procedure, 6-17, 9-152
REFRESH_SNAPSHOT_REPGROUP procedure, 9-153
REGISTER_STATISTICS procedure, 6-16, 9-156
RELOCATE_MASTERDEF procedure, 6-3, 9-157
REMOVE_MASTER_DATABASES procedure, 3-25, 9-159
REPCAT_IMPORT_CHECK procedure, 6-15, 9-161
RESUME_MASTER_ACTIVITY procedure, 3-19, 9-162
SET_COLUMNS procedure, 3-20, 9-165
SUSPEND_MASTER_ACTIVITY procedure, 3-18, 9-167
SWITCH_SNAPSHOT_MASTER procedure, 4-16, 6-3, 9-168
VALIDATE procedure, 9-170
WAIT_MASTER_LOG procedure, 9-173
DBMS_REPCAT_ADMIN package
GRANT_ADMIN_ANY_SCHEMA procedure, 9-175
GRANT_ADMIN_SCHEMA procedure, 9-176
REVOKE_ADMIN_ANY_SCHEMA procedure, 9-177
REVOKE_ADMIN_SCHEMA procedure, 9-178
DBMS_REPCAT_AUTH package
GRANT_SURROGATE_REPCAT procedure, 9-180
REVOKE_SURROGATE_REPCAT procedure, 9-181
DBMS_REPUTIL package
FROM_REMOTE variable, 9-197
GLOBAL_NAME variable, 9-197
REPLICATION_IS_ON variable, 9-197
REPLICATION_OFF procedure, 7-33, 9-183
REPLICATION_ON procedure, 7-34, 9-184
DBMS_SNAPSHOT package
BEGIN_TABLE_REORGANIZATION procedure, 2-31, 9-186
END_TABLE_REORGANIZATION procedure, 2-31, 9-187
I_AM_A_REFRESH function, 9-188
PURGE_LOG procedure, 2-29, 9-189
REFRESH procedure, 9-190
REGISTER_SNAPSHOT procedure, 2-36, 9-193
SET_I_AM_A_REFRESH procedure, 9-195
UNREGISTER_SNAPSHOT, 9-196
UNREGISTER_SNAPSHOT procedure, 2-36
DDL
changes not propagated to master site, 6-23
executing within a master group, 6-2
propagating statements to master sites, 6-2
deadlocks
resolving
in synchronous propagation, 7-7
decision support application
basic replication for, 1-5
DEFCALL view, 10-20
DefDefaultDest table
adding destinations to, 8-6, 9-19
removing destinations from, 8-6, 9-20, 9-21
DEFDEFAULTDEST view, 10-21
DEFERRCOUNT view, 10-21
deferred constraints and updatable snapshots, A-4
deferred RPCs, 8-3
deferred transactions, 1-22
creating, 8-3
DefCall table, 8-5
DefCallDest table, 8-5
DEFCALLDEST view, 10-20
DefDefaultDest table
adding destination to, 8-6, 9-19
removing destinations from, 8-6, 9-20, 9-21

DEFDEFAULTDEST view, 10-21
deferred remote procedure calls (RPCs)
argument types, 9-12
argument values, 9-16
arguments to, 8-5, 9-8
building, 8-5, 9-5
executing immediately, 9-31
DEFERROR view, 10-22
DefTran table, 8-4
DEFTRAN view, 10-24
DEFTRANDEST view, 10-24
deleting, 6-10
diagnosing problems with, 6-25
displaying, 6-9
executing, 6-9
managing, 6-8
purging, 3-12
queue
clearing after deleting master sites, 3-26
purging, 1-23, 3-14
push, 1-22
re-executing, 9-26
removing from queue, 9-23
scheduling execution, 9-37
starting, 8-4, 9-9
views for, 10-19 to 10-24
Deferred Transactions by Dest folder, 6-9
DefError table
deleting transactions from, 9-22
DEFERROR view, 10-22
defining query
snapshots, 1-7, 2-12
definition sites
relocating for a master group, 6-3
DEFLOB, 10-22
DEFSCHEDULE, 10-23
DEFTRAN, 10-24
DEFTRANDEST view, 10-24
delay seconds
setup wizard, 3-13
Delay Seconds setting
Create New Scheduled Link property sheet, 3-10
delete conflicts, 1-24, 5-3
avoiding, 5-5, 7-25
resolution, 5-8
configuring, 5-36
methods, assigning, 5-36
resolution methods
removing, 5-36
resolving, 5-48
deleting
master groups, 3-17
scheduled links, 3-12
snapshot logs, 4-12
snapshots, 4-21
dependency
ordering
replicated transactions, 3-33
tracking
parallel propagation, 3-34
Destination Map folder, 3-29
destination maps
displaying for master groups, 3-29
destination master sites
adding to master groups, 3-15, 3-25
Destinations page
Create New Master Group property sheet, 3-15
Edit Master Group property sheet, 3-25, 6-3
DESTROY procedure
DBMS_REFRESH package, 2-38
detecting conflicts, 5-5, 5-6
diagnosing problems with, 6-22
DIFFERENCES
procedure, 6-17
differences
between tables, 9-55
rectifying, 9-59
DISABLED procedure
DBMS_DEFER_SYS package, 3-12
disabling
propagation, 9-40
disabling replication, 7-32
discard
conflict resolution method, 5-19
use of, 5-38
discard uniqueness
conflict resolution method, 5-35
disconnected environments
as in advanced replication, 1-13

distributed transactions
problems with, 6-25
distributing application loads
as in advanced replication, 1-13
DML
changes not propagated to other sites, 6-23
DO_DEFERRED_REPCAT_ADMIN package, 6-6
DO_DEFERRED_REPCAT_ADMIN procedure
DBMS_REPCAT package, 6-5, 6-6 to 6-7
DROP SNAPSHOT command, 2-36
DROP SNAPSHOT LOG command, 2-33
DROP_MASTER_REPGROUP procedure
DBMS_REPCAT package, 3-17
DROP_MASTER_REPOBJECT procedure
DBMS_REPCAT package, 3-24
DROP_SNAPSHOT_REPOBJECT procedure
DBMS_REPCAT package, 4-21
dropping
column groups, 5-14
syntax, 9-123
master sites, 9-159
priority group members
by priority, 5-27
by value, 5-26
priority groups, 5-27, 9-130
replicated objects
from master sites
syntax, 9-128
from snapshot sites
syntax, 9-136
groups of, 9-126
site priority groups, 5-30, 9-133
members of, 5-30
sites
by priority level, 5-30
snapshot logs, 2-33
snapshot sites, 9-135
snapshots, 2-36
dynamic ownership, 1-25
conflict avoidance and, 7-27
workflow partitioning, 7-27
dynamic sites
ownership, 5-4

## E

earliest and latest timestamp
conflict resolution methods, 5-18
Edit Database Destination property sheet
Replication Manager, 3-30
Edit DB Connection property sheet
Replication Manager, 3-14
setup wizard, 3-13
Edit Master Group property sheet
Destinations page, 3-25, 6-3
Operations page, 3-18, 3-28
Replication Manager, 3-18, 3-19
setup wizard, 3-25
Edit Replication Object property sheet
General page, 3-23, 3-27, 4-20
Min(imize) Communications setting, 3-28
Edit Scheduled Link property sheet
Replication Manager, 3-11
setup wizard, 3-11
Edit Snapshot Log property sheet
Filter Columns page, 4-11
General page, 4-11
Replication Manager, 4-11
Tablespace and Extents page, 4-11
Edit Snapshot property sheet
Tablespace and Extents page, 4-20
Enabled setting
Create New Scheduled Link property sheet, 3-9
enabling replication, 7-32
END_INSTANTIATION procedure
DBMS_OFFLINE_OG package, 7-17
END_LOAD procedure
DBMS_OFFLINE_OG package, 7-17
END_TABLE_REORGANIZATION procedure
DBMS_SNAPSHOT package, 2-31
enhancements
new features, A-2
error
transactions
managing, 6-10
errors
avoid with VALIDATE, A-4
transactions
displaying, 6-11

resolving, 6-11
examples
minimizing
communication, 5-41
Execute DDL dialog
Replication Manager, 6-2
EXECUTE_DDL procedure
DBMS_REPCAT package, 6-2
EXIST clause
for simple subquery snapshots, 2-24
Export utility
offline instantiation of master site, 7-16

## F

fail-over sites
implementing, 7-14
failover sites
as in advanced replication, 1-13
failure, media
recovering read-only snapshots from, 2-37
fast refreshes, 1-9
and table reorg, A-4
snapshot requirements for, 2-17
features requiring migration, B-11
Feedback
on ORACLE documentation, vi
filter columns
for simple subquery snapshots, 2-9, 2-24
Filter Columns page
Create Snapshot Log property sheet, 4-11
Edit Snapshot Log property sheet, 4-11
fine grained quiesce, A-3
Finish page
Replication Manager, 3-7
setup wizard, 3-7

## G

gen_rep_pack, 5-22
General page
Create New Master Group property sheet, 3-15
Create Snapshot Log property sheet, 4-11
Edit Replication Object property sheet, 3-27, 4-20
Edit Snapshot Log property sheet, 4-11
Snapshot Log property sheet, 2-35
GENERATE_REPLICATION_SUPPORT procedure
DBMS_REPCAT package, 3-28
generated objects
displaying for master groups, 3-30
Generated Objects folder, 3-30
generated replication objects
wrapper package, 1-20
generating
replication support
procedural replication, 7-3
replication support and migration, B-3
snapshot support, 9-147
generating replication support
migration and, B-8

## H

hybrid configurations
advanced replication, 1-16

## I

identity columns
detecting conflicts and, 1-26
Import utility
offline instantiation of master site, 7-17
import_check, 6-15
importing
object groups
offline instantiation and, 9-46, 9-48
snapshots
offline instantiation and, 9-51, 9-53
status check, 9-161
in basic replication, 1-5
incremental migration, B-4
indexes
partitioned tables and, A-4
snapshots, 2-40
information
consolidation
advanced replication and, 1-15
distribution
in basic replication, 1-4

off-loading
in basic replication, 1-5
transport, 1-5
information distribution
basic replication for, 1-4
initialization parameters
JOB_QUEUE_INTERVAL, 2-6, 4-10, 6-6
JOB_QUEUE_PROCESSES, 2-6, 4-10, 6-6
PARALLEL_MAX_SERVERS, 3-32
PARALLEL_MIN_SERVERS, 3-32
PARALLEL_SERVER_IDLE_TIME, 3-32
INIT.ORA, 3-17
instantiation
offline, 7-14
in advanced replication, 7-17
of an advanced replication master site, 7-16
instantiation offline, A-4
internal snapshot objects, 1-7
internal triggers, A-2
intersection tables, 2-20 to 2-23
Interval Expression setting
Create New Scheduled Link property sheet, 3-9
Setup wizard, 3-12


## J


job queue process, 3-8, 4-10
job queues, 1-22
JOB_QUEUE_INTERVAL initialization parameters, 2-6, 4-10, 6-6
JOB_QUEUE_PROCESSES initialization parameters, 2-6, 4-10, 6-6
jobs
breaking, 6-14
checking for scheduled links, 6-25
diagnosing problems with, 6-14
enabling, 6-14
local
displaying, 6-12
editing properties of, 6-13
manually executing, 6-14
queues for
removing jobs from, 9-42 to 9-43
joins
for simple subquery snapshots, 2-24


## L


large object types
allowed in replicated tables, 3-20
latest timestamp
conflict resolution method, 5-18
link specifications
incomplete, 6-25
LOBs
omitting old values for, 5-40
support for, A-3
Local Errors folder, 6-11
local jobs
displaying, 6-12
editing properties of, 6-13


## M


MAKE procedure
DBMS_REFRESH package, 2-26, 4-22
NEXT_DATE parameter, 2-26
management
basic replication environment, 2-27 to 2-44
management and use new features, A-3
management of master and snapshot groups
advanced, 6-2
managing update conflict resolution, 5-15
manual conflict resolution
versus automatic, 5-7
manual refreshes, 1-11
many-to-many references
simple subquery snapshots, 2-23
many-to-one references
simple subquery snapshots, 2-23
mass deployment, A-2
master and snapshot groups
advanced management of, 6-2
master definition sites, 1-19
relocating, 6-3, 9-157
master groups, 1-18
adding destination master sites, 3-24, 3-25
adding replication objects, 3-15, 3-19
adding sites with tables having circular dependencies, 3-25
adding sites with tables having self-referential

constraints, 3-25
administration requests
deleting, 6-6
altering object definition, 3-23
applying administration requests for, 6-5
choosing, for snapshot groups, 4-4
connection qualifiers, 3-16
creating, 3-15
definition sites for
relocating, 6-3
deleting, 3-17
deleting master sites from, 3-25
destination master sites, 3-15
displaying destination maps for, 3-29
displaying generated objects, 3-30
executing DDL within, 6-2
generating replication support for changes, 3-26
listing, 3-29
listing objects in, 3-29
managing, 3-15 to 3-30
objects
generating support for, 3-27
propagating DDL to sites in, 6-2
relocating master definition sites, 6-3
removing replication objects, 3-24
resuming replication activity, 3-19
suspending replication activity, 3-17
validating, 6-2
viewing information about, 3-29
Master Groups folder, 3-29, 6-5
master sites, 1-18
adding to master groups, 3-24
administrator accounts, 3-5
choosing, for snapshot groups, 4-4
creating, 9-77
creating, for multimaster replication, 3-5
customizing settings, 3-7
default propagation characteristics, 3-6
deleting from master groups, 3-25
determining differences, 6-17
diagnosing problems with, 6-22
dropping, 9-159
fine grained quiesce, A-3
incremental migration of, B-6
manually loading table data from, 3-22
master groups
managing, 3-15 to 3-30
minimizing data propagation, 3-28
offline instantiation, 7-16 to 7-17
propagating changes between, 9-37
propagator accounts, 3-5
receiver accounts, 3-5
replicating data to, 3-22
replicating object definitions to, 3-21
manual, 3-21
scheduled links for, 3-9
guidelines, 3-10
scheduled purges for
guidelines, 3-13
schema creation for, 3-6
snapshot registration, A-3
specifying for snapshot sites, 4-6
master sites and migration, B-2
master table
columns
number restriction for simple subquery snapshots, 2-24
datatypes allowed in, 2-12
dropping, 2-36
PRIMARY KEY constraint for snapshot, 2-12
reorganizing, 2-30
requirements for primary key snapshots, 2-8
master tables
reorganizing
methods, 2-31 to 2-33
snapshot logs, 1-9
snapshot logs for, 2-6
truncating, 2-31
maximum value conflict resolution method, 5-17
media failure
recovery
read-only snapshots, 2-37
members
altering priority for, 5-25
migration
features requiring, B-11
incremental, B-4
master sites, B-2
obsolete procedures, B-12
overview, B-2

snapshot sites, B-5
upgrading to primary key snapshots, B-10
using import and export, B-9
Min(imize) Communications setting
Edit Replication Object property sheet, 3-28
min_communication
examples of, 5-41
Minimize Communication setting
Replication Manager, 4-19
minimizing
updates and min_communication, 5-41
minimum value conflict resolution method, 5-17
mode of propagation, 7-8
modifying
tables
without replicating changes, 7-32
monitoring
environment in basic replication, 2-44
multimaster replication, 1-14
advanced options, 3-31 to 3-35
building environment, 3-1 to 3-3
example, 3-2
creating
master groups, 3-15
customizing master site settings, 3-7
database preparation, 3-4
local job queues, 6-12
master site creation, 3-5
monitoring system, 3-29 to 3-30
monitoring systems, 6-4 to 6-15
reviewing configuration settings, 3-7
scheduled links, 3-9
schema creation for, 3-6
security for, 7-18
setup wizard, 3-4
transaction propagation protection, 3-33 to 3-34
multiple column groups
ensuring data integrity with, 5-7
multiple conflicts
resolution methods
for backups, 5-11
using, 5-10
using for notification, 5-11

## N

naming
snapshots, 2-11
NCLOB support, A-3
new features
data subsetting and subqueries, A-2
enhanced security, A-4
for management and use, A-3
LOB support, A-3
performance, A-2
subqueries for snapshots, A-2
subquery snapshots, A-2
New Master Site dialog
setup wizard, 3-5
New Replicated Object Schema dialog
setup wizard, 3-6
New Snapshot Site dialog
setup wizard, 4-6
Next Date field
Create Refresh Group property sheet, 2-26
Next Date setting
Create New Scheduled Link property sheet, 3-9
Setup wizard, 3-12
NEXT_DATE parameter
DBMS_REFRESH package
MAKE procedure, 2-26
notification
methods
custom, for conflict resolution, 5-10
using multiple conflict resolution methods
for, 5-11
notification log
conflict
creating, 5-51
notification log table
conflict
sample, 5-51
notification methods
user-defined, 5-50
notification package
conflict
creating, 5-51

## O


objects
  altering in a master group, 3-23
  definitions
    replication to master sites, 3-21
  master groups
    generating support for, 3-27
  removing from master groups, 3-24
  replicated
    re-creating, 6-24
Objects page
  Create New Master Group property sheet, 3-15
obsolete procedures, B-12
offline instantiation, 3-22
  master sites, 7-16 to 7-17
  replicated object groups, 9-45, 9-46, 9-47, 9-48, 9-49
  snapshots, 9-51, 9-53
  support for, A-4
OPEN_LINKS initialization parameters
  initialization parameters
    OPEN_LINK, 3-17
Operations page
  Edit Master Group property sheet, 3-18, 3-28, 6-2
Options page, 2-28
  Create Snapshot property sheet, 2-14
Oracle Replication Manager, 1-18
Oracle7
  and min_communication, 4-19
Oracle8
  features and migration, B-11
ordering
  conflict resolution methods for column groups, 5-15
ordering conflicts
  avoiding, 5-38
overwrite
  conflict resolution method
    use of, 5-38
  conflict resolution methods, 5-19


## P


package variables
  from_remote, 9-197
  global_name, 9-197
  i_am_a_refresh, 9-188
  i_am_a_snapshot, 9-197
  replication_is_on, 9-197
parallel propagation, 1-22, A-2
  advanced replication environment, 3-31 to 3-32
  dependency
    tracking, 3-34
  planning for, 3-31
Parallel Propagation setting
  Create New Scheduled Link property sheet, 3-10
parallel server processes
  configuring for advanced replication environments, 3-32
PARALLEL_MAX_SERVERS initialization parameters, 3-32
PARALLEL_MIN_SERVERS initialization parameters, 3-32
PARALLEL_SERVER_IDLE_TIME initialization parameters, 3-32
partitioned tables
  indexes and, A-4
partitions
  replication and, 3-21
PCTFREE
  value for complex snapshots, 2-43
  value for snapshot log, 2-9
PCTUSED
  value for complex snapshots, 2-43
  value for snapshot log, 2-9
performance enhancements, A-2
performance tuning
  simple subquery snapshots, 2-41
  snapshots, 2-40
periodic purges
  scheduling, 3-14
periodic pushes
  scheduling, 3-11
pgroup, 5-22
PL/SQL triggers

using with snapshot base tables, 4-21
prebuilt conflicts
resolution methods
restrictions for, 5-9
prebuilt uniqueness conflicts
resolution methods, 5-9
prebuilt update conflict resolution methods, 5-16
prebuilt update conflicts
resolution methods, 5-8
preface
Send Us Your Comments, xxiii
preparing for, 2-4
PRESERVE SNAPSHOT LOG option
TRUNCATE TABLE command, 2-31
PRIMARY KEY constraint
simple subquery snapshots and, 2-23
snapshot requirement for, 2-12
primary key snapshots, 1-12, A-3
requirements for creating log, 2-8
primary keys
missing from replicated tables, 3-16
upgrading snapshots, B-10
primary ownership, 1-25
primary sites
ownership, 5-4
priority group members
dropping, 5-27
dropping by value, 5-26
priority groups, 5-22
adding members to, 5-24, 9-79
altering members
priorities, 5-25, 9-91
values, 5-25, 9-93
and site priority, 5-20
creating, 5-23, 9-119
dropping, 5-27, 9-130
dropping members
by priority, 5-27
by value, 5-26
removing members from, 9-129, 9-131
site priority groups
adding members to, 9-81
using for conflict resolution, 5-22
priority level of sites
altering, 5-29
priority_groups, 5-20
privileges
altering snapshots, 2-36
basic replication environment, 2-3, 2-5
deleting rows from snapshot logs, 2-30
dropping snapshots, 2-37
snapshot creation, 2-11
snapshot log creation, 2-8
procedural replication, 1-28
conflicts and, 7-3
deleting and, 7-25
detecting conflicts, 1-28
generating support for, 7-3
restrictions, 7-2
using, 7-2
wrapper, 1-28
Processes setting
Create New Scheduled Link property sheet, 3-10
propagating changes
altering propagation method, 9-99
propagation
disabling, 9-40
minimizing data, 5-40
mode of, 7-8
altering in a master site, 7-10
of changes
altering propagation method, 9-87
parallel, 1-22, A-2
security context of propagator, 7-6
serial, 1-22
setting default characteristics, 3-6
status of, 9-24
propagation reduction, A-2
propagator, 1-24
account
snapshot sites, 4-5, 4-7
registering, 9-34
propagator accounts
master sites, 3-5
PURGE SNAPSHOT LOG option
TRUNCATE TABLE command, 2-31
PURGE_LOG procedure
DBMS_SNAPSHOT package, 2-29
purges

manual, 3-14
periodic
scheduling, 3-14, 4-8
purging
deferred transaction queue, 3-12
RepCatLog table, 9-151
snapshot logs, 2-29
statistics, 9-152
pushes
continuous
scheduling, 3-11, 3-13
periodic
scheduling, 3-11

## Q

query defining
for snapshots, 2-12
queue
deferred transactions, 1-22
quiescing
and fine grained quiesce, A-3
master groups, 3-17
replicated schemas, 9-167

## R

read-only replication, 1-2, 1-4 to 1-8
uses of, 1-4
read-only snapshots, 1-6
altering
privileges required, 2-36
base table, 1-7
dropping, 2-36
index, 1-8
managing, 2-34 to 2-37
privileges required to query, 2-34
recovery from media failure, 2-37
refresh types, 1-11
registration, 2-35
manual, 2-36
unregistering, 2-36
using, 2-34
view, 1-8
real-time
data replication, 1-29
replication, 1-28, 7-6
receiver, 1-24
accounts
master sites, 3-5
recovery, 6-15
for replication, 6-15
RECTIFY
procedure, 6-18
rectifying
tables, 6-18, 9-59
re-enabling
advanced replication, 7-34
refresh
failures, 2-39
fast and table reorg, A-4
retries, 2-39
snapshot sites
syntax, 9-153
snapshots, 9-190
REFRESH clause
ALTER SNAPSHOT command, 2-44
CREATE SNAPSHOT command, 2-44
refresh groups, 1-10
adding members, 2-38
adding members to, 9-63
adding snapshots, 4-22
altering settings, 2-38
automatic refreshes, 1-10
creating, 2-3, 2-24, 4-3
for snapshot sites, 4-22
creating new, 9-67
data dictionary views, 10-25
deleting, 2-38, 9-66
deleting members, 2-38
designing, 2-5
listing group members, 10-31
managing, 2-5 to 2-40
manual refresh, 1-11, 2-39
refresh interval, 2-26
changing, 9-64
REFRESH view, 10-30
REFRESH_CHILDREN view, 10-31
refreshing
manually, 9-70

removing members from, 9-71
settings
next date, 2-26
snapshots, 4-22
troubleshooting refresh, 2-39
refresh intervals
listing, 10-30
parameter constraints, 1-10
REFRESH view, 10-30
snapshot refresh groups, 1-10
REFRESH procedure
DBMS_REFRESH package, 2-39
refresh settings
snapshots, 2-12
refresh types
default, 1-11
manual, 1-11
read-only snapshots, 1-11
snapshot, 2-27
REFRESH view, 10-30
refresh, planning for, 2-5
REFRESH_CHILDREN view, 10-31
refreshes
snapshot, 4-9
REGISTER_SNAPSHOT procedure
DBMS_SNAPSHOT package, 2-36
REGISTERED_SNAPSHOTS, 10-27
registering
propagator for local database, 9-34
snapshots, A-3
RELOCATE_MASTERDEF procedure
DBMS_REPCAT package, 6-3
remote data
referencing for snapshots, 2-13
remote procedure calls, 1-22
deferring, 8-5
REMOVE_MASTER_DATABASES procedure
DBMS_REPCAT package, 3-25
removing
columns in column groups, 5-14
conflict resolution methods, 5-15
RepCat table
updating, 6-28
REPCATLOG view, 3-30, 6-6 to 6-7, 10-5
RepCatLog view
purging, 9-151
REPCOLUMN view, 10-6
RepColumn_Group table
updating, 6-28, 9-101
REPCOLUMN_GROUP view, 10-6
REPCONFLICT view, 10-7
REPDDL view, 10-7
REPGENOBJECTS view, 10-18
REPGERATED view, 10-8
REPGROUP view, 3-30, 10-3
RepGroup view
updating, 9-104
REPGROUPED_COLUMN view, 10-9
REPKEY_COLUMNS view, 10-9
replicated environment
determining differences in tables, 6-17
replicated object groups
dropping, 9-126
offline instantiation of, 9-45, 9-46, 9-47, 9-48, 9-49
replicated objects
altering, 9-89
at snapshot sites
problems creating, 6-26
creating
master sites, 9-110
snapshot sites, 9-115
DROP_MASTER_REPOBJECT and, 9-128
dropping
snapshot site, 9-136
generating support for, 9-141, 9-142, 9-144
groups
creating
master sites, 9-109
re-creating, 6-24
replicated procedures
generating support for, 7-3
replicated tables
alternate keys for, 3-20
and DML incompatibility, 6-24
datatypes allowed, 3-20
differences between, 6-17
populating manually, 3-22
replicated transactions
dependency ordering, 3-33

replication
administrator, 1-23
advanced
introduction to conflicts, 5-2
re-enabling, 7-34
advanced techniques, 7-1 to 7-35
advanced, uses for, 1-13
backup and recovery for, 6-15
basic, 1-2, 1-4 to 1-12, 2-1 to 2-44
catalog, 1-19
catalog views, 10-2
conflicts, 1-24
avoiding, 1-26
column groups, 1-26
data models and, 1-24
detecting, 1-26
procedural replication, 1-28
resolution methods, 1-27
resolving, 1-26
row-level replication, 1-26
definition, 1-2
disabling, 7-32, 7-33
enabling, 7-32
environment
advanced
adding new sites to, 7-8
group, 1-18
multimaster, 3-1 to 3-35
advanced options, 3-31 to 3-35
building environment
example, 3-2
monitoring, 3-29 to 3-30, 6-4 to 6-15
objects
generated, 1-20
of object definitions to master sites, 3-21
manual, 3-21
off/on affects current session, 7-33
procedural, 1-28
propagator, 1-24
real-time, 1-28, 7-6
receiver, 1-24
resuming for master groups, 3-19
sites, 1-18
suspending for master groups, 3-17
triggers, 7-34
uses of read-only, 1-4
replication catalog view
incorrect, 6-25
replication groups
master sites
managing, 3-15 to 3-30
replication management API, 1-19, 9-1
Replication Manager, 1-18
Add Objects to Group dialog, 3-19
administration requests, 6-4 to 6-8
Alter Replication Object dialog, 3-23
Create New Master Group property sheet, 3-15
Create New Scheduled Link property sheet, 3-9
Create Refresh Group property sheet, 4-3
Create Snapshot Log property sheet, 4-11
Edit Database Destination property sheet, 3-30
Edit Master Group property sheet, 3-18, 3-19
Edit Replication Object property sheet, 3-23
Edit Snapshot Log property sheet, 4-11
Execute DDL dialog, 6-2
Finish page, 3-7
Scheduled Links folder, 3-12
Scheduling folder, 3-12
Set Alternate Key Columns dialog, 3-16, 3-20
Set Date dialog, 3-9, 3-12
Set Interval dialog, 3-9, 3-12
Set Scheduled Link dialog, 3-9
setup wizard, 3-4
replication objects, 1-18
adding to master groups, 3-15, 3-19
altering definition, 3-23
deleting from master groups, 3-24
replication support
for tables
unable to generate, 6-24
generating for all master group tables, 3-28
generating for master groups, 3-26
generation for objects, 3-27
replication tables
updating Comments, 6-28
replication triggers, A-2
RepObject table
updating, 6-28, 9-106
REPOBJECT view, 3-30, 10-10

REPPARAMETER_COLUMN view, 10-11
REPPRIORITY view, 10-12
RepPriority_Group table
  updating, 6-28, 9-102
REPPRIORITY_GROUP view, 10-13
REPPROP view, 10-13
REPRESOL_STATS_CONTROL view, 10-15
RepResolution table
  updating, 6-28, 9-107
REPRESOLUTION view, 10-14
REPRESOLUTION_METHOD view, 10-15
RepResolution_Statistics table
  purging, 6-17, 9-152
REPRESOLUTION_STATISTICS view, 10-16
RepResolution_Statistics view
  gathering statistics, 6-16
RepSchema table
  updating, 6-28
RepSite view
  updating, 9-105
REPSITES view, 3-30, 10-17
resolution methods
  additive prebuilt, 5-16
  average prebuilt, 5-16
  discard, 5-19
  earliest timestamp, 5-18
  for column groups
    ordering, 5-15
  for conflicts
    removing, 5-15
  latest timestamp, 5-18
  managing, 5-15
  maximum value method, 5-17
  minimum value method, 5-17
  overwrite, 5-19
resolution statistics
  gathering, 6-16
  viewing, 6-16
restrictions
  procedural replication, 7-2
RESUME_MASTER_ACTIVITY procedure
  DBMS_REPCAT package, 3-19
RESUME_SUBSET_OF_MASTERS procedure
  DBMS_OFFLINE_OG package, 7-16
resuming replication activity, 9-162
rollback segments
  setting for snapshot, 2-27
  setup wizard, 3-12
ROWID snapshots, 1-12, 2-43
row-level replication, 1-20
  detecting conflicts, 1-26, 5-5
rows
  identifying during conflict detection, 5-6
RPC, 1-22

## S

SCHEDULE_EXECUTION procedure
  DBMS_DEFER_SYS package, 3-10, 3-11, 3-13, 3-14
SCHEDULE_PUSH procedure
  DBMS_DEFER_SYS package, 3-10, 3-11, 3-13, 3-14
scheduled links
  continuous pushes, 3-11, 3-13
  creating, 3-9
    snapshot site, 4-7
  database link to use, 3-9
  deleting, 3-12
  editing, 3-11
  guidelines, 3-10
  managing, 3-9
  number of background processes used, 3-10
  parallel propagation, 3-10, 3-31 to 3-32
  periodic pushes, 3-11
  serial propagation, 3-10, 3-31
  viewing status, 3-12
Scheduled Links folder
  Replication Manager, 3-12
scheduled purges
  editing, 3-14
  guidelines, 3-13
  periodic purges, 3-14
Scheduling folder
  Replication Manager, 3-12
Scheduling page
  Create Refresh Group property sheet, 4-22
schema
  creating for snapshots, 2-4
  for snapshot sites, 2-2, 2-4

Schema Manager
Create Refresh Group property sheet, 2-26
Create Snapshot Log property sheet, 2-8
Create Snapshot property sheet, 2-14, 2-15
creating snapshots with, 2-7, 2-10
snapshot log creation, 2-7
Snapshot Log property sheet, 2-35
schemas
creating for multimaster replication, 3-6
creating for snapshot sites, 4-7
security, A-4
for multimaster replication, 7-18
for snapshot replication, 7-22
setup
alternative, 7-24
Select from Master Schemas list
setup wizard, 4-7
self-referential constraints in tables
adding sites with to master groups, 3-25
Send Us Your Comments
boilerplate, xxiii
serial propagation, 1-22
scheduled links, 3-10
serialization
of transactions, 7-3
Set Alternate Key Columns dialog
Replication Manager, 3-16, 3-20
Set Date dialog
Replication Manager, 3-9, 3-12
Set Interval dialog
Replication Manager, 3-9, 3-12
Set Scheduled Link dialog
Replication Manager, 3-9
SET_COLUMNS procedure
DBMS_REPCAT package, 3-20
set_disabled, 9-40
SET_DISABLED procedure
DBMS_DEFER_SYS package, 3-11, 3-13, 3-14
setup wizard
configuring snapshot sites, 4-3
Create Master Sites page, 3-24
Create Scheduled Link property sheet, 3-11, 3-13
creating
administrator accounts, 3-5, 4-7
propagator accounts, 3-5, 4-7
receiver accounts, 3-5
Customize Master Site property sheet, 3-7
Customize Snapshot property sheet, 4-8
Edit DB Connection property sheet, 3-13, 3-14
Edit Master Group property sheet, 3-25
Edit Scheduled Link property sheet, 3-11
Finish page, 3-7
New Master Site dialog, 3-5
New Replicated Object Schema dialog, 3-6
Replication Manager, 3-4
Select from Master Schemas list, 4-7
shadow column groups, 1-27, 5-7
shared ownership, 1-25
simple snapshots
refresh groups
creating, 2-24
SQL restrictions, 2-13
structure, 1-7
simple subquery snapshots
many-to-many references, 2-20, 2-23
many-to-one references, 2-19, 2-23
requirements, 2-9
tuning, 2-41
site ownership
dynamic, 5-4
primary, 5-4
site priority
altering, 9-95
as a backup method during timestamp conflict resolution, 5-18
managing, 5-28
site priority groups, 5-28
adding members to, 5-29, 9-81
altering members
priorities, 5-29
values, 5-30
creating, 5-28
syntax, 9-121
dropping, 5-30, 9-133
dropping members, 5-30
removing members from, 9-134
using, 5-28
sites
dropping by priority level, 5-30

snapshot group wizard
Replication Manager, 4-3
snapshot groups
choosing master sites, 4-4
choosing objects to replicate, 4-4
managing, 4-12 to 4-16
selecting master group, 4-4
Snapshot Log property sheet, 2-28
General page, 2-35
Options page, 2-28
Storage page, 2-28
snapshot logs
altering, 2-28, 4-11
privileges required, 2-28
contents of, 1-9
creating, 2-3, 2-6, 4-3, 4-11
privileges required, 2-8
deleting, 4-12
deleting rows
privileges required, 2-30
dropping, 2-33
privileges required, 2-33
filter columns
adding, 2-28
internal operations, 2-10
managing space, 2-28
master table
dropping, 2-33
purging, 2-29, 9-189
naming, 2-8
purging, 2-29
manual, 2-29
privileges required, 2-29
reducing space allocated to, 2-30
reorganizing master tables with, 2-30
storage parameters, 2-8
truncating, 2-30
privileges required, 2-30
truncating master table with, 2-31
underlying table for, 1-10
when to create, 2-8
snapshot refresh groups
adding members, 2-38
altering settings, 2-38
creating, 2-3
deleting, 2-38
deleting members, 2-38
managing, 2-5 to 2-40
manual refresh, 2-39
snapshot refreshes, 4-9
complete, 1-9
fast, 1-9
group
refresh interval, 1-10
groups, 1-10
snapshot logs and, 1-9
snapshot registration at master sites, A-3
snapshot site replication, 4-1 to 4-24
snapshot sites
account configurations, 4-7
adding to advanced replication
environments, 4-2 to 4-10
advanced
alternative security setup, 7-24
changing masters, 9-168
configuration, 4-3
creating
syntax, 9-113
creating refresh groups for, 4-22
customizing settings, 4-8
database links for, 2-2, 2-5
designing, 4-2
dropping, 9-135
offline instantiation
advanced replication, 7-17
preparing for replication, 4-5
propagating changes to master, 9-37
refreshing
syntax, 9-153
replication administrator, 4-5
replication propagator, 4-5
reviewing settings, 4-8
scheduled links for, 3-9
guidelines, 3-10
scheduled purge for
guidelines, 3-13
schema creation for, 4-7
schema for, 2-2
SNP background processes for, 4-10
specifying master site for, 4-6

snapshot sites and migration, B-5
snapshots, 2-5
altering
privileges required, 2-36
altering definition, 4-21
assigning to refresh groups, 2-3
cloning
offline instantiation, 7-14
cloning for basic replication, 7-15
clustered, 2-15
complex, 1-11, 2-41
value for PCTFREE, 2-43
value for PCTUSED, 2-43
creating, 2-3, 4-17
privileges required, 2-11
with subquery, 4-20
with WHERE clause, 4-18
creating schema for, 2-4
creating synonyms based on, 2-35
creating views based on, 2-35
creating with subqueries, 2-16
data dictionary views for, 10-25 to 10-31
data load options, 2-15
datatypes supported, 4-18
defining query, 1-7, 2-12
deleting from snapshot groups, 4-21
dropping
privileges required, 2-37
dropping master table for, 2-36
editing, 4-20
editing storage settings, 4-20
enabling/disabling replication for, 7-35
granting access, 2-3
granting access to, 4-4
group, 1-19
groups
advanced management of, 6-2
indexing, 2-40
managing, 4-16 to 4-21
managing base table, 4-21
naming, 2-11
offline instantiation of, 9-51, 9-53
performance tuning, 2-40
primary key, 1-12, 2-12, A-3
requirements for creating log, 2-8
read-only, 1-6
altering, 2-36
base table, 1-7
dropping, 2-36
index, 1-8
managing, 2-34 to 2-37
privileges required to query, 2-34
registration, 2-35, 2-36
simple with subqueries, A-2
unregistering, 2-36
using, 2-34
view, 1-8
referencing remote data, 2-13
refresh, 1-9, 1-11
failures, 2-39
individual snapshot, 2-44
problems with, 6-26
querying for last refresh time, 2-35
retries, 2-39
rollback segment for, 2-27
troubleshooting, 2-39, 2-40
refresh groups, 4-22
adding members, 2-38
altering settings, 2-38
creating, 2-24
data dictionary views, 10-25
deleting, 2-38
deleting members, 2-38
designing, 2-5
manual refresh, 2-39
refresh setting, 2-12
refresh types, 2-27
refreshing, 9-190
replication
security for, 7-22
requirements for fast refresh, 2-17
ROWID, 2-43
rowid, 1-12
simple
structure, 1-7
simple subquery
advanced subsetting, 2-16
AND expression and, 2-24
examples, 2-16 to 2-21
EXISTS clause and, 2-24

filter columns requirement, 2-9, 2-24
joins and, 2-24
number of columns in master tables, 2-24
requirements, 2-9
site, 1-19
sites
changing masters, 6-3
schema for, 2-4
SNAPSHOTS view, 10-26
SQL
restrictions for simple, 2-13
storage options, 2-14
subquery, 1-7, 2-16
many-to-many references, 2-20
many-to-one references, 2-19
subquery snapshots, restrictions for, 2-23
trace file, 2-40
troubleshooting, 6-26
tuning, 2-41
underlying objects for, 1-7
updatable, 1-15
deferred constraints, A-4
propagation mechanism, 1-23
updating, 1-9
upgrading to primary key, B-10
using PL/SQL triggers with base tables, 4-21
viewing information about, 2-35
Snapshots page
Create Refresh Group property sheet, 4-22
SNAPSHOTS view, 10-26
SNAPSHOTS_LOGS view, 10-28
SNP background processes, 3-8, 4-10
basic replication
environment, 2-6
for automatic snapshot refresh, 1-11
snapshot sites, 4-10
space
reducing snapshot log, 2-30
SQL
commands
snapshot creation, 2-16 to 2-21
restrictions for simple snapshots, 2-13
static ownership, 1-25
statistics
auditing conflict resolution, 6-16
collecting, 9-156
deleting, 6-17
for conflict resolution, 6-16
status
propagation, 9-24
Stop on Error setting
Create New Scheduled Link property sheet, 3-10
storage options
PCTFREE, 2-9
PCTUSED, 2-9
Storage page
Create Snapshot Property Sheet, 2-14
Snapshot Log property sheet, 2-28
storage parameters
snapshot log
altering, 2-28
snapshot logs, 2-8
snapshots, 2-14
storage settings
editing for snapshots, 4-20
store-and-forward replication, 1-21
subqueries for snapshots, A-2
subquery
creating snapshots with, 4-20
subquery snapshots, 1-7, 2-16
advanced subsetting, 2-16
AND expression and, 2-24
creating, 4-20
EXIST clause and, 2-24
filter columns requirement, 2-9, 2-24
joins for, 2-24
many-to-many references, 2-20, 2-23
many-to-one references, 2-19, 2-23
number of columns in master tables, 2-24
requirements, 2-9
tuning, 2-41
SUBTRACT procedure
DBMS_REFRESH package, 2-38
survivability, 7-11
design considerations, 7-13
Parallel Server and, 7-12
SUSPEND_MASTER_ACTIVITY procedure
DBMS_REPCAT package, 3-18
SWITCH_SNAPSHOT_MASTER procedure

DBMS_REPCAT package
ALTER_SNAPSHOT_PROPAGATION
procedure, 4-16
synchronous data propagation, 1-28, 7-6
synchronously replicated transactions
destination of, 7-7
synonyms
creating, based on snapshot, 2-35
system-based security, A-4

## T

table data
replicating to master sites, 3-22
tables
comparing, 9-55
intersection, 2-20 to 2-23
master groups
generating replication support for, 3-28
modifying
without replicating changes, 7-32
partitioned
replicating, 3-21
partitioned and indexes, A-4
problems generating replication support
for, 6-24
rectifying, 6-18, 9-59
reorg, A-4
Tablespace and Extents page
Create Snapshot Log property sheet, 4-11
Edit Snapshot Log property sheet, 4-11
Edit Snapshot property sheet, 4-20
timestamp conflict resolution methods
sample trigger, 5-31
token passing, 7-28
sample implementation, 7-27
trace file
snapshots, 2-40
transactions
deferred
deleting, 6-10
executing, 6-9
error
displaying, 6-11
propagation
protection mechanisms, 3-33 to 3-34
queue for
deferred, diagnosing problems with, 6-25
serialization of, 7-3
triggers
internal, A-2
replicating, 7-34
troubleshooting, 6-22
TRUNCATE command, 2-30
TRUNCATE TABLE command
PRESERVE SNAPSHOT LOG option, 2-31
PURGE SNAPSHOT LOG option, 2-31

## U

Ungrouped Snapshots folder, 4-22
UNIQUE constraint
simple subquery snapshots and, 2-23
uniqueness conflicts, 1-24, 5-3
avoiding, 5-5
resolution, 5-8
configuring, 5-33
resolution methods, 5-9
assigning, 5-33
removing, 5-33
resolving, 5-47
UNREGISTER_SNAPSHOT procedure
DBMS_SNAPSHOT package, 2-36
UNSCHEDULE_PUSH procedure
DBMS_DEFER_SYS package, 3-12
updatable snapshots, 1-15
generating replication support for, 4-20
propagation mechanism, 1-23
properties of, 1-15
update conflicts, 1-24, 5-3
avoiding, 5-5
resolution
and column groups, 5-7
using site priority for, 5-28
resolution methods
assigning, 5-15
prebuilt, 5-8
resolution of
minimizing data propagation for, 5-40
resolving, 5-47

updates
and min_communication, 5-41
minimizing communication, 5-41
updating
Comments, 6-28
replication tables, 6-28
USER_REFRESH view, 2-40
USER_REFRESH_CHILDREN view, 2-40
user-defined
notification methods, 5-50
user-defined conflict resolution method, 5-46

## V

VALIDATE procedure, A-4
View Database Destination property sheet, 6-10
views
creating, based on snapshot, 2-35
DBA_REFRESH, 2-44
DBA_REFRESH_CHILDREN, 2-44
DBA_REGISTERED_SNAPSHOTS, 2-35, 2-44
DBA_RGROUP, 2-44
DBA_SNAPSHOT_LOGS, 2-35, 2-44
DBA_SNAPSHOTS, 2-44
DEFCALL, 10-20
DEFCALLDEST, 10-20
DEFDEFAULTDEST, 10-21
DEFERRCOUNT, 10-21
DEFERROR, 10-22
DEFLOB, 10-22
DEFPROPAGATOR, 10-23
DEFSCHEDULE, 10-23
DEFTRAN, 10-24
DEFTRANDEST, 10-24
REFRESH, 10-30
REFRESH_CHILDREN, 10-31
REGISTERED_SNAPSHOTS, 10-27
REPCATLOG, 3-30, 6-6 to 6-7, 10-5
REPCOLUMN, 10-6
REPCOLUMN_GROUP, 10-6
REPCONFLICT, 10-7
REPDDL, 10-7
REPGENERATED, 10-8
REPGENOBJECTS, 10-18
REPGROUP, 3-30, 10-3
REPGROUPED_COLUMN, 10-9
REPKEY_COLUMNS, 10-9
replication catalog, 10-2
REPOBJECT, 3-30, 10-10
REPPARAMETER_COLUMN, 10-11
REPPRIORITY, 10-12
REPPRIORITY_GROUP, 10-13
REPPROP, 10-13
REPRESOL_STATS_CONTROL, 10-15
REPRESOLUTION, 10-14
REPRESOLUTION_METHOD, 10-15
REPRESOLUTION_STATISTICS, 10-16
REPSITES, 3-30, 10-17
SNAPSHOT_REFRESH_TIMES
SNAPSHOT_REFRESH_TIMES view, 10-29
SNAPSHOTS, 10-26
SNAPSHOTS_LOGS, 10-28
USER_REFRESH, 2-40
USER_REFRESH_CHILDREN, 2-40

## W

WHERE clause
creating updatable snapshots with
updatable snapshots
creating with WHERE clause, 4-18
workflow, 7-27
wrapper
procedural replication, 1-28